

PLASMA

2.3.0

Generated by Doxygen 1.6.3

Tue Nov 30 18:44:57 2010

Contents

1	Module Index	1
1.1	Modules	1
2	Data Type Index	3
2.1	Data Types List	3
3	Module Documentation	5
3.1	Simple Interface - Double Complex	5
3.1.1	Detailed Description	7
3.1.2	Function/Subroutine Documentation	7
3.1.2.1	PLASMA_zcgels	7
3.1.2.2	PLASMA_zcgesv	8
3.1.2.3	PLASMA_zcposv	9
3.1.2.4	PLASMA_zcungesv	11
3.1.2.5	PLASMA_zgelqf	12
3.1.2.6	PLASMA_zgelqs	13
3.1.2.7	PLASMA_zgels	14
3.1.2.8	PLASMA_zgemm	15
3.1.2.9	PLASMA_zgeqrf	16
3.1.2.10	PLASMA_zgeqrs	16
3.1.2.11	PLASMA_zgesv	17
3.1.2.12	PLASMA_zgetrf	18
3.1.2.13	PLASMA_zgetrs	18
3.1.2.14	PLASMA_zhemm	19
3.1.2.15	PLASMA_zher2k	20
3.1.2.16	PLASMA_zherk	22
3.1.2.17	PLASMA_zlange	22
3.1.2.18	PLASMA_zlanhe	23
3.1.2.19	PLASMA_zlansy	24

3.1.2.20	PLASMA_zLapack_to_Tile	25
3.1.2.21	PLASMA_zlauum	25
3.1.2.22	PLASMA_zplghe	26
3.1.2.23	PLASMA_zplgsy	26
3.1.2.24	PLASMA_zplrnt	27
3.1.2.25	PLASMA_zposv	28
3.1.2.26	PLASMA_zpotrf	29
3.1.2.27	PLASMA_zpotri	29
3.1.2.28	PLASMA_zpotrs	30
3.1.2.29	PLASMA_zsymm	31
3.1.2.30	PLASMA_zsyr2k	32
3.1.2.31	PLASMA_zsyrk	33
3.1.2.32	PLASMA_zTile_to_Lapack	34
3.1.2.33	PLASMA_ztrmm	34
3.1.2.34	PLASMA_ztrsm	35
3.1.2.35	PLASMA_ztrsmpl	36
3.1.2.36	PLASMA_ztrtri	37
3.1.2.37	PLASMA_zunglq	38
3.1.2.38	PLASMA_zungqr	38
3.1.2.39	PLASMA_zunmlq	39
3.1.2.40	PLASMA_zunmqr	40
3.2	Simple Interface - Single Complex	41
3.2.1	Detailed Description	42
3.2.2	Function/Subroutine Documentation	42
3.2.2.1	PLASMA_cgelqf	42
3.2.2.2	PLASMA_cgelqs	43
3.2.2.3	PLASMA_cgels	44
3.2.2.4	PLASMA_cgemm	45
3.2.2.5	PLASMA_cgeqrf	46
3.2.2.6	PLASMA_cgeqrs	46
3.2.2.7	PLASMA_cgesv	47
3.2.2.8	PLASMA_cgetrf	48
3.2.2.9	PLASMA_cgetrs	49
3.2.2.10	PLASMA_chemm	49
3.2.2.11	PLASMA_cher2k	51
3.2.2.12	PLASMA_cherk	52

3.2.2.13	PLASMA_clange	52
3.2.2.14	PLASMA_clanhe	53
3.2.2.15	PLASMA_clansy	54
3.2.2.16	PLASMA_cLapack_to_Tile	55
3.2.2.17	PLASMA_clauum	55
3.2.2.18	PLASMA_cplghe	56
3.2.2.19	PLASMA_cplgsy	56
3.2.2.20	PLASMA_cplrnt	57
3.2.2.21	PLASMA_cposv	58
3.2.2.22	PLASMA_cpotrf	59
3.2.2.23	PLASMA_cpotri	59
3.2.2.24	PLASMA_cpotrs	60
3.2.2.25	PLASMA_csymm	61
3.2.2.26	PLASMA_csy2k	62
3.2.2.27	PLASMA_csyrk	63
3.2.2.28	PLASMA_cTile_to_Lapack	64
3.2.2.29	PLASMA_ctrmm	64
3.2.2.30	PLASMA_ctrsm	65
3.2.2.31	PLASMA_ctrsmpl	66
3.2.2.32	PLASMA_ctrtri	67
3.2.2.33	PLASMA_cunglq	68
3.2.2.34	PLASMA_cungqr	68
3.2.2.35	PLASMA_cunmlq	69
3.2.2.36	PLASMA_cunmqr	70
3.3	Simple Interface - Double Real	71
3.3.1	Detailed Description	72
3.3.2	Function/Subroutine Documentation	72
3.3.2.1	PLASMA_dgelqf	72
3.3.2.2	PLASMA_dgelqs	72
3.3.2.3	PLASMA_dgels	73
3.3.2.4	PLASMA_dgemm	74
3.3.2.5	PLASMA_dgeqrf	75
3.3.2.6	PLASMA_dgeqrs	76
3.3.2.7	PLASMA_dgesv	77
3.3.2.8	PLASMA_dgetrf	77
3.3.2.9	PLASMA_dgetrs	78

3.3.2.10	PLASMA_dlange	79
3.3.2.11	PLASMA_dlansy	80
3.3.2.12	PLASMA_dLapack_to_Tile	80
3.3.2.13	PLASMA_dlauum	81
3.3.2.14	PLASMA_dorglq	82
3.3.2.15	PLASMA_dorgqr	82
3.3.2.16	PLASMA_dormlq	83
3.3.2.17	PLASMA_dormqr	84
3.3.2.18	PLASMA_dplgsy	85
3.3.2.19	PLASMA_dplmt	85
3.3.2.20	PLASMA_dposv	86
3.3.2.21	PLASMA_dpotrf	87
3.3.2.22	PLASMA_dpotri	88
3.3.2.23	PLASMA_dpotrs	88
3.3.2.24	PLASMA_dsgeev	89
3.3.2.25	PLASMA_dsposv	90
3.3.2.26	PLASMA_dsungesv	91
3.3.2.27	PLASMA_dsymm	93
3.3.2.28	PLASMA_dsyr2k	94
3.3.2.29	PLASMA_dsyrf	95
3.3.2.30	PLASMA_dTile_to_Lapack	96
3.3.2.31	PLASMA_dtrmm	96
3.3.2.32	PLASMA_dtrsm	97
3.3.2.33	PLASMA_dtrsmpl	98
3.3.2.34	PLASMA_dtrtri	99
3.4	Simple Interface - Single Real	100
3.4.1	Detailed Description	100
3.4.2	Function/Subroutine Documentation	101
3.4.2.1	PLASMA_sgelqf	101
3.4.2.2	PLASMA_sgelqs	101
3.4.2.3	PLASMA_sgels	102
3.4.2.4	PLASMA_sgemm	103
3.4.2.5	PLASMA_sgeqrf	104
3.4.2.6	PLASMA_sgeqrs	105
3.4.2.7	PLASMA_sgesv	105
3.4.2.8	PLASMA_sgetrf	106

3.4.2.9	PLASMA_sgetrs	107
3.4.2.10	PLASMA_slange	108
3.4.2.11	PLASMA_slansy	108
3.4.2.12	PLASMA_sLapack_to_Tile	109
3.4.2.13	PLASMA_slauum	110
3.4.2.14	PLASMA_sorglq	110
3.4.2.15	PLASMA_sorgqr	111
3.4.2.16	PLASMA_sormlq	112
3.4.2.17	PLASMA_sormqr	112
3.4.2.18	PLASMA_splgsy	113
3.4.2.19	PLASMA_splrnt	114
3.4.2.20	PLASMA_sposv	114
3.4.2.21	PLASMA_spotrf	115
3.4.2.22	PLASMA_spotri	116
3.4.2.23	PLASMA_spotrs	117
3.4.2.24	PLASMA_ssymm	117
3.4.2.25	PLASMA_ssy2k	119
3.4.2.26	PLASMA_ssyk	120
3.4.2.27	PLASMA_sTile_to_Lapack	120
3.4.2.28	PLASMA_strmm	121
3.4.2.29	PLASMA_strsm	122
3.4.2.30	PLASMA_strsmpl	123
3.4.2.31	PLASMA_strtri	124
3.5	Advanced Interface: Synchronous - Double Complex	125
3.5.1	Detailed Description	126
3.5.2	Function/Subroutine Documentation	126
3.5.2.1	PLASMA_zcgels_Tile	126
3.5.2.2	PLASMA_zcgesv_Tile	127
3.5.2.3	PLASMA_zcposv_Tile	128
3.5.2.4	PLASMA_zcungesv_Tile	128
3.5.2.5	PLASMA_zgelqf_Tile	129
3.5.2.6	PLASMA_zgelqs_Tile	130
3.5.2.7	PLASMA_zgels_Tile	130
3.5.2.8	PLASMA_zgemm_Tile	131
3.5.2.9	PLASMA_zgeqrf_Tile	132
3.5.2.10	PLASMA_zgeqrs_Tile	132

3.5.2.11	PLASMA_zgesv_Tile	133
3.5.2.12	PLASMA_zgetrf_Tile	133
3.5.2.13	PLASMA_zgetrs_Tile	134
3.5.2.14	PLASMA_zhemm_Tile	135
3.5.2.15	PLASMA_zher2k_Tile	135
3.5.2.16	PLASMA_zherk_Tile	136
3.5.2.17	PLASMA_zlange_Tile	137
3.5.2.18	PLASMA_zlanhe_Tile	137
3.5.2.19	PLASMA_zlansy_Tile	138
3.5.2.20	PLASMA_zlauum_Tile	138
3.5.2.21	PLASMA_zplghe_Tile	139
3.5.2.22	PLASMA_zplgsy_Tile	139
3.5.2.23	PLASMA_zplrnt_Tile	140
3.5.2.24	PLASMA_zposv_Tile	140
3.5.2.25	PLASMA_zpotrf_Tile	141
3.5.2.26	PLASMA_zpotri_Tile	142
3.5.2.27	PLASMA_zpotrs_Tile	142
3.5.2.28	PLASMA_zsymm_Tile	143
3.5.2.29	PLASMA_zsyr2k_Tile	144
3.5.2.30	PLASMA_zsyrk_Tile	144
3.5.2.31	PLASMA_ztrmm_Tile	145
3.5.2.32	PLASMA_ztrsm_Tile	146
3.5.2.33	PLASMA_ztrsml_Tile	147
3.5.2.34	PLASMA_ztrtri_Tile	147
3.5.2.35	PLASMA_zunglq_Tile	148
3.5.2.36	PLASMA_zungqr_Tile	149
3.5.2.37	PLASMA_zunmlq_Tile	149
3.5.2.38	PLASMA_zunmqr_Tile	150
3.6	Advanced Interface: Synchronous - Single Complex	151
3.6.1	Detailed Description	152
3.6.2	Function/Subroutine Documentation	152
3.6.2.1	PLASMA_cgelqf_Tile	152
3.6.2.2	PLASMA_cgelqs_Tile	152
3.6.2.3	PLASMA_cgels_Tile	153
3.6.2.4	PLASMA_cgemm_Tile	154
3.6.2.5	PLASMA_cgeqrf_Tile	154

3.6.2.6	PLASMA_cgeqrs_Tile	155
3.6.2.7	PLASMA_cgesv_Tile	155
3.6.2.8	PLASMA_cgetrf_Tile	156
3.6.2.9	PLASMA_cgetrs_Tile	157
3.6.2.10	PLASMA_chemm_Tile	157
3.6.2.11	PLASMA_cher2k_Tile	158
3.6.2.12	PLASMA_cherk_Tile	159
3.6.2.13	PLASMA_clange_Tile	159
3.6.2.14	PLASMA_clanhe_Tile	160
3.6.2.15	PLASMA_clansy_Tile	160
3.6.2.16	PLASMA_clauum_Tile	161
3.6.2.17	PLASMA_cplghe_Tile	162
3.6.2.18	PLASMA_cplgsy_Tile	162
3.6.2.19	PLASMA_cplrnt_Tile	163
3.6.2.20	PLASMA_cposv_Tile	163
3.6.2.21	PLASMA_cpotrf_Tile	164
3.6.2.22	PLASMA_cpotri_Tile	164
3.6.2.23	PLASMA_cpotrs_Tile	165
3.6.2.24	PLASMA_csymm_Tile	166
3.6.2.25	PLASMA_csyr2k_Tile	166
3.6.2.26	PLASMA_csyrk_Tile	167
3.6.2.27	PLASMA_ctrmm_Tile	168
3.6.2.28	PLASMA_ctrsm_Tile	169
3.6.2.29	PLASMA_ctrsmpl_Tile	169
3.6.2.30	PLASMA_ctrtri_Tile	170
3.6.2.31	PLASMA_cunglq_Tile	171
3.6.2.32	PLASMA_cungqr_Tile	171
3.6.2.33	PLASMA_cunmlq_Tile	172
3.6.2.34	PLASMA_cunmqr_Tile	172
3.7	Advanced Interface: Synchronous - Double Real	174
3.7.1	Detailed Description	175
3.7.2	Function/Subroutine Documentation	175
3.7.2.1	PLASMA_dgelqf_Tile	175
3.7.2.2	PLASMA_dgelqs_Tile	175
3.7.2.3	PLASMA_dgels_Tile	176
3.7.2.4	PLASMA_dgemm_Tile	176

3.7.2.5	PLASMA_dgeqrf_Tile	177
3.7.2.6	PLASMA_dgeqrs_Tile	178
3.7.2.7	PLASMA_dgesv_Tile	178
3.7.2.8	PLASMA_dgetrf_Tile	179
3.7.2.9	PLASMA_dgetrs_Tile	180
3.7.2.10	PLASMA_dlange_Tile	180
3.7.2.11	PLASMA_dlansy_Tile	181
3.7.2.12	PLASMA_dlauum_Tile	181
3.7.2.13	PLASMA_dorglq_Tile	182
3.7.2.14	PLASMA_dorgqr_Tile	182
3.7.2.15	PLASMA_dormlq_Tile	183
3.7.2.16	PLASMA_dormqr_Tile	183
3.7.2.17	PLASMA_dplgsy_Tile	184
3.7.2.18	PLASMA_dplmt_Tile	185
3.7.2.19	PLASMA_dposv_Tile	185
3.7.2.20	PLASMA_dpotrf_Tile	186
3.7.2.21	PLASMA_dpotri_Tile	186
3.7.2.22	PLASMA_dpotrs_Tile	187
3.7.2.23	PLASMA_dsgesv_Tile	188
3.7.2.24	PLASMA_dsposv_Tile	188
3.7.2.25	PLASMA_dsungesv_Tile	189
3.7.2.26	PLASMA_dsymm_Tile	190
3.7.2.27	PLASMA_dsyr2k_Tile	191
3.7.2.28	PLASMA_dsyrk_Tile	191
3.7.2.29	PLASMA_dtrmm_Tile	192
3.7.2.30	PLASMA_dtrsm_Tile	193
3.7.2.31	PLASMA_dtrsml_Tile	193
3.7.2.32	PLASMA_dtrtri_Tile	194
3.8	Advanced Interface: Synchronous - Single Real	196
3.8.1	Detailed Description	196
3.8.2	Function/Subroutine Documentation	197
3.8.2.1	PLASMA_sgelqf_Tile	197
3.8.2.2	PLASMA_sgelqs_Tile	197
3.8.2.3	PLASMA_sgels_Tile	198
3.8.2.4	PLASMA_sgemm_Tile	198
3.8.2.5	PLASMA_sgeqrf_Tile	199

3.8.2.6	PLASMA_sgeqrs_Tile	200
3.8.2.7	PLASMA_sgesv_Tile	200
3.8.2.8	PLASMA_sgetrf_Tile	201
3.8.2.9	PLASMA_sgetrs_Tile	201
3.8.2.10	PLASMA_slange_Tile	202
3.8.2.11	PLASMA_slansy_Tile	203
3.8.2.12	PLASMA_slauum_Tile	203
3.8.2.13	PLASMA_sorglq_Tile	204
3.8.2.14	PLASMA_sorgqr_Tile	204
3.8.2.15	PLASMA_sormlq_Tile	205
3.8.2.16	PLASMA_sormqr_Tile	205
3.8.2.17	PLASMA_splgsy_Tile	206
3.8.2.18	PLASMA_splrnt_Tile	207
3.8.2.19	PLASMA_sposv_Tile	207
3.8.2.20	PLASMA_spotrf_Tile	208
3.8.2.21	PLASMA_spotri_Tile	208
3.8.2.22	PLASMA_spotrs_Tile	209
3.8.2.23	PLASMA_ssymm_Tile	210
3.8.2.24	PLASMA_ssy2k_Tile	210
3.8.2.25	PLASMA_ssyk_Tile	211
3.8.2.26	PLASMA_strmm_Tile	212
3.8.2.27	PLASMA_strsm_Tile	212
3.8.2.28	PLASMA_strsmpl_Tile	213
3.8.2.29	PLASMA_strtri_Tile	214
3.9	Advanced Interface: Asynchronous - Double Complex	215
3.9.1	Detailed Description	216
3.9.2	Function/Subroutine Documentation	217
3.9.2.1	PLASMA_zcgels_Tile_Async	217
3.9.2.2	PLASMA_zcgesv_Tile_Async	217
3.9.2.3	PLASMA_zcposv_Tile_Async	217
3.9.2.4	PLASMA_zcungesv_Tile_Async	218
3.9.2.5	PLASMA_zgelqf_Tile_Async	218
3.9.2.6	PLASMA_zgelqs_Tile_Async	219
3.9.2.7	PLASMA_zgels_Tile_Async	219
3.9.2.8	PLASMA_zgemm_Tile_Async	220
3.9.2.9	PLASMA_zgeqrf_Tile_Async	220

3.9.2.10	PLASMA_zgeqrs_Tile_Async	220
3.9.2.11	PLASMA_zgesv_Tile_Async	221
3.9.2.12	PLASMA_zgetrf_Tile_Async	221
3.9.2.13	PLASMA_zgetrs_Tile_Async	222
3.9.2.14	PLASMA_zhemm_Tile_Async	222
3.9.2.15	PLASMA_zher2k_Tile_Async	223
3.9.2.16	PLASMA_zherk_Tile_Async	223
3.9.2.17	PLASMA_zlange_Tile_Async	223
3.9.2.18	PLASMA_zlanhe_Tile_Async	224
3.9.2.19	PLASMA_zlansy_Tile_Async	224
3.9.2.20	PLASMA_zLapack_to_Tile_Async	225
3.9.2.21	PLASMA_zlauum_Tile_Async	225
3.9.2.22	PLASMA_zplghe_Tile_Async	225
3.9.2.23	PLASMA_zplgsy_Tile_Async	226
3.9.2.24	PLASMA_zplrnt_Tile_Async	226
3.9.2.25	PLASMA_zposv_Tile_Async	227
3.9.2.26	PLASMA_zpotrf_Tile_Async	227
3.9.2.27	PLASMA_zpotri_Tile_Async	228
3.9.2.28	PLASMA_zpotrs_Tile_Async	228
3.9.2.29	PLASMA_zsymm_Tile_Async	228
3.9.2.30	PLASMA_zsyr2k_Tile_Async	229
3.9.2.31	PLASMA_zsyrk_Tile_Async	229
3.9.2.32	PLASMA_zTile_to_Lapack_Async	230
3.9.2.33	PLASMA_ztrmm_Tile_Async	230
3.9.2.34	PLASMA_ztrsm_Tile_Async	231
3.9.2.35	PLASMA_ztrsml_Tile_Async	231
3.9.2.36	PLASMA_ztrtri_Tile_Async	231
3.9.2.37	PLASMA_zunglq_Tile_Async	232
3.9.2.38	PLASMA_zungqr_Tile_Async	232
3.9.2.39	PLASMA_zunmlq_Tile_Async	233
3.9.2.40	PLASMA_zunmqr_Tile_Async	233
3.10	Advanced Interface: Asynchronous - Single Complex	234
3.10.1	Detailed Description	235
3.10.2	Function/Subroutine Documentation	235
3.10.2.1	PLASMA_cgelqf_Tile_Async	235
3.10.2.2	PLASMA_cgelqs_Tile_Async	236

3.10.2.3	PLASMA_cgels_Tile_Async	236
3.10.2.4	PLASMA_cgemm_Tile_Async	237
3.10.2.5	PLASMA_cgeqrf_Tile_Async	237
3.10.2.6	PLASMA_cgeqrs_Tile_Async	237
3.10.2.7	PLASMA_cgesv_Tile_Async	238
3.10.2.8	PLASMA_cgetrf_Tile_Async	238
3.10.2.9	PLASMA_cgetrs_Tile_Async	239
3.10.2.10	PLASMA_chemm_Tile_Async	239
3.10.2.11	PLASMA_cher2k_Tile_Async	240
3.10.2.12	PLASMA_cherk_Tile_Async	240
3.10.2.13	PLASMA_clange_Tile_Async	240
3.10.2.14	PLASMA_clanhe_Tile_Async	241
3.10.2.15	PLASMA_clansy_Tile_Async	241
3.10.2.16	PLASMA_cLapack_to_Tile_Async	242
3.10.2.17	PLASMA_clauum_Tile_Async	242
3.10.2.18	PLASMA_cplghe_Tile_Async	242
3.10.2.19	PLASMA_cplgsy_Tile_Async	243
3.10.2.20	PLASMA_cplrnt_Tile_Async	243
3.10.2.21	PLASMA_cposv_Tile_Async	244
3.10.2.22	PLASMA_cpotrf_Tile_Async	244
3.10.2.23	PLASMA_cpotri_Tile_Async	245
3.10.2.24	PLASMA_cpotrs_Tile_Async	245
3.10.2.25	PLASMA_csymm_Tile_Async	245
3.10.2.26	PLASMA_csyr2k_Tile_Async	246
3.10.2.27	PLASMA_csyk_Tile_Async	246
3.10.2.28	PLASMA_cTile_to_Lapack_Async	247
3.10.2.29	PLASMA_ctrmm_Tile_Async	247
3.10.2.30	PLASMA_ctrsm_Tile_Async	248
3.10.2.31	PLASMA_ctrsmpl_Tile_Async	248
3.10.2.32	PLASMA_ctrtri_Tile_Async	248
3.10.2.33	PLASMA_cunglq_Tile_Async	249
3.10.2.34	PLASMA_cungqr_Tile_Async	249
3.10.2.35	PLASMA_cunmlq_Tile_Async	250
3.10.2.36	PLASMA_cunmqr_Tile_Async	250
3.11	Advanced Interface: Asynchronous - Double Real	251
3.11.1	Detailed Description	252

3.11.2	Function/Subroutine Documentation	252
3.11.2.1	PLASMA_dgelqf_Tile_Async	252
3.11.2.2	PLASMA_dgelqs_Tile_Async	253
3.11.2.3	PLASMA_dgels_Tile_Async	253
3.11.2.4	PLASMA_dgemm_Tile_Async	254
3.11.2.5	PLASMA_dgeqrf_Tile_Async	254
3.11.2.6	PLASMA_dgeqrs_Tile_Async	254
3.11.2.7	PLASMA_dgesv_Tile_Async	255
3.11.2.8	PLASMA_dgetrf_Tile_Async	255
3.11.2.9	PLASMA_dgetrs_Tile_Async	256
3.11.2.10	PLASMA_dlange_Tile_Async	256
3.11.2.11	PLASMA_dlansy_Tile_Async	257
3.11.2.12	PLASMA_dLapack_to_Tile_Async	257
3.11.2.13	PLASMA_dlauum_Tile_Async	257
3.11.2.14	PLASMA_dorglq_Tile_Async	258
3.11.2.15	PLASMA_dorgqr_Tile_Async	258
3.11.2.16	PLASMA_dormlq_Tile_Async	259
3.11.2.17	PLASMA_dormqr_Tile_Async	259
3.11.2.18	PLASMA_dplgsy_Tile_Async	259
3.11.2.19	PLASMA_dplmt_Tile_Async	260
3.11.2.20	PLASMA_dposv_Tile_Async	260
3.11.2.21	PLASMA_dpotrf_Tile_Async	261
3.11.2.22	PLASMA_dpotri_Tile_Async	261
3.11.2.23	PLASMA_dpotrs_Tile_Async	262
3.11.2.24	PLASMA_dsgeev_Tile_Async	262
3.11.2.25	PLASMA_dsposv_Tile_Async	262
3.11.2.26	PLASMA_dsungesv_Tile_Async	263
3.11.2.27	PLASMA_dsymm_Tile_Async	263
3.11.2.28	PLASMA_dsyr2k_Tile_Async	264
3.11.2.29	PLASMA_dsyrk_Tile_Async	264
3.11.2.30	PLASMA_dTile_to_Lapack_Async	264
3.11.2.31	PLASMA_dtrmm_Tile_Async	265
3.11.2.32	PLASMA_dtrsm_Tile_Async	265
3.11.2.33	PLASMA_dtrsmpl_Tile_Async	266
3.11.2.34	PLASMA_dtrtri_Tile_Async	266
3.12	Advanced Interface: Asynchronous - Single Real	267

3.12.1	Detailed Description	268
3.12.2	Function/Subroutine Documentation	268
3.12.2.1	PLASMA_sgelqf_Tile_Async	268
3.12.2.2	PLASMA_sgelqs_Tile_Async	269
3.12.2.3	PLASMA_sgels_Tile_Async	269
3.12.2.4	PLASMA_sgemm_Tile_Async	269
3.12.2.5	PLASMA_sgeqrf_Tile_Async	270
3.12.2.6	PLASMA_sgeqrs_Tile_Async	270
3.12.2.7	PLASMA_sgesv_Tile_Async	271
3.12.2.8	PLASMA_sgetrf_Tile_Async	271
3.12.2.9	PLASMA_sgetrs_Tile_Async	272
3.12.2.10	PLASMA_slange_Tile_Async	272
3.12.2.11	PLASMA_slansy_Tile_Async	272
3.12.2.12	PLASMA_sLapack_to_Tile_Async	273
3.12.2.13	PLASMA_slauum_Tile_Async	273
3.12.2.14	PLASMA_sorglq_Tile_Async	274
3.12.2.15	PLASMA_sorgqr_Tile_Async	274
3.12.2.16	PLASMA_sormlq_Tile_Async	274
3.12.2.17	PLASMA_sormqr_Tile_Async	275
3.12.2.18	PLASMA_splgsy_Tile_Async	275
3.12.2.19	PLASMA_splrnt_Tile_Async	276
3.12.2.20	PLASMA_sposv_Tile_Async	276
3.12.2.21	PLASMA_spotrf_Tile_Async	277
3.12.2.22	PLASMA_spotri_Tile_Async	277
3.12.2.23	PLASMA_spotrs_Tile_Async	277
3.12.2.24	PLASMA_ssymm_Tile_Async	278
3.12.2.25	PLASMA_ssyr2k_Tile_Async	278
3.12.2.26	PLASMA_ssyrik_Tile_Async	279
3.12.2.27	PLASMA_sTile_to_Lapack_Async	279
3.12.2.28	PLASMA_strmm_Tile_Async	280
3.12.2.29	PLASMA_strsm_Tile_Async	280
3.12.2.30	PLASMA_strsmpl_Tile_Async	280
3.12.2.31	PLASMA_strtri_Tile_Async	281
4	Data Type Documentation	283
4.1	plasma_context_map_struct Type Reference	283
4.1.1	Detailed Description	283

4.2	plasma_context_struct Type Reference	284
4.2.1	Detailed Description	284
4.3	primedec Type Reference	285
4.4	pthread_cond_s Type Reference	286
4.5	pthread_s Type Reference	287

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Simple Interface - Double Complex	5
Simple Interface - Single Complex	41
Simple Interface - Double Real	71
Simple Interface - Single Real	100
Advanced Interface: Synchronous - Double Complex	125
Advanced Interface: Synchronous - Single Complex	151
Advanced Interface: Synchronous - Double Real	174
Advanced Interface: Synchronous - Single Real	196
Advanced Interface: Asynchronous - Double Complex	215
Advanced Interface: Asynchronous - Single Complex	234
Advanced Interface: Asynchronous - Double Real	251
Advanced Interface: Asynchronous - Single Real	267

Chapter 2

Data Type Index

2.1 Data Types List

Here are the data types with brief descriptions:

plasma_context_map_struct	283
plasma_context_struct	284
primedec	285
pthread_cond_s	286
pthread_s	287

Chapter 3

Module Documentation

3.1 Simple Interface - Double Complex

Functions/Subroutines

- int [PLASMA_zcgels](#) (PLASMA_enum trans, int M, int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB, PLASMA_Complex64_t *X, int LDX, int *ITER)
- int [PLASMA_zcgesv](#) (int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB, PLASMA_Complex64_t *X, int LDX, int *ITER)
- int [PLASMA_zcposv](#) (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB, PLASMA_Complex64_t *X, int LDX, int *ITER)
- int [PLASMA_zcungesv](#) (PLASMA_enum trans, int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB, PLASMA_Complex64_t *X, int LDX, int *ITER)
- int [PLASMA_zgelqf](#) (int M, int N, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *T)
- int [PLASMA_zgelqs](#) (int M, int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *T, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_zgels](#) (PLASMA_enum trans, int M, int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *T, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_zgemm](#) (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t *C, int LDC)
- int [PLASMA_zgeqrf](#) (int M, int N, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *T)
- int [PLASMA_zgeqrs](#) (int M, int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *T, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_zgesv](#) (int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *L, int *IPIV, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_zgetrf](#) (int M, int N, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *L, int *IPIV)
- int [PLASMA_zgetrs](#) (PLASMA_enum trans, int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *L, int *IPIV, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_zhemm](#) (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex64_t alpha, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t *C, int LDC)

- int [PLASMA_zher2k](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB, double beta, PLASMA_Complex64_t *C, int LDC)
- int [PLASMA_zherk](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, double alpha, PLASMA_Complex64_t *A, int LDA, double beta, PLASMA_Complex64_t *C, int LDC)
- double [PLASMA_zlange](#) (PLASMA_enum norm, int M, int N, PLASMA_Complex64_t *A, int LDA, double *work)
- double [PLASMA_zlanhe](#) (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_Complex64_t *A, int LDA, double *work)
- double [PLASMA_zlansy](#) (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_Complex64_t *A, int LDA, double *work)
- int [PLASMA_zlauum](#) (PLASMA_enum uplo, int N, PLASMA_Complex64_t *A, int LDA)
- int [PLASMA_zplghe](#) (double bump, int N, PLASMA_Complex64_t *A, int LDA, unsigned long long int seed)
- int [PLASMA_zplgsy](#) (PLASMA_Complex64_t bump, int N, PLASMA_Complex64_t *A, int LDA, unsigned long long int seed)
- int [PLASMA_zplrnt](#) (int M, int N, PLASMA_Complex64_t *A, int LDA, unsigned long long int seed)
- int [PLASMA_zposv](#) (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_zpotrf](#) (PLASMA_enum uplo, int N, PLASMA_Complex64_t *A, int LDA)
- int [PLASMA_zpotri](#) (PLASMA_enum uplo, int N, PLASMA_Complex64_t *A, int LDA)
- int [PLASMA_zpotrs](#) (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_zsymm](#) (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex64_t alpha, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t *C, int LDC)
- int [PLASMA_zsyr2k](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t *C, int LDC)
- int [PLASMA_zsyrrk](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t beta, PLASMA_Complex64_t *C, int LDC)
- int [PLASMA_ztrmm](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex64_t alpha, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_ztrsm](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex64_t alpha, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_ztrsmlq](#) (int N, int NRHS, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *L, int *IPIV, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_ztrtri](#) (PLASMA_enum uplo, PLASMA_enum diag, int N, PLASMA_Complex64_t *A, int LDA)
- int [PLASMA_zunglq](#) (int M, int N, int K, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *T, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_zungqr](#) (int M, int N, int K, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *T, PLASMA_Complex64_t *Q, int LDQ)
- int [PLASMA_zunmlq](#) (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *T, PLASMA_Complex64_t *B, int LDB)
- int [PLASMA_zunmqr](#) (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex64_t *A, int LDA, PLASMA_Complex64_t *T, PLASMA_Complex64_t *B, int LDB)

- int `PLASMA_zLapack_to_Tile` (`PLASMA_Complex64_t *Af77`, int `LDA`, `PLASMA_desc *A`)
- int `PLASMA_zTile_to_Lapack` (`PLASMA_desc *A`, `PLASMA_Complex64_t *Af77`, int `LDA`)

3.1.1 Detailed Description

This is the group of double complex functions using the simple user interface.

3.1.2 Function/Subroutine Documentation

3.1.2.1 int PLASMA_zcgels (PLASMA_enum *trans*, int *M*, int *N*, int *NRHS*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *B*, int *LDB*, PLASMA_Complex64_t * *X*, int *LDX*, int * *ITER*)

`PLASMA_zcgels` - Solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

`trans` = `PlasmaNoTrans` and $M \geq N$: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize $\| B - A * X \|$.

`trans` = `PlasmaNoTrans` and $M < N$: find the minimum norm solution of an underdetermined system $A * X = B$.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

`PLASMA_zcgels` first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX*16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX*16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX*16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if $ITER > ITERMAX$ or for all the RHS we have: $RNRM < N * XNRM * ANRM * EPS * BWDMAX$ where:

- `ITER` is the number of the current iteration in the iterative refinement process
- `RNRM` is the infinity-norm of the residual
- `XNRM` is the infinity-norm of the solution
- `ANRM` is the infinity-operator-norm of the matrix A
- `EPS` is the machine epsilon returned by `DLAMCH('Epsilon')`.

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

The values `ITERMAX` and `BWDMAX` are fixed to 30 and 1.0D+00 respectively.

We follow Bjorck's algorithm proposed in "Iterative Refinement of Linear Least Squares solutions I", BIT, 7:257-278, 1967.

Parameters

← *trans* Intended usage: = `PlasmaNoTrans`: the linear system involves A; = `PlasmaConjTrans`: the linear system involves $A ** H$. Currently only `PlasmaNoTrans` is supported.

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.
- ← *A* The M-by-N matrix A. This matrix is not modified.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- ← *B* The M-by-NRHS matrix B of right hand side vectors, stored columnwise. Not modified.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, M, N)$.
- *X* If return value = 0, the solution vectors, stored columnwise. if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;
- ← *LDX* The leading dimension of the array B. $LDB \geq \max(1, M, N)$.
- *ITER* The number of the current iteration in the iterative refinement process

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

- [PLASMA_zcgels_Tile](#)
- [PLASMA_zcgels_Tile_Async](#)
- [PLASMA_dsgels](#)
- [PLASMA_zgels](#)

3.1.2.2 int PLASMA_zcgesv (int N, int NRHS, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB, PLASMA_Complex64_t * X, int LDX, int * ITER)

PLASMA_zcgesv - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

PLASMA_zcgesv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX*16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX*16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX*16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if $ITER > ITERMAX$ or for all the RHS we have: $RNRM < N * XNRM * ANRM * EPS * BWDMAX$ where:

- *ITER* is the number of the current iteration in the iterative refinement process
- *RNRM* is the infinity-norm of the residual

- XNRM is the infinity-norm of the solution
- ANRM is the infinity-operator-norm of the matrix A
- EPS is the machine epsilon returned by DLAMCH('Epsilon').

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

The values ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

Parameters

- ← *N* The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← *A* The N-by-N coefficient matrix A. This matrix is not modified.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← *B* The N-by-NRHS matrix of right hand side matrix B.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.
- *X* If return value = 0, the N-by-NRHS solution matrix X.
- ← *LDX* The leading dimension of the array B. $LDX \geq \max(1, N)$.
- *ITER* The number of the current iteration in the iterative refinement process

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_zcgesv_Tile](#)
[PLASMA_zcgesv_Tile_Async](#)
[PLASMA_dsgesv](#)
[PLASMA_zgesv](#)

3.1.2.3 `int PLASMA_zcposv(PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB, PLASMA_Complex64_t * X, int LDX, int * ITER)`

`PLASMA_zcposv` - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$A = U^{**}H * U$, if `uplo = PlasmaUpper`, or $A = L * L^{**}H$, if `uplo = PlasmaLower`,

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations $A * X = B$.

PLASMA_zposv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX*16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX*16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX*16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if $ITER > ITERMAX$ or for all the RHS we have: $RNRM < N * XNRM * ANRM * EPS * BWDMAX$ where:

- $ITER$ is the number of the current iteration in the iterative refinement process
- $RNRM$ is the infinity-norm of the residual
- $XNRM$ is the infinity-norm of the solution
- $ANRM$ is the infinity-operator-norm of the matrix A
- EPS is the machine epsilon returned by DLAMCH('Epsilon').

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

The values $ITERMAX$ and $BWDMAX$ are fixed to 30 and 1.0D+00 respectively.

Parameters

- ← N The number of linear equations, i.e., the order of the matrix A . $N \geq 0$.
- ← $NRHS$ The number of right hand sides, i.e., the number of columns of the matrix B . $NRHS \geq 0$.
- ← A The N -by- N symmetric positive definite (or Hermitian) coefficient matrix A . If $uplo = PlasmaUpper$, the leading N -by- N upper triangular part of A contains the upper triangular part of the matrix A , and the strictly lower triangular part of A is not referenced. If $UPLO = 'L'$, the leading N -by- N lower triangular part of A contains the lower triangular part of the matrix A , and the strictly upper triangular part of A is not referenced. This matrix is not modified.
- ← LDA The leading dimension of the array A . $LDA \geq \max(1, N)$.
- ← B The N -by- $NRHS$ matrix of right hand side matrix B .
- ← LDB The leading dimension of the array B . $LDB \geq \max(1, N)$.
- X If return value = 0, the N -by- $NRHS$ solution matrix X .
- ← LDX The leading dimension of the array B . $LDX \geq \max(1, N)$.
- $ITER$ The number of the current iteration in the iterative refinement process

Returns

Return values

- $PLASMA_SUCCESS$ successful exit
- <0 if $-i$, the i -th argument had an illegal value
- >0 if i , the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_zcposv_Tile](#)
[PLASMA_zcposv_Tile_Async](#)
[PLASMA_dsposv](#)
[PLASMA_zposv](#)

3.1.2.4 `int PLASMA_zcungesv (PLASMA_enum trans, int N, int NRHS, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB, PLASMA_Complex64_t * X, int LDX, int * ITER)`

PLASMA_zcungesv - Solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and $M \geq N$: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize $\| B - A * X \|$.

trans = PlasmaNoTrans and $M < N$: find the minimum norm solution of an underdetermined system $A * X = B$.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

PLASMA_zcungesv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX*16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX*16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX*16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if $ITER > ITERMAX$ or for all the RHS we have: $RNRM < N * XNRM * ANRM * EPS * BWDMAX$ where:

- ITER is the number of the current iteration in the iterative refinement process
- RNRM is the infinity-norm of the residual
- XNRM is the infinity-norm of the solution
- ANRM is the infinity-operator-norm of the matrix A
- EPS is the machine epsilon returned by DLAMCH('Epsilon').

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

The values ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

We follow Bjorck's algorithm proposed in "Iterative Refinement of Linear Least Squares solutions I", BIT, 7:257-278, 1967.4

Parameters

← *trans* Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A^{**H} . Currently only PlasmaNoTrans is supported.

← *M* The number of rows of the matrix A. $M \geq 0$.

- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.
- ← *A* The M-by-N matrix A. This matrix is not modified.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- ← *B* The M-by-NRHS matrix B of right hand side vectors, stored columnwise. Not modified.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, M, N)$.
- *X* If return value = 0, the solution vectors, stored columnwise. if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;
- ← *LDX* The leading dimension of the array B. $LDB \geq \max(1, M, N)$.
- *ITER* The number of the current iteration in the iterative refinement process

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

- [PLASMA_zcungesv_Tile](#)
- [PLASMA_zcungesv_Tile_Async](#)
- [PLASMA_dsungesv](#)
- [PLASMA_zgels](#)

3.1.2.5 `int PLASMA_zgelqf (int M, int N, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * T)`

`PLASMA_zgelqf` - Computes the tile LQ factorization of a complex M-by-N matrix A: $A = L * Q$.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if $M \leq N$); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- *T* On exit, auxiliary factorization data, required by `PLASMA_zgelqs` to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zgelqf_Tile](#)
[PLASMA_zgelqf_Tile_Async](#)
[PLASMA_cgelqf](#)
[PLASMA_dgelqf](#)
[PLASMA_sgelqf](#)
[PLASMA_zgelqs](#)

3.1.2.6 int PLASMA_zgelqs (int *M*, int *N*, int *NRHS*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *T*, PLASMA_Complex64_t * *B*, int *LDB*)

PLASMA_zgelqs - Compute a minimum-norm solution $\min \|A * X - B\|$ using the LQ factorization $A = L * Q$ computed by PLASMA_zgelqf.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq M \geq 0$.
- ← *NRHS* The number of columns of B. $NRHS \geq 0$.
- ← *A* Details of the LQ factorization of the original matrix A as returned by PLASMA_zgelqf.
- ← *LDA* The leading dimension of the array A. $LDA \geq M$.
- ← *T* Auxiliary factorization data, computed by PLASMA_zgelqf.
- ← *B* On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq N$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zgelqs_Tile](#)
[PLASMA_zgelqs_Tile_Async](#)
[PLASMA_cgelqs](#)
[PLASMA_dgelqs](#)
[PLASMA_sgelqs](#)
[PLASMA_zgelqf](#)

3.1.2.7 `int PLASMA_zgels (PLASMA_enum trans, int M, int N, int NRHS, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * T, PLASMA_Complex64_t * B, int LDB)`

`PLASMA_zgels` - solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

`trans = PlasmaNoTrans` and $M \geq N$: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize $\| B - A * X \|$.

`trans = PlasmaNoTrans` and $M < N$: find the minimum norm solution of an underdetermined system $A * X = B$.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Parameters

- ← **trans** Intended usage: = `PlasmaNoTrans`: the linear system involves A; = `PlasmaConjTrans`: the linear system involves $A * H$. Currently only `PlasmaNoTrans` is supported.
- ← **M** The number of rows of the matrix A. $M \geq 0$.
- ← **N** The number of columns of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.
- ↔ **A** On entry, the M-by-N matrix A. On exit, if $M \geq N$, A is overwritten by details of its QR factorization as returned by `PLASMA_zgeqrf`; if $M < N$, A is overwritten by details of its LQ factorization as returned by `PLASMA_zgelqf`.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, M)$.
- **T** On exit, auxiliary factorization data.
- ↔ **B** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, M, N)$.

Returns

Return values

- `PLASMA_SUCCESS` successful exit
- <0 if -i, the i-th argument had an illegal value

See also

- [PLASMA_zgels_Tile](#)
- [PLASMA_zgels_Tile_Async](#)
- [PLASMA_cgels](#)
- [PLASMA_dgels](#)
- [PLASMA_sgels](#)

3.1.2.8 `int PLASMA_zgemm (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t * C, int LDC)`

PLASMA_zgemm - Performs one of the matrix-matrix operations

$$C = \alpha[op(A) \times op(B)] + \beta C$$

,

where `op(X)` is one of

`op(X) = X` or `op(X) = X'` or `op(X) = conjg(X')`

`alpha` and `beta` are scalars, and `A`, `B` and `C` are matrices, with `op(A)` an `m` by `k` matrix, `op(B)` a `k` by `n` matrix and `C` an `m` by `n` matrix.

Parameters

- ← **transA** Specifies whether the matrix `A` is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: `A` is not transposed; = PlasmaTrans: `A` is transposed; = PlasmaConjTrans: `A` is conjugate transposed.
- ← **transB** Specifies whether the matrix `B` is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: `B` is not transposed; = PlasmaTrans: `B` is transposed; = PlasmaConjTrans: `B` is conjugate transposed.
- ← **M** `M` specifies the number of rows of the matrix `op(A)` and of the matrix `C`. `M` \geq 0.
- ← **N** `N` specifies the number of columns of the matrix `op(B)` and of the matrix `C`. `N` \geq 0.
- ← **K** `K` specifies the number of columns of the matrix `op(A)` and the number of rows of the matrix `op(B)`. `K` \geq 0.
- ← **alpha** `alpha` specifies the scalar `alpha`
- ← **A** `A` is a LDA-by-ka matrix, where `ka` is `K` when `transA = PlasmaNoTrans`, and is `M` otherwise.
- ← **LDA** The leading dimension of the array `A`. `LDA` \geq `max(1,M)`.
- ← **B** `B` is a LDB-by-kb matrix, where `kb` is `N` when `transB = PlasmaNoTrans`, and is `K` otherwise.
- ← **LDB** The leading dimension of the array `B`. `LDB` \geq `max(1,N)`.
- ← **beta** `beta` specifies the scalar `beta`
- ← **C** `C` is a LDC-by-N matrix. On exit, the array is overwritten by the `M` by `N` matrix (`alpha*op(A)*op(B) + beta*C`)
- ← **LDC** The leading dimension of the array `C`. `LDC` \geq `max(1,M)`.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zgemm_Tile](#)
[PLASMA_cgemm](#)
[PLASMA_dgemm](#)
[PLASMA_sgemm](#)

3.1.2.9 `int PLASMA_zgeqrf(int M, int N, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * T)`

`PLASMA_zgeqrf` - Computes the tile QR factorization of a complex M-by-N matrix A: $A = Q * R$.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the $\min(M,N)$ -by- N upper trapezoidal matrix R (R is upper triangular if $M \geq N$); the elements below the diagonal represent the unitary matrix Q as a product of elementary reflectors stored by tiles.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,M)$.
- *T* On exit, auxiliary factorization data, required by `PLASMA_zgeqrs` to solve the system of equations.

Returns

Return values

- `PLASMA_SUCCESS` successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zgeqrf_Tile](#)
[PLASMA_zgeqrf_Tile_Async](#)
[PLASMA_cgeqrf](#)
[PLASMA_dgeqrf](#)
[PLASMA_sgeqrf](#)
[PLASMA_zgeqrs](#)

3.1.2.10 `int PLASMA_zgeqrs(int M, int N, int NRHS, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * T, PLASMA_Complex64_t * B, int LDB)`

`PLASMA_zgeqrs` - Compute a minimum-norm solution $\min \|A * X - B\|$ using the RQ factorization $A = R * Q$ computed by `PLASMA_zgeqrf`.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq M \geq 0$.
- ← *NRHS* The number of columns of B. $NRHS \geq 0$.
- ↔ *A* Details of the QR factorization of the original matrix A as returned by `PLASMA_zgeqrf`.
- ← *LDA* The leading dimension of the array A. $LDA \geq M$.
- ← *T* Auxiliary factorization data, computed by `PLASMA_zgeqrf`.
- ↔ *B* On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1,N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zgeqrs_Tile](#)
[PLASMA_zgeqrs_Tile_Async](#)
[PLASMA_cgeqrs](#)
[PLASMA_dgeqrs](#)
[PLASMA_sgeqrs](#)
[PLASMA_zgeqrf](#)

3.1.2.11 int PLASMA_zgesv (int N, int NRHS, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * L, int * IPIV, PLASMA_Complex64_t * B, int LDB)

PLASMA_zgesv - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The tile LU decomposition with partial tile pivoting and row interchanges is used to factor A. The factored form of A is then used to solve the system of equations $A * X = B$.

Parameters

- ← *N* The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ↔ *A* On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- *L* On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.
- *IPIV* On exit, the pivot indices that define the permutations (not equivalent to LAPACK).
- ↔ *B* On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value
 >0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_zgesv_Tile](#)
[PLASMA_zgesv_Tile_Async](#)
[PLASMA_cgesv](#)
[PLASMA_dgesv](#)
[PLASMA_sgesv](#)
[PLASMA_zcgesv](#)

3.1.2.12 int PLASMA_zgetrf (int *M*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *L*, int * *IPIV*)

PLASMA_zgetrf - Computes an LU factorization of a general M-by-N matrix A using the tile LU algorithm with partial tile pivoting with row interchanges.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- *L* On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_zgetrs to solve the system of equations.
- *IPIV* The pivot indices that define the permutations (not equivalent to LAPACK).

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

See also

[PLASMA_zgetrf_Tile](#)
[PLASMA_zgetrf_Tile_Async](#)
[PLASMA_cgetrf](#)
[PLASMA_dgetrf](#)
[PLASMA_sgetrf](#)
[PLASMA_zgetrs](#)

3.1.2.13 int PLASMA_zgetrs (PLASMA_enum *trans*, int *N*, int *NRHS*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *L*, int * *IPIV*, PLASMA_Complex64_t * *B*, int *LDB*)

PLASMA_zgetrs - Solves a system of linear equations $A * X = B$, with a general N-by-N matrix A using the tile LU factorization computed by PLASMA_zgetrf.

Parameters

- ← **trans** Intended to specify the the form of the system of equations: = PlasmaNoTrans: $A * X = B$ (No transpose) = PlasmaTrans: $A^{**T} * X = B$ (Transpose) = PlasmaConjTrans: $A^{**H} * X = B$ (Conjugate transpose) Currently only PlasmaNoTrans is supported.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The tile factors L and U from the factorization, computed by PLASMA_zgetrf.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1,N)$.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_zgetrf.
- ← **IPIV** The pivot indices from PLASMA_zgetrf (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1,N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit

Returns

<0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zgetrs_Tile](#)
[PLASMA_zgetrs_Tile_Async](#)
[PLASMA_cgetrs](#)
[PLASMA_dgetrs](#)
[PLASMA_sgetrs](#)
[PLASMA_zgetrf](#)

3.1.2.14 int PLASMA_zhemm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex64_t alpha, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t * C, int LDC)

PLASMA_zhemm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an hermitian matrix and B and C are m by n matrices.

Parameters

← *side* Specifies whether the hermitian matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← *uplo* Specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the hermitian matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the hermitian matrix A is to be referenced.

← *M* Specifies the number of rows of the matrix C. $M \geq 0$.

← *N* Specifies the number of columns of the matrix C. $N \geq 0$.

← *alpha* Specifies the scalar alpha.

← *A* A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← *LDA* The leading dimension of the array A. $LDA \geq \max(1, ka)$.

← *B* B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← *LDB* The leading dimension of the array B. $LDB \geq \max(1, M)$.

← *beta* Specifies the scalar beta.

↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← *LDC* The leading dimension of the array C. $LDC \geq \max(1, M)$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zhemm_Tile](#)

[PLASMA_chemm](#)

[PLASMA_dhemm](#)

[PLASMA_shemm](#)

3.1.2.15 `int PLASMA_zher2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB, double beta, PLASMA_Complex64_t * C, int LDC)`

PLASMA_zher2k - Performs one of the hermitian rank 2k operations

$$C = \alpha[op(A) \times conjg(op(B)')] + conjg(\alpha)[op(B) \times conjg(op(A)')] + \beta C$$

, or

$$C = \alpha[conjg(op(A)') \times op(B)] + conjg(\alpha)[conjg(op(B)') \times op(A)] + \beta C$$

,

where $op(X)$ is one of

$op(X) = X$ or $op(X) = conjg(X')$

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← **trans** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times conjg(op(B)')] + conjg(\alpha)[op(B) \times conjg(op(A)')] + \beta C$$

= PlasmaConjTrans:

$$C = \alpha[conjg(op(A)') \times op(B)] + conjg(\alpha)[conjg(op(B)') \times op(A)] + \beta C$$

← **N** N specifies the order of the matrix C. N must be at least zero.

← **K** K specifies the number of columns of the A and B matrices with trans = PlasmaNoTrans. K specifies the number of rows of the A and B matrices with trans = PlasmaTrans.

← **alpha** alpha specifies the scalar alpha.

← **A** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← **LDA** The leading dimension of the array A. LDA must be at least $\max(1, N)$, otherwise LDA must be at least $\max(1, K)$.

← **B** B is a LDB-by-kb matrix, where kb is K when trans = PlasmaNoTrans, and is N otherwise.

← **LDB** The leading dimension of the array B. LDB must be at least $\max(1, N)$, otherwise LDB must be at least $\max(1, K)$.

← **beta** beta specifies the scalar beta.

↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← **LDC** The leading dimension of the array C. $LDC \geq \max(1, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zher2k_Tile](#)

[PLASMA_cher2k](#)

[PLASMA_dher2k](#)

[PLASMA_sher2k](#)

3.1.2.16 int PLASMA_zherk (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, double *alpha*, PLASMA_Complex64_t * *A*, int *LDA*, double *beta*, PLASMA_Complex64_t * *C*, int *LDC*)

PLASMA_zherk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times conjg(op(A)')] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = conjg(X')

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← *N* N specifies the order of the matrix C. N must be at least zero.
- ← *K* K specifies the number of columns of the matrix op(A).
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← *LDA* The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.
- ← *LDC* The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zherk_Tile](#)
[PLASMA_cherk](#)
[PLASMA_dherk](#)
[PLASMA_sherk](#)

3.1.2.17 double PLASMA_zlange (PLASMA_enum *norm*, int *M*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*, double * *work*)

PLASMA_zlange returns the value

$zlange = (\max(\text{abs}(A(i,j))), \text{NORM} = \text{PlasmaMaxNorm} ((\text{norm1}(A), \text{NORM} = \text{PlasmaOneNorm} ((\text{normI}(A), \text{NORM} = \text{PlasmaInfNorm} ((\text{normF}(A), \text{NORM} = \text{PlasmaFrobeniusNorm}$

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that $\max(\text{abs}(A(i,j)))$ is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← *M* The number of rows of the matrix A. $M \geq 0$. When $M = 0$, the returned value is set to zero.
- ← *N* The number of columns of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The M-by-N matrix A.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,M)$.
- ← *work* double precision array of dimension $(\text{MAX}(1,\text{LWORK}))$, where $\text{LWORK} \geq M$ when $\text{NORM} = \text{PlasmaInfNorm}$; otherwise, *WORK* is not referenced.

Returns

Return values

the norm described above.

See also

[PLASMA_zlange_Tile](#)
[PLASMA_zlange_Tile_Async](#)
[PLASMA_clange](#)
[PLASMA_dlange](#)
[PLASMA_slange](#)

3.1.2.18 double PLASMA_zlanhe (PLASMA_enum *norm*, PLASMA_enum *uplo*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*, double * *work*)

PLASMA_zlanhe returns the value

$zlanhe = (\max(\text{abs}(A(i,j))), \text{NORM} = \text{PlasmaMaxNorm} ((\text{norm1}(A), \text{NORM} = \text{PlasmaOneNorm} ((\text{normI}(A), \text{NORM} = \text{PlasmaInfNorm} ((\text{normF}(A), \text{NORM} = \text{PlasmaFrobeniusNorm}$

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that $\max(\text{abs}(A(i,j)))$ is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The number of columns/rows of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The N-by-N matrix A.

- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← *work* double precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

[PLASMA_zlanhe_Tile](#)
[PLASMA_zlanhe_Tile_Async](#)
[PLASMA_clanhe](#)
[PLASMA_dlanhe](#)
[PLASMA_slanhe](#)

3.1.2.19 double PLASMA_zlansy (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_Complex64_t * A, int LDA, double * work)

PLASMA_zlansy returns the value

$zlansy = (\max(\text{abs}(A(i,j))), \text{NORM} = \text{PlasmaMaxNorm} ((\text{norm1}(A), \text{NORM} = \text{PlasmaOneNorm} ((\text{normI}(A), \text{NORM} = \text{PlasmaInfNorm} ((\text{normF}(A), \text{NORM} = \text{PlasmaFrobeniusNorm}$

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that $\max(\text{abs}(A(i,j)))$ is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The number of columns/rows of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The N-by-N matrix A.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← *work* double precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

[PLASMA_zlansy_Tile](#)

[PLASMA_zlansy_Tile_Async](#)
[PLASMA_clansy](#)
[PLASMA_dlansy](#)
[PLASMA_slansy](#)

3.1.2.20 `int PLASMA_zLapack_to_Tile (PLASMA_Complex64_t * Af77, int LDA, PLASMA_desc * A)`

PLASMA_zLapack_to_Tile - Conversion from LAPACK layout to tile layout.

Parameters

- ← *Af77* LAPACK matrix.
- ← *LDA* The leading dimension of the matrix *Af77*.
- ↔ *A* Descriptor of the PLASMA matrix in tile layout. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, *A->mat* is not used and set to *Af77* when returns, else if PLASMA_TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, *A->mat* has to be allocated before.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zLapack_to_Tile_Async](#)
[PLASMA_zTile_to_Lapack](#)
[PLASMA_cLapack_to_Tile](#)
[PLASMA_dLapack_to_Tile](#)
[PLASMA_sLapack_to_Tile](#)

3.1.2.21 `int PLASMA_zlauum (PLASMA_enum uplo, int N, PLASMA_Complex64_t * A, int LDA)`

PLASMA_zlauum - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A .

If $UPLO = 'U'$ or $'u'$ then the upper triangle of the result is stored, overwriting the factor U in A . If $UPLO = 'L'$ or $'l'$ then the lower triangle of the result is stored, overwriting the factor L in A .

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The order of the triangular factor U or L . $N \geq 0$.
- ↔ *A* On entry, the triangular factor U or L . On exit, if $UPLO = 'U'$, the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if $UPLO = 'L'$, the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.
- ← *LDA* The leading dimension of the array A . $LDA \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zlauum_Tile](#)
[PLASMA_zlauum_Tile_Async](#)
[PLASMA_clauum](#)
[PLASMA_dlauum](#)
[PLASMA_slauum](#)
[PLASMA_zpotri](#)

3.1.2.22 int PLASMA_zplghe (double *bump*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_zplghe - Generate a random hermitian matrix by tiles.

Parameters

← *bump* The value to add to the diagonal to be sure to have a positive definite matrix.
 ← *N* The order of the matrix A. $N \geq 0$.
 → *A* On exit, The random hermitian matrix A generated.
 ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zplghe_Tile](#)
[PLASMA_zplghe_Tile_Async](#)
[PLASMA_cplghe](#)
[PLASMA_dpplghe](#)
[PLASMA_splghe](#)
[PLASMA_zplrnt](#)
[PLASMA_zplgsy](#)

3.1.2.23 int PLASMA_zplgsy (PLASMA_Complex64_t *bump*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_zplgsy - Generate a random hermitian matrix by tiles.

Parameters

- ← *bump* The value to add to the diagonal to be sure to have a positive definite matrix.
- ← *N* The order of the matrix A. $N \geq 0$.
- *A* On exit, The random hermitian matrix A generated.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zplgsy_Tile](#)
[PLASMA_zplgsy_Tile_Async](#)
[PLASMA_cplgsy](#)
[PLASMA_dplgsy](#)
[PLASMA_splgsy](#)
[PLASMA_zplrnt](#)
[PLASMA_zplgsy](#)

3.1.2.24 int PLASMA_zplrnt (int M, int N, PLASMA_Complex64_t * A, int LDA, unsigned long long int seed)

PLASMA_zplrnt - Generate a random matrix by tiles.

Parameters

- ← *M* The number of rows of A.
- ← *N* The order of the matrix A. $N \geq 0$.
- *A* On exit, The random matrix A generated.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zplrnt_Tile](#)
[PLASMA_zplrnt_Tile_Async](#)
[PLASMA_cplrnt](#)
[PLASMA_dplrnt](#)
[PLASMA_splrnt](#)
[PLASMA_zplghe](#)
[PLASMA_zplgsy](#)

3.1.2.25 int PLASMA_zposv (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB)

PLASMA_zposv - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$$A = \begin{cases} U^H \times U, & \text{if uplo=PlasmaUpper} \\ L \times L^H, & \text{if uplo=PlasmaLower} \end{cases}$$

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations $A * X = B$.

Parameters

- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **N** The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ↔ **A** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U * H * U$ or $A = L * L * H$.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_zposv_Tile](#)
[PLASMA_zposv_Tile_Async](#)
[PLASMA_cposv](#)
[PLASMA_dposv](#)
[PLASMA_sposv](#)

3.1.2.26 int PLASMA_zpotrf (PLASMA_enum *uplo*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*)

PLASMA_zpotrf - Computes the Cholesky factorization of a symmetric positive definite (or Hermitian positive definite in the complex case) matrix *A*. The factorization has the form

$$A = \begin{cases} U^H \times U, & \text{if } uplo = PlasmaUpper \\ L \times L^H, & \text{if } uplo = PlasmaLower \end{cases}$$

where *U* is an upper triangular matrix and *L* is a lower triangular matrix.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.
- ← *N* The order of the matrix *A*. $N \geq 0$.
- ↔ *A* On entry, the symmetric positive definite (or Hermitian) matrix *A*. If *uplo* = PlasmaUpper, the leading *N*-by-*N* upper triangular part of *A* contains the upper triangular part of the matrix *A*, and the strictly lower triangular part of *A* is not referenced. If *UPLO* = 'L', the leading *N*-by-*N* lower triangular part of *A* contains the lower triangular part of the matrix *A*, and the strictly upper triangular part of *A* is not referenced. On exit, if return value = 0, the factor *U* or *L* from the Cholesky factorization $A = U^{**}H*U$ or $A = L*L^{**}H$.
- ← *LDA* The leading dimension of the array *A*. $LDA \geq \max(1,N)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- < 0 if -*i*, the *i*-th argument had an illegal value
- > 0 if *i*, the leading minor of order *i* of *A* is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_zpotrf_Tile](#)
[PLASMA_zpotrf_Tile_Async](#)
[PLASMA_cpotrf](#)
[PLASMA_dpotrf](#)
[PLASMA_spotrf](#)
[PLASMA_zpotrs](#)

3.1.2.27 int PLASMA_zpotri (PLASMA_enum *uplo*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*)

PLASMA_zpotri - Computes the inverse of a complex Hermitian positive definite matrix *A* using the Cholesky factorization $A = U^{**}H*U$ or $A = L*L^{**}H$ computed by PLASMA_zpotrf.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.
- ← *N* The order of the matrix *A*. $N \geq 0$.

- ↔ **A** On entry, the triangular factor U or L from the Cholesky factorization $A = U^{**}H^{*}U$ or $A = L^{*}L^{**}H$, as computed by `PLASMA_zpotrf`. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1,N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

See also

[PLASMA_zpotri_Tile](#)
[PLASMA_zpotri_Tile_Async](#)
[PLASMA_cpotri](#)
[PLASMA_dpotri](#)
[PLASMA_spotri](#)
[PLASMA_zpotrf](#)

3.1.2.28 int PLASMA_zpotrs (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB)

`PLASMA_zpotrs` - Solves a system of linear equations $A * X = B$ with a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A using the Cholesky factorization $A = U^{**}H^{*}U$ or $A = L^{*}L^{**}H$ computed by `PLASMA_zpotrf`.

Parameters

- ← **uplo** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The triangular factor U or L from the Cholesky factorization $A = U^{**}H^{*}U$ or $A = L^{*}L^{**}H$, computed by `PLASMA_zpotrf`.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1,N)$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1,N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zpotrs_Tile](#)
[PLASMA_zpotrs_Tile_Async](#)
[PLASMA_cpotrs](#)
[PLASMA_dpotrs](#)
[PLASMA_spotrs](#)
[PLASMA_zpotrf](#)

3.1.2.29 `int PLASMA_zsymm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex64_t alpha, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t * C, int LDC)`

PLASMA_zsymm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an symmetric matrix and B and C are m by n matrices.

Parameters

← **side** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← **uplo** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← **M** Specifies the number of rows of the matrix C. $M \geq 0$.

← **N** Specifies the number of columns of the matrix C. $N \geq 0$.

← **alpha** Specifies the scalar alpha.

← **A** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← **LDA** The leading dimension of the array A. $LDA \geq \max(1,ka)$.

← **B** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← **LDB** The leading dimension of the array B. $LDB \geq \max(1,M)$.

← **beta** Specifies the scalar beta.

↔ **C** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← **LDC** The leading dimension of the array C. $LDC \geq \max(1,M)$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zsymm_Tile](#)

[PLASMA_csymm](#)

[PLASMA_dsymm](#)

[PLASMA_ssymm](#)

3.1.2.30 `int PLASMA_zsyr2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t * C, int LDC)`

PLASMA_zsyr2k - Performs one of the symmetric rank 2k operations

$$C = \alpha[op(A) \times conjg(op(B)')] + \alpha[op(B) \times conjg(op(A)')] + \beta C$$

, or

$$C = \alpha[conjg(op(A)') \times op(B)] + \alpha[conjg(op(B)') \times op(A)] + \beta C$$

,

where `op(X)` is one of

`op(X) = X` or `op(X) = conjg(X')`

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← **trans** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times conjg(op(B)')] + \alpha[op(B) \times conjg(op(A)')] + \beta C$$

= PlasmaTrans:

$$C = \alpha[conjg(op(A)') \times op(B)] + \alpha[conjg(op(B)') \times op(A)] + \beta C$$

← **N** N specifies the order of the matrix C. N must be at least zero.

← **K** K specifies the number of columns of the A and B matrices with `trans = PlasmaNoTrans`. K specifies the number of rows of the A and B matrices with `trans = PlasmaTrans`.

← **alpha** alpha specifies the scalar alpha.

← **A** A is a LDA-by-ka matrix, where ka is K when `trans = PlasmaNoTrans`, and is N otherwise.

← **LDA** The leading dimension of the array A. LDA must be at least `max(1, N)`, otherwise LDA must be at least `max(1, K)`.

← **B** B is a LDB-by-kb matrix, where kb is K when `trans = PlasmaNoTrans`, and is N otherwise.

- ← **LDB** The leading dimension of the array B. LDB must be at least $\max(1, N)$, otherwise LDB must be at least $\max(1, K)$.
- ← **beta** beta specifies the scalar beta.
- ↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.
- ← **LDC** The leading dimension of the array C. $LDC \geq \max(1, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zsyr2k_Tile](#)
[PLASMA_csyr2k](#)
[PLASMA_dsyr2k](#)
[PLASMA_ssyr2k](#)

3.1.2.31 `int PLASMA_zsyrk(PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t beta, PLASMA_Complex64_t * C, int LDC)`

PLASMA_zsyrk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times conjg(op(A)')] + \beta C$$

,

where $op(X)$ is one of

$op(X) = X$ or $op(X) = conjg(X')$

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

- ← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← **trans** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans : A is transposed.
- ← **N** N specifies the order of the matrix C. N must be at least zero.
- ← **K** K specifies the number of columns of the matrix $op(A)$.
- ← **alpha** alpha specifies the scalar alpha.
- ← **A** A is a LDA-by-ka matrix, where ka is K when $trans = PlasmaNoTrans$, and is N otherwise.
- ← **LDA** The leading dimension of the array A. LDA must be at least $\max(1, N)$, otherwise LDA must be at least $\max(1, K)$.
- ← **beta** beta specifies the scalar beta
- ↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← *LDC* The leading dimension of the array C. $LDC \geq \max(1, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zsyrk_Tile](#)
[PLASMA_csyrk](#)
[PLASMA_dsyrk](#)
[PLASMA_ssyrk](#)

3.1.2.32 int PLASMA_zTile_to_Lapack (PLASMA_desc * A, PLASMA_Complex64_t * Af77, int LDA)

PLASMA_Tile_to_Lapack - Conversion from tile layout to LAPACK layout.

Parameters

← *A* Descriptor of the PLASMA matrix in tile layout.
 ↔ *Af77* LAPACK matrix. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, *Af77* has to be *A*->mat, else if PLASMA_TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, *Af77* has to be allocated before.
 ← *LDA* The leading dimension of the matrix *Af77*.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zTile_to_Lapack_Async](#)
[PLASMA_zLapack_to_Tile](#)
[PLASMA_cTile_to_Lapack](#)
[PLASMA_dTile_to_Lapack](#)
[PLASMA_sTile_to_Lapack](#)

3.1.2.33 int PLASMA_ztrmm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex64_t alpha, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB)

PLASMA_ztrmm - Computes $B = \alpha * \text{op}(A) * B$ or $B = \alpha * B * \text{op}(A)$.

Parameters

← *side* Specifies whether *A* appears on the left or on the right of *X*: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$

- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **transA** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← **diag** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The triangular matrix A. If **uplo** = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If **uplo** = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If **diag** = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_ztrmm_Tile](#)
[PLASMA_ztrmm_Tile_Async](#)
[PLASMA_ctrmm](#)
[PLASMA_dtrmm](#)
[PLASMA_strmm](#)

3.1.2.34 int PLASMA_ztrsm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex64_t alpha, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * B, int LDB)

PLASMA_ztrsm - Computes triangular solve $A * X = B$ or $X * A = B$.

Parameters

- ← **side** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **transA** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.

- ← *diag* Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← *N* The order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← *A* The triangular matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If *uplo* = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If *diag* = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_ztrsm_Tile](#)
[PLASMA_ztrsm_Tile_Async](#)
[PLASMA_ctrsm](#)
[PLASMA_dtrsm](#)
[PLASMA_strsm](#)

3.1.2.35 int PLASMA_ztrsmpl (int N, int NRHS, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * L, int * IPIV, PLASMA_Complex64_t * B, int LDB)

PLASMA_ztrsmpl - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix.

Parameters

- ← *N* The order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← *A* The tile factor L from the factorization, computed by PLASMA_zgetrf.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← *L* Auxiliary factorization data, related to the tile L factor, computed by PLASMA_zgetrf.
- ← *IPIV* The pivot indices from PLASMA_zgetrf (not equivalent to LAPACK).
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_ztrsmpi_Tile](#)
[PLASMA_ztrsmpi_Tile_Async](#)
[PLASMA_ctrsmpl](#)
[PLASMA_dtrsmpi](#)
[PLASMA_strsmpl](#)
[PLASMA_zgetrf](#)

3.1.2.36 int PLASMA_ztrtri (PLASMA_enum *uplo*, PLASMA_enum *diag*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*)

PLASMA_ztrtri - Computes the inverse of a complex upper or lower triangular matrix A.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
 ← *diag* = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.
 ← *N* The order of the matrix A. $N \geq 0$.
 ↔ *A* On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.
 ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value
 >0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

[PLASMA_ztrtri_Tile](#)
[PLASMA_ztrtri_Tile_Async](#)
[PLASMA_ctrtri](#)
[PLASMA_dtrtri](#)
[PLASMA_strtri](#)
[PLASMA_zpotri](#)

3.1.2.37 `int PLASMA_zunglq(int M, int N, int K, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * T, PLASMA_Complex64_t * B, int LDB)`

`PLASMA_zunglq` - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by `PLASMA_zgelqf`.

Parameters

- ← *M* The number of rows of the matrix Q. $M \geq 0$.
- ← *N* The number of columns of the matrix Q. $N \geq M$.
- ← *K* The number of rows of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the LQ factorization of the original matrix A as returned by `PLASMA_zgelqf`.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- ← *T* Auxiliary factorization data, computed by `PLASMA_zgelqf`.
- *B* On exit, the M-by-N matrix Q.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, M)$.

Returns

Return values

- `PLASMA_SUCCESS` successful exit
- `PLASMA_SUCCESS` <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zunglq_Tile](#)
[PLASMA_zunglq_Tile_Async](#)
[PLASMA_cunglq](#)
[PLASMA_dunglq](#)
[PLASMA_sunglq](#)
[PLASMA_zgelqf](#)

3.1.2.38 `int PLASMA_zungqr(int M, int N, int K, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * T, PLASMA_Complex64_t * Q, int LDQ)`

`PLASMA_zungqr` - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by `PLASMA_zgeqrf`.

Parameters

- ← *M* The number of rows of the matrix Q. $M \geq 0$.
- ← *N* The number of columns of the matrix Q. $N \geq M$.
- ← *K* The number of columns of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the QR factorization of the original matrix A as returned by `PLASMA_zgeqrf`.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

- ← *T* Auxiliary factorization data, computed by PLASMA_zgeqrf.
- *Q* On exit, the M-by-N matrix Q.
- ← *LDQ* The leading dimension of the array Q. LDQ $\geq \max(1,M)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zungqr_Tile](#)
[PLASMA_zungqr_Tile_Async](#)
[PLASMA_cungqr](#)
[PLASMA_dungqr](#)
[PLASMA_sungqr](#)
[PLASMA_zgeqrf](#)

3.1.2.39 int PLASMA_zunmlq (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *T*, PLASMA_Complex64_t * *B*, int *LDB*)

PLASMA_zunmlq - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_zgelqf. Q is of order M.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q**H from the left; = PlasmaRight: apply Q or Q**H from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjugate transpose, apply Q**H. Currently only PlasmaConjTrans is supported.
- ← *M* The number of rows of the matrix C. $M \geq 0$.
- ← *N* The number of columns of the matrix C. $N \geq 0$.
- ← *K* The number of rows of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the LQ factorization of the original matrix A as returned by PLASMA_zgelqf.
- ← *LDA* The leading dimension of the array A. LDA $\geq \max(1,K)$.
- ← *T* Auxiliary factorization data, computed by PLASMA_zgelqf.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by Q*B or Q**H*B.
- ← *LDB* The leading dimension of the array C. LDB $\geq \max(1,M)$.

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zunmlq_Tile](#)
[PLASMA_zunmlq_Tile_Async](#)
[PLASMA_cunmlq](#)
[PLASMA_dunmlq](#)
[PLASMA_sunmlq](#)
[PLASMA_zgelqf](#)

3.1.2.40 int PLASMA_zunmqr (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex64_t * A, int LDA, PLASMA_Complex64_t * T, PLASMA_Complex64_t * B, int LDB)

PLASMA_zunmqr - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_zgeqrf. Q is of order M.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q**H from the left; = PlasmaRight: apply Q or Q**H from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjugate transpose, apply Q**H. Currently only PlasmaConjTrans is supported.
- ← *M* The number of rows of the matrix C. $M \geq 0$.
- ← *N* The number of columns of the matrix C. $N \geq 0$.
- ← *K* The number of columns of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_zgeqrf.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$;
- ← *T* Auxiliary factorization data, computed by PLASMA_zgeqrf.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by Q*B or Q**H*B.
- ← *LDB* The leading dimension of the array C. $LDB \geq \max(1, M)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_zunmqr_Tile](#)
[PLASMA_zunmqr_Tile_Async](#)
[PLASMA_cunmqr](#)
[PLASMA_dunmqr](#)
[PLASMA_sunmqr](#)
[PLASMA_zgeqrf](#)

3.2 Simple Interface - Single Complex

Functions/Subroutines

- int [PLASMA_cgelqf](#) (int M, int N, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T)
- int [PLASMA_cgelqs](#) (int M, int N, int NRHS, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_cgels](#) (PLASMA_enum trans, int M, int N, int NRHS, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_cgemm](#) (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t *C, int LDC)
- int [PLASMA_cgeqrf](#) (int M, int N, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T)
- int [PLASMA_cgeqrs](#) (int M, int N, int NRHS, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_cgesv](#) (int N, int NRHS, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *L, int *IPIV, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_cgetrf](#) (int M, int N, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *L, int *IPIV)
- int [PLASMA_cgetrs](#) (PLASMA_enum trans, int N, int NRHS, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *L, int *IPIV, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_chemm](#) (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t *C, int LDC)
- int [PLASMA_cher2k](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB, float beta, PLASMA_Complex32_t *C, int LDC)
- int [PLASMA_cherk](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, float alpha, PLASMA_Complex32_t *A, int LDA, float beta, PLASMA_Complex32_t *C, int LDC)
- float [PLASMA_clange](#) (PLASMA_enum norm, int M, int N, PLASMA_Complex32_t *A, int LDA, float *work)
- float [PLASMA_clanhe](#) (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_Complex32_t *A, int LDA, float *work)
- float [PLASMA_clansy](#) (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_Complex32_t *A, int LDA, float *work)
- int [PLASMA_clauum](#) (PLASMA_enum uplo, int N, PLASMA_Complex32_t *A, int LDA)
- int [PLASMA_cplghe](#) (float bump, int N, PLASMA_Complex32_t *A, int LDA, unsigned long long int seed)
- int [PLASMA_cplgsy](#) (PLASMA_Complex32_t bump, int N, PLASMA_Complex32_t *A, int LDA, unsigned long long int seed)
- int [PLASMA_cplrnt](#) (int M, int N, PLASMA_Complex32_t *A, int LDA, unsigned long long int seed)
- int [PLASMA_cposv](#) (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_cpotrf](#) (PLASMA_enum uplo, int N, PLASMA_Complex32_t *A, int LDA)
- int [PLASMA_cpotri](#) (PLASMA_enum uplo, int N, PLASMA_Complex32_t *A, int LDA)
- int [PLASMA_cpotrs](#) (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB)

- int [PLASMA_csymm](#) (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t *C, int LDC)
- int [PLASMA_csyr2k](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t *C, int LDC)
- int [PLASMA_csyrrk](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t beta, PLASMA_Complex32_t *C, int LDC)
- int [PLASMA_ctrmm](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_ctrsm](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_ctrsmpl](#) (int N, int NRHS, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *L, int *IPIV, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_ctrtri](#) (PLASMA_enum uplo, PLASMA_enum diag, int N, PLASMA_Complex32_t *A, int LDA)
- int [PLASMA_cunglq](#) (int M, int N, int K, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_cungqr](#) (int M, int N, int K, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *Q, int LDQ)
- int [PLASMA_cunmlq](#) (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_cunmqr](#) (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *B, int LDB)
- int [PLASMA_cLapack_to_Tile](#) (PLASMA_Complex32_t *Af77, int LDA, PLASMA_desc *A)
- int [PLASMA_cTile_to_Lapack](#) (PLASMA_desc *A, PLASMA_Complex32_t *Af77, int LDA)

3.2.1 Detailed Description

This is the group of single complex functions using the simple user interface.

3.2.2 Function/Subroutine Documentation

3.2.2.1 int [PLASMA_cgelqf](#) (int M, int N, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * T)

[PLASMA_cgelqf](#) - Computes the tile LQ factorization of a complex M-by-N matrix A: $A = L * Q$.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if $M \leq N$); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, M)$.
- **T** On exit, auxiliary factorization data, required by PLASMA_cgelqs to solve the system of equations.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cgelqf_Tile](#)
[PLASMA_cgelqf_Tile_Async](#)
[PLASMA_cgelqf](#)
[PLASMA_dgelqf](#)
[PLASMA_sgelqf](#)
[PLASMA_cgelqs](#)

3.2.2.2 int PLASMA_cgelqs (int M, int N, int NRHS, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * T, PLASMA_Complex32_t * B, int LDB)

PLASMA_cgelqs - Compute a minimum-norm solution $\min \|A * X - B\|$ using the LQ factorization $A = L * Q$ computed by PLASMA_cgelqf.

Parameters

- ← **M** The number of rows of the matrix A. $M \geq 0$.
- ← **N** The number of columns of the matrix A. $N \geq M \geq 0$.
- ← **NRHS** The number of columns of B. $NRHS \geq 0$.
- ← **A** Details of the LQ factorization of the original matrix A as returned by PLASMA_cgelqf.
- ← **LDA** The leading dimension of the array A. $LDA \geq M$.
- ← **T** Auxiliary factorization data, computed by PLASMA_cgelqf.
- ↔ **B** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq N$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cgelqs_Tile](#)

[PLASMA_cgelqs_Tile_Async](#)
[PLASMA_cgelqs](#)
[PLASMA_dgelqs](#)
[PLASMA_sgelqs](#)
[PLASMA_cgelqf](#)

3.2.2.3 int PLASMA_cgels (PLASMA_enum trans, int M, int N, int NRHS, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * T, PLASMA_Complex32_t * B, int LDB)

PLASMA_cgels - solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and $M \geq N$: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize $\|B - A * X\|$.

trans = PlasmaNoTrans and $M < N$: find the minimum norm solution of an underdetermined system $A * X = B$.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Parameters

- ← **trans** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves $A \setminus * \setminus H$. Currently only PlasmaNoTrans is supported.
- ← **M** The number of rows of the matrix A. $M \geq 0$.
- ← **N** The number of columns of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.
- ↔ **A** On entry, the M-by-N matrix A. On exit, if $M \geq N$, A is overwritten by details of its QR factorization as returned by PLASMA_cgeqrf; if $M < N$, A is overwritten by details of its LQ factorization as returned by PLASMA_cgelqf.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, M)$.
- **T** On exit, auxiliary factorization data.
- ↔ **B** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, M, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cgels_Tile](#)
[PLASMA_cgels_Tile_Async](#)
[PLASMA_cgels](#)
[PLASMA_dgels](#)
[PLASMA_sgels](#)

3.2.2.4 `int PLASMA_cgemm (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t * C, int LDC)`

PLASMA_cgemm - Performs one of the matrix-matrix operations

$$C = \alpha[op(A) \times op(B)] + \beta C$$

,

where `op(X)` is one of

`op(X) = X` or `op(X) = X'` or `op(X) = conjfg(X')`

`alpha` and `beta` are scalars, and `A`, `B` and `C` are matrices, with `op(A)` an `m` by `k` matrix, `op(B)` a `k` by `n` matrix and `C` an `m` by `n` matrix.

Parameters

- ← **transA** Specifies whether the matrix `A` is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: `A` is not transposed; = PlasmaTrans: `A` is transposed; = PlasmaConjTrans: `A` is conjugate transposed.
- ← **transB** Specifies whether the matrix `B` is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: `B` is not transposed; = PlasmaTrans: `B` is transposed; = PlasmaConjTrans: `B` is conjugate transposed.
- ← **M** `M` specifies the number of rows of the matrix `op(A)` and of the matrix `C`. `M` \geq 0.
- ← **N** `N` specifies the number of columns of the matrix `op(B)` and of the matrix `C`. `N` \geq 0.
- ← **K** `K` specifies the number of columns of the matrix `op(A)` and the number of rows of the matrix `op(B)`. `K` \geq 0.
- ← **alpha** `alpha` specifies the scalar `alpha`
- ← **A** `A` is a LDA-by-ka matrix, where ka is `K` when `transA = PlasmaNoTrans`, and is `M` otherwise.
- ← **LDA** The leading dimension of the array `A`. LDA \geq `max(1,M)`.
- ← **B** `B` is a LDB-by-kb matrix, where kb is `N` when `transB = PlasmaNoTrans`, and is `K` otherwise.
- ← **LDB** The leading dimension of the array `B`. LDB \geq `max(1,N)`.
- ← **beta** `beta` specifies the scalar `beta`
- ← **C** `C` is a LDC-by-N matrix. On exit, the array is overwritten by the `M` by `N` matrix (`alpha*op(A)*op(B) + beta*C`)
- ← **LDC** The leading dimension of the array `C`. LDC \geq `max(1,M)`.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cgemm_Tile](#)
[PLASMA_cgemm](#)
[PLASMA_dgemm](#)
[PLASMA_sgemm](#)

3.2.2.5 int PLASMA_cgeqrf (int *M*, int *N*, PLASMA_Complex32_t * *A*, int *LDA*, PLASMA_Complex32_t * *T*)

PLASMA_cgeqrf - Computes the tile QR factorization of a complex M-by-N matrix A: $A = Q * R$.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix R (R is upper triangular if $M \geq N$); the elements below the diagonal represent the unitary matrix Q as a product of elementary reflectors stored by tiles.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- *T* On exit, auxiliary factorization data, required by PLASMA_cgeqrs to solve the system of equations.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cgeqrf_Tile](#)
[PLASMA_cgeqrf_Tile_Async](#)
[PLASMA_cgeqrf](#)
[PLASMA_dgeqrf](#)
[PLASMA_sgeqrf](#)
[PLASMA_cgeqrs](#)

3.2.2.6 int PLASMA_cgeqrs (int *M*, int *N*, int *NRHS*, PLASMA_Complex32_t * *A*, int *LDA*, PLASMA_Complex32_t * *T*, PLASMA_Complex32_t * *B*, int *LDB*)

PLASMA_cgeqrs - Compute a minimum-norm solution $\min \|A * X - B\|$ using the RQ factorization $A = R * Q$ computed by PLASMA_cgeqrf.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq M \geq 0$.
- ← *NRHS* The number of columns of B. $NRHS \geq 0$.
- ↔ *A* Details of the QR factorization of the original matrix A as returned by PLASMA_cgeqrf.
- ← *LDA* The leading dimension of the array A. $LDA \geq M$.
- ← *T* Auxiliary factorization data, computed by PLASMA_cgeqrf.
- ↔ *B* On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1,N)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cgeqrs_Tile](#)
[PLASMA_cgeqrs_Tile_Async](#)
[PLASMA_cgeqrs](#)
[PLASMA_dgeqrs](#)
[PLASMA_sgeqrs](#)
[PLASMA_cgeqrf](#)

3.2.2.7 int PLASMA_cgesv (int N, int NRHS, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * L, int * IPIV, PLASMA_Complex32_t * B, int LDB)

PLASMA_cgesv - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The tile LU decomposition with partial tile pivoting and row interchanges is used to factor A. The factored form of A is then used to solve the system of equations $A * X = B$.

Parameters

- ← *N* The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ↔ *A* On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,N)$.
- *L* On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.
- *IPIV* On exit, the pivot indices that define the permutations (not equivalent to LAPACK).
- ↔ *B* On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1,N)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_cgesv_Tile](#)
[PLASMA_cgesv_Tile_Async](#)
[PLASMA_cgesv](#)
[PLASMA_dgesv](#)
[PLASMA_sgesv](#)
[PLASMA_ccgesv](#)

3.2.2.8 `int PLASMA_cgetrf (int M, int N, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * L, int * IPIV)`

PLASMA_cgetrf - Computes an LU factorization of a general M-by-N matrix A using the tile LU algorithm with partial tile pivoting with row interchanges.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- *L* On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_cgetrs to solve the system of equations.
- *IPIV* The pivot indices that define the permutations (not equivalent to LAPACK).

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

See also

[PLASMA_cgetrf_Tile](#)
[PLASMA_cgetrf_Tile_Async](#)
[PLASMA_cgetrf](#)
[PLASMA_dgetrf](#)
[PLASMA_sgetrf](#)
[PLASMA_cgetrs](#)

3.2.2.9 `int PLASMA_cgetrs (PLASMA_enum trans, int N, int NRHS, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * L, int * IPIV, PLASMA_Complex32_t * B, int LDB)`

PLASMA_cgetrs - Solves a system of linear equations $A * X = B$, with a general N-by-N matrix A using the tile LU factorization computed by PLASMA_cgetrf.

Parameters

- ← **trans** Intended to specify the the form of the system of equations: = PlasmaNoTrans: $A * X = B$ (No transpose) = PlasmaTrans: $A^{**T} * X = B$ (Transpose) = PlasmaConjTrans: $A \backslash * \backslash * H * X = B$ (Conjugate transpose) Currently only PlasmaNoTrans is supported.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The tile factors L and U from the factorization, computed by PLASMA_cgetrf.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_cgetrf.
- ← **IPIV** The pivot indices from PLASMA_cgetrf (not equivalent to LAPACK).
- ← **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

`PLASMA_SUCCESS` successful exit

Returns

<0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cgetrs_Tile](#)
[PLASMA_cgetrs_Tile_Async](#)
[PLASMA_cgetrs](#)
[PLASMA_dgetrs](#)
[PLASMA_sgetrs](#)
[PLASMA_cgetrf](#)

3.2.2.10 `int PLASMA_chemm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex32_t alpha, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t * C, int LDC)`

PLASMA_chemm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an hermitian matrix and B and C are m by n matrices.

Parameters

← *side* Specifies whether the hermitian matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← *uplo* Specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the hermitian matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the hermitian matrix A is to be referenced.

← *M* Specifies the number of rows of the matrix C. $M \geq 0$.

← *N* Specifies the number of columns of the matrix C. $N \geq 0$.

← *alpha* Specifies the scalar alpha.

← *A* A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← *LDA* The leading dimension of the array A. $LDA \geq \max(1,ka)$.

← *B* B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← *LDB* The leading dimension of the array B. $LDB \geq \max(1,M)$.

← *beta* Specifies the scalar beta.

↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← *LDC* The leading dimension of the array C. $LDC \geq \max(1,M)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_chemm_Tile](#)

[PLASMA_chemm](#)

[PLASMA_dhemm](#)

[PLASMA_shemm](#)

3.2.2.11 `int PLASMA_cher2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * B, int LDB, float beta, PLASMA_Complex32_t * C, int LDC)`

PLASMA_cher2k - Performs one of the hermitian rank 2k operations

$$C = \alpha[op(A) \times conjfg(op(B)')] + conjfg(\alpha)[op(B) \times conjfg(op(A)')] + \beta C$$

, or

$$C = \alpha[conjfg(op(A)') \times op(B)] + conjfg(\alpha)[conjfg(op(B)') \times op(A)] + \beta C$$

where `op(X)` is one of

`op(X) = X` or `op(X) = conjfg(X')`

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
 ← **trans** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times conjfg(op(B)')] + conjfg(\alpha)[op(B) \times conjfg(op(A)')] + \beta C$$

= PlasmaConjTrans:

$$C = \alpha[conjfg(op(A)') \times op(B)] + conjfg(\alpha)[conjfg(op(B)') \times op(A)] + \beta C$$

← **N** N specifies the order of the matrix C. N must be at least zero.

← **K** K specifies the number of columns of the A and B matrices with `trans = PlasmaNoTrans`. K specifies the number of rows of the A and B matrices with `trans = PlasmaTrans`.

← **alpha** alpha specifies the scalar alpha.

← **A** A is a LDA-by-ka matrix, where ka is K when `trans = PlasmaNoTrans`, and is N otherwise.

← **LDA** The leading dimension of the array A. LDA must be at least `max(1, N)`, otherwise LDA must be at least `max(1, K)`.

← **B** B is a LDB-by-kb matrix, where kb is K when `trans = PlasmaNoTrans`, and is N otherwise.

← **LDB** The leading dimension of the array B. LDB must be at least `max(1, N)`, otherwise LDB must be at least `max(1, K)`.

← **beta** beta specifies the scalar beta.

↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← **LDC** The leading dimension of the array C. `LDC >= max(1, N)`.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cher2k_Tile](#)

[PLASMA_cher2k](#)

[PLASMA_dher2k](#)

[PLASMA_sher2k](#)

3.2.2.12 `int PLASMA_cherk(PLASMA_enum uplo, PLASMA_enum trans, int N, int K, float alpha, PLASMA_Complex32_t * A, int LDA, float beta, PLASMA_Complex32_t * C, int LDC)`

PLASMA_cherk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times conjfg(op(A)')] + \beta C$$

,

where `op(X)` is one of

`op(X) = X` or `op(X) = conjfg(X')`

where `alpha` and `beta` are real scalars, `C` is an `n`-by-`n` hermitian matrix and `A` is an `n`-by-`k` matrix in the first case and a `k`-by-`n` matrix in the second case.

Parameters

- ← ***uplo*** = PlasmaUpper: Upper triangle of `C` is stored; = PlasmaLower: Lower triangle of `C` is stored.
- ← ***trans*** Specifies whether the matrix `A` is transposed or conjugate transposed: = PlasmaNoTrans: `A` is not transposed; = PlasmaConjTrans: `A` is conjugate transposed.
- ← ***N*** `N` specifies the order of the matrix `C`. `N` must be at least zero.
- ← ***K*** `K` specifies the number of columns of the matrix `op(A)`.
- ← ***alpha*** `alpha` specifies the scalar `alpha`.
- ← ***A*** `A` is a `LDA`-by-`ka` matrix, where `ka` is `K` when `trans = PlasmaNoTrans`, and is `N` otherwise.
- ← ***LDA*** The leading dimension of the array `A`. `LDA` must be at least `max(1, N)`, otherwise `LDA` must be at least `max(1, K)`.
- ← ***beta*** `beta` specifies the scalar `beta`
- ↔ ***C*** `C` is a `LDC`-by-`N` matrix. On exit, the array `uplo` part of the matrix is overwritten by the `uplo` part of the updated matrix.
- ← ***LDC*** The leading dimension of the array `C`. `LDC` \geq `max(1, N)`.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_cherk_Tile](#)
[PLASMA_cherk](#)
[PLASMA_dherk](#)
[PLASMA_sherk](#)

3.2.2.13 `float PLASMA_clange(PLASMA_enum norm, int M, int N, PLASMA_Complex32_t * A, int LDA, float * work)`

PLASMA_clange returns the value

$\text{clange} = (\max(\text{abs}(A(i,j))), \text{NORM} = \text{PlasmaMaxNorm} ((\text{norm1}(A), \text{NORM} = \text{PlasmaOneNorm} ((\text{normI}(A), \text{NORM} = \text{PlasmaInfNorm} ((\text{normF}(A), \text{NORM} = \text{PlasmaFrobeniusNorm}$

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that $\max(\text{abs}(A(i,j)))$ is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← *M* The number of rows of the matrix A. $M \geq 0$. When $M = 0$, the returned value is set to zero.
- ← *N* The number of columns of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The M-by-N matrix A.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,M)$.
- ← *work* float precision array of dimension $(\text{MAX}(1,\text{LWORK}))$, where $\text{LWORK} \geq M$ when $\text{NORM} = \text{PlasmaInfNorm}$; otherwise, *WORK* is not referenced.

Returns

Return values

the norm described above.

See also

[PLASMA_clange_Tile](#)
[PLASMA_clange_Tile_Async](#)
[PLASMA_clange](#)
[PLASMA_dlange](#)
[PLASMA_slange](#)

3.2.2.14 float PLASMA_clanhe (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_Complex32_t * A, int LDA, float * work)

PLASMA_clanhe returns the value

$\text{clanhe} = (\max(\text{abs}(A(i,j))), \text{NORM} = \text{PlasmaMaxNorm} ((\text{norm1}(A), \text{NORM} = \text{PlasmaOneNorm} ((\text{normI}(A), \text{NORM} = \text{PlasmaInfNorm} ((\text{normF}(A), \text{NORM} = \text{PlasmaFrobeniusNorm}$

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that $\max(\text{abs}(A(i,j)))$ is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The number of columns/rows of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The N-by-N matrix A.

- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← *work* float precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

[PLASMA_clanhe_Tile](#)
[PLASMA_clanhe_Tile_Async](#)
[PLASMA_clanhe](#)
[PLASMA_dlanhe](#)
[PLASMA_slanhe](#)

3.2.2.15 float PLASMA_clansy (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_Complex32_t * A, int LDA, float * work)

PLASMA_clansy returns the value

$clansy = (\max(\text{abs}(A(i,j))), \text{NORM} = \text{PlasmaMaxNorm} ((\text{norm1}(A), \text{NORM} = \text{PlasmaOneNorm} ((\text{normI}(A), \text{NORM} = \text{PlasmaInfNorm} ((\text{normF}(A), \text{NORM} = \text{PlasmaFrobeniusNorm}$

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that $\max(\text{abs}(A(i,j)))$ is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The number of columns/rows of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The N-by-N matrix A.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← *work* float precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

[PLASMA_clansy_Tile](#)

[PLASMA_clansy_Tile_Async](#)
[PLASMA_clansy](#)
[PLASMA_dlansy](#)
[PLASMA_slansy](#)

3.2.2.16 int PLASMA_cLapack_to_Tile (PLASMA_Complex32_t * Af77, int LDA, PLASMA_desc * A)

PLASMA_cLapack_to_Tile - Conversion from LAPACK layout to tile layout.

Parameters

- ← *Af77* LAPACK matrix.
- ← *LDA* The leading dimension of the matrix *Af77*.
- ↔ *A* Descriptor of the PLASMA matrix in tile layout. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, *A->mat* is not used and set to *Af77* when returns, else if PLASMA_TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, *A->mat* has to be allocated before.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cLapack_to_Tile_Async](#)
[PLASMA_cTile_to_Lapack](#)
[PLASMA_cLapack_to_Tile](#)
[PLASMA_dLapack_to_Tile](#)
[PLASMA_sLapack_to_Tile](#)

3.2.2.17 int PLASMA_clauum (PLASMA_enum uplo, int N, PLASMA_Complex32_t * A, int LDA)

PLASMA_clauum - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A .

If $UPLO = 'U'$ or $'u'$ then the upper triangle of the result is stored, overwriting the factor U in A . If $UPLO = 'L'$ or $'l'$ then the lower triangle of the result is stored, overwriting the factor L in A .

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The order of the triangular factor U or L . $N \geq 0$.
- ↔ *A* On entry, the triangular factor U or L . On exit, if $UPLO = 'U'$, the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if $UPLO = 'L'$, the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.
- ← *LDA* The leading dimension of the array A . $LDA \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_clauum_Tile](#)
[PLASMA_clauum_Tile_Async](#)
[PLASMA_clauum](#)
[PLASMA_dlauum](#)
[PLASMA_slauum](#)
[PLASMA_cpotri](#)

3.2.2.18 int PLASMA_cplghe (float *bump*, int *N*, PLASMA_Complex32_t * *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_cplghe - Generate a random hermitian matrix by tiles.

Parameters

← *bump* The value to add to the diagonal to be sure to have a positive definite matrix.
 ← *N* The order of the matrix A. $N \geq 0$.
 → *A* On exit, The random hermitian matrix A generated.
 ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cplghe_Tile](#)
[PLASMA_cplghe_Tile_Async](#)
[PLASMA_cplghe](#)
[PLASMA_dplghe](#)
[PLASMA_splghe](#)
[PLASMA_cplrnt](#)
[PLASMA_cplgsy](#)

3.2.2.19 int PLASMA_cplgsy (PLASMA_Complex32_t *bump*, int *N*, PLASMA_Complex32_t * *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_cplgsy - Generate a random hermitian matrix by tiles.

Parameters

- ← *bump* The value to add to the diagonal to be sure to have a positive definite matrix.
- ← *N* The order of the matrix A. $N \geq 0$.
- *A* On exit, The random hermitian matrix A generated.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cplgsy_Tile](#)
[PLASMA_cplgsy_Tile_Async](#)
[PLASMA_cplgsy](#)
[PLASMA_dplgsy](#)
[PLASMA_splgsy](#)
[PLASMA_cplrnt](#)
[PLASMA_cplgsy](#)

3.2.2.20 int PLASMA_cplrnt (int M, int N, PLASMA_Complex32_t * A, int LDA, unsigned long long int seed)

PLASMA_cplrnt - Generate a random matrix by tiles.

Parameters

- ← *M* The number of rows of A.
- ← *N* The order of the matrix A. $N \geq 0$.
- *A* On exit, The random matrix A generated.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cplrnt_Tile](#)
[PLASMA_cplrnt_Tile_Async](#)
[PLASMA_cplrnt](#)
[PLASMA_dplrnt](#)
[PLASMA_splrnt](#)
[PLASMA_cplghe](#)
[PLASMA_cplgsy](#)

3.2.2.21 int PLASMA_cposv (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * B, int LDB)

PLASMA_cposv - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$$A = \begin{cases} U^H \times U, & \text{if uplo=PlasmaUpper} \\ L \times L^H, & \text{if uplo=PlasmaLower} \end{cases}$$

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations $A * X = B$.

Parameters

- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **N** The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ↔ **A** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U \backslash * H * U$ or $A = L * L \backslash * H$.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_cposv_Tile](#)
[PLASMA_cposv_Tile_Async](#)
[PLASMA_cposv](#)
[PLASMA_dposv](#)
[PLASMA_sposv](#)

3.2.2.22 int PLASMA_cpotrf (PLASMA_enum *uplo*, int *N*, PLASMA_Complex32_t * *A*, int *LDA*)

PLASMA_cpotrf - Computes the Cholesky factorization of a symmetric positive definite (or Hermitian positive definite in the complex case) matrix *A*. The factorization has the form

$$A = \begin{cases} U^H \times U, & \text{if } uplo = PlasmaUpper \\ L \times L^H, & \text{if } uplo = PlasmaLower \end{cases}$$

where *U* is an upper triangular matrix and *L* is a lower triangular matrix.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.
- ← *N* The order of the matrix *A*. $N \geq 0$.
- ↔ *A* On entry, the symmetric positive definite (or Hermitian) matrix *A*. If *uplo* = PlasmaUpper, the leading *N*-by-*N* upper triangular part of *A* contains the upper triangular part of the matrix *A*, and the strictly lower triangular part of *A* is not referenced. If *UPLO* = 'L', the leading *N*-by-*N* lower triangular part of *A* contains the lower triangular part of the matrix *A*, and the strictly upper triangular part of *A* is not referenced. On exit, if return value = 0, the factor *U* or *L* from the Cholesky factorization $A = U \backslash * \backslash * H * U$ or $A = L * L \backslash * \backslash * H$.
- ← *LDA* The leading dimension of the array *A*. $LDA \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- < 0 if -*i*, the *i*-th argument had an illegal value
- > 0 if *i*, the leading minor of order *i* of *A* is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_cpotrf_Tile](#)
[PLASMA_cpotrf_Tile_Async](#)
[PLASMA_cpotrf](#)
[PLASMA_dpotrf](#)
[PLASMA_spotrf](#)
[PLASMA_cpots](#)

3.2.2.23 int PLASMA_cpotri (PLASMA_enum *uplo*, int *N*, PLASMA_Complex32_t * *A*, int *LDA*)

PLASMA_cpotri - Computes the inverse of a complex Hermitian positive definite matrix *A* using the Cholesky factorization $A = U \backslash * \backslash * H * U$ or $A = L * L \backslash * \backslash * H$ computed by PLASMA_cpotrf.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.
- ← *N* The order of the matrix *A*. $N \geq 0$.

- ↔ **A** On entry, the triangular factor U or L from the Cholesky factorization $A = U \backslash * \backslash * H * U$ or $A = L * L \backslash * \backslash * H$, as computed by `PLASMA_cpotrf`. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

See also

[PLASMA_cpotri_Tile](#)
[PLASMA_cpotri_Tile_Async](#)
[PLASMA_cpotri](#)
[PLASMA_dpotri](#)
[PLASMA_spotri](#)
[PLASMA_cpotrf](#)

3.2.2.24 int PLASMA_cpotrs (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * B, int LDB)

`PLASMA_cpotrs` - Solves a system of linear equations $A * X = B$ with a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A using the Cholesky factorization $A = U \backslash * \backslash * H * U$ or $A = L * L \backslash * \backslash * H$ computed by `PLASMA_cpotrf`.

Parameters

- ← **uplo** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The triangular factor U or L from the Cholesky factorization $A = U \backslash * \backslash * H * U$ or $A = L * L \backslash * \backslash * H$, computed by `PLASMA_cpotrf`.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cpotrs_Tile](#)
[PLASMA_cpotrs_Tile_Async](#)
[PLASMA_cpotrs](#)
[PLASMA_dpotrs](#)
[PLASMA_spotrs](#)
[PLASMA_cpotrf](#)

3.2.2.25 `int PLASMA_csymm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex32_t alpha, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t * C, int LDC)`

PLASMA_csymm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an symmetric matrix and B and C are m by n matrices.

Parameters

← **side** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← **uplo** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← **M** Specifies the number of rows of the matrix C. $M \geq 0$.

← **N** Specifies the number of columns of the matrix C. $N \geq 0$.

← **alpha** Specifies the scalar alpha.

← **A** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← **LDA** The leading dimension of the array A. $LDA \geq \max(1,ka)$.

← **B** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← **LDB** The leading dimension of the array B. $LDB \geq \max(1,M)$.

← **beta** Specifies the scalar beta.

↔ **C** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← **LDC** The leading dimension of the array C. $LDC \geq \max(1,M)$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_csymm_Tile](#)

[PLASMA_csymm](#)

[PLASMA_dsymm](#)

[PLASMA_ssymm](#)

3.2.2.26 `int PLASMA_csyr2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t * C, int LDC)`

PLASMA_csyr2k - Performs one of the symmetric rank 2k operations

$$C = \alpha[op(A) \times conjfg(op(B)')] + \alpha[op(B) \times conjfg(op(A)')] + \beta C$$

, or

$$C = \alpha[conjfg(op(A)') \times op(B)] + \alpha[conjfg(op(B)') \times op(A)] + \beta C$$

,

where `op(X)` is one of

`op(X) = X` or `op(X) = conjfg(X')`

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← **trans** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times conjfg(op(B)')] + \alpha[op(B) \times conjfg(op(A)')] + \beta C$$

= PlasmaTrans:

$$C = \alpha[conjfg(op(A)') \times op(B)] + \alpha[conjfg(op(B)') \times op(A)] + \beta C$$

← **N** N specifies the order of the matrix C. N must be at least zero.

← **K** K specifies the number of columns of the A and B matrices with `trans = PlasmaNoTrans`. K specifies the number of rows of the A and B matrices with `trans = PlasmaTrans`.

← **alpha** alpha specifies the scalar alpha.

← **A** A is a LDA-by-ka matrix, where ka is K when `trans = PlasmaNoTrans`, and is N otherwise.

← **LDA** The leading dimension of the array A. LDA must be at least `max(1, N)`, otherwise LDA must be at least `max(1, K)`.

← **B** B is a LDB-by-kb matrix, where kb is K when `trans = PlasmaNoTrans`, and is N otherwise.

- ← **LDB** The leading dimension of the array B. LDB must be at least $\max(1, N)$, otherwise LDB must be at least $\max(1, K)$.
- ← **beta** beta specifies the scalar beta.
- ↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.
- ← **LDC** The leading dimension of the array C. $LDC \geq \max(1, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_csyr2k_Tile](#)
[PLASMA_csyr2k](#)
[PLASMA_dsyr2k](#)
[PLASMA_ssyr2k](#)

3.2.2.27 `int PLASMA_csyrk (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t beta, PLASMA_Complex32_t * C, int LDC)`

PLASMA_csyrk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times conjfg(op(A)')] + \beta C$$

,

where $op(X)$ is one of

$op(X) = X$ or $op(X) = conjfg(X')$

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

- ← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← **trans** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans : A is transposed.
- ← **N** N specifies the order of the matrix C. N must be at least zero.
- ← **K** K specifies the number of columns of the matrix $op(A)$.
- ← **alpha** alpha specifies the scalar alpha.
- ← **A** A is a LDA-by-ka matrix, where ka is K when $trans = PlasmaNoTrans$, and is N otherwise.
- ← **LDA** The leading dimension of the array A. LDA must be at least $\max(1, N)$, otherwise LDA must be at least $\max(1, K)$.
- ← **beta** beta specifies the scalar beta
- ↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← *LDC* The leading dimension of the array C. $LDC \geq \max(1, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_csyrc_Tile](#)
[PLASMA_csyrc](#)
[PLASMA_dsyrc](#)
[PLASMA_ssyrc](#)

3.2.2.28 int PLASMA_cTile_to_Lapack (PLASMA_desc * A, PLASMA_Complex32_t * Af77, int LDA)

PLASMA_Tile_to_Lapack - Conversion from tile layout to LAPACK layout.

Parameters

← *A* Descriptor of the PLASMA matrix in tile layout.
 ↔ *Af77* LAPACK matrix. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, *Af77* has to be A->mat, else if PLASMA_TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, *Af77* has to be allocated before.
 ← *LDA* The leading dimension of the matrix *Af77*.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cTile_to_Lapack_Async](#)
[PLASMA_cLapack_to_Tile](#)
[PLASMA_cTile_to_Lapack](#)
[PLASMA_dTile_to_Lapack](#)
[PLASMA_sTile_to_Lapack](#)

3.2.2.29 int PLASMA_ctrmm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex32_t alpha, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * B, int LDB)

PLASMA_ctrmm - Computes $B = \alpha * \text{op}(A) * B$ or $B = \alpha * B * \text{op}(A)$.

Parameters

← *side* Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B =$
 PlasmaRight: $X * A = B$

- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **transA** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← **diag** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The triangular matrix A. If **uplo** = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If **uplo** = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If **diag** = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_ctrmm_Tile](#)
[PLASMA_ctrmm_Tile_Async](#)
[PLASMA_ctrmm](#)
[PLASMA_dtrmm](#)
[PLASMA_strmm](#)

3.2.2.30 int PLASMA_ctrsm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex32_t alpha, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * B, int LDB)

PLASMA_ctrsm - Computes triangular solve $A * X = B$ or $X * A = B$.

Parameters

- ← **side** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **transA** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.

- ← **diag** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The triangular matrix A. If `uplo = PlasmaUpper`, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If `uplo = PlasmaLower`, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If `diag = PlasmaUnit`, the diagonal elements of A are also not referenced and are assumed to be 1.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_ctrsm_Tile](#)
[PLASMA_ctrsm_Tile_Async](#)
[PLASMA_ctrsm](#)
[PLASMA_dtrsm](#)
[PLASMA_strsm](#)

3.2.2.31 int PLASMA_ctrsmpl (int N, int NRHS, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * L, int * IPIV, PLASMA_Complex32_t * B, int LDB)

PLASMA_ctrsmpl - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix.

Parameters

- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The tile factor L from the factorization, computed by PLASMA_cgetrf.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_cgetrf.
- ← **IPIV** The pivot indices from PLASMA_cgetrf (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_ctrsmpl_Tile](#)
[PLASMA_ctrsmpl_Tile_Async](#)
[PLASMA_ctrsmpl](#)
[PLASMA_dtrsmpl](#)
[PLASMA_strsmpl](#)
[PLASMA_cgetrf](#)

3.2.2.32 int PLASMA_ctrtri (PLASMA_enum *uplo*, PLASMA_enum *diag*, int *N*, PLASMA_Complex32_t * *A*, int *LDA*)

PLASMA_ctrtri - Computes the inverse of a complex upper or lower triangular matrix A.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
 ← *diag* = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.
 ← *N* The order of the matrix A. $N \geq 0$.
 ↔ *A* On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.
 ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value
 >0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

[PLASMA_ctrtri_Tile](#)
[PLASMA_ctrtri_Tile_Async](#)
[PLASMA_ctrtri](#)
[PLASMA_dtrtri](#)
[PLASMA_strtri](#)
[PLASMA_cpotri](#)

3.2.2.33 `int PLASMA_cunglq (int M, int N, int K, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * T, PLASMA_Complex32_t * B, int LDB)`

`PLASMA_cunglq` - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by `PLASMA_cgelqf`.

Parameters

- ← *M* The number of rows of the matrix Q. $M \geq 0$.
- ← *N* The number of columns of the matrix Q. $N \geq M$.
- ← *K* The number of rows of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the LQ factorization of the original matrix A as returned by `PLASMA_cgelqf`.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- ← *T* Auxiliary factorization data, computed by `PLASMA_cgelqf`.
- *B* On exit, the M-by-N matrix Q.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, M)$.

Returns

Return values

- `PLASMA_SUCCESS` successful exit
- `PLASMA_SUCCESS` <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cunglq_Tile](#)
[PLASMA_cunglq_Tile_Async](#)
[PLASMA_cunglq](#)
[PLASMA_dunglq](#)
[PLASMA_sunglq](#)
[PLASMA_cgelqf](#)

3.2.2.34 `int PLASMA_cungqr (int M, int N, int K, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * T, PLASMA_Complex32_t * Q, int LDQ)`

`PLASMA_cungqr` - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by `PLASMA_cgeqrf`.

Parameters

- ← *M* The number of rows of the matrix Q. $M \geq 0$.
- ← *N* The number of columns of the matrix Q. $N \geq M$.
- ← *K* The number of columns of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the QR factorization of the original matrix A as returned by `PLASMA_cgeqrf`.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

- ← *T* Auxiliary factorization data, computed by PLASMA_cgeqrf.
- *Q* On exit, the M-by-N matrix Q.
- ← *LDQ* The leading dimension of the array Q. LDQ >= max(1,M).

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cungqr_Tile](#)
[PLASMA_cungqr_Tile_Async](#)
[PLASMA_cungqr](#)
[PLASMA_dungqr](#)
[PLASMA_sungqr](#)
[PLASMA_cgeqrf](#)

3.2.2.35 int PLASMA_cunmlq (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, PLASMA_Complex32_t * *A*, int *LDA*, PLASMA_Complex32_t * *T*, PLASMA_Complex32_t * *B*, int *LDB*)

PLASMA_cunmlq - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_cgelsqf. Q is of order M.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q**H from the left; = PlasmaRight: apply Q or Q**H from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjugate transpose, apply Q**H. Currently only PlasmaConjTrans is supported.
- ← *M* The number of rows of the matrix C. M >= 0.
- ← *N* The number of columns of the matrix C. N >= 0.
- ← *K* The number of rows of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.
- ← *A* Details of the LQ factorization of the original matrix A as returned by PLASMA_cgelsqf.
- ← *LDA* The leading dimension of the array A. LDA >= max(1,K).
- ← *T* Auxiliary factorization data, computed by PLASMA_cgelsqf.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by Q*B or Q**H*B.
- ← *LDB* The leading dimension of the array C. LDB >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cunmlq_Tile](#)
[PLASMA_cunmlq_Tile_Async](#)
[PLASMA_cunmlq](#)
[PLASMA_dunmlq](#)
[PLASMA_sunmlq](#)
[PLASMA_cgelqf](#)

3.2.2.36 `int PLASMA_cunmqr (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex32_t * A, int LDA, PLASMA_Complex32_t * T, PLASMA_Complex32_t * B, int LDB)`

PLASMA_cunmqr - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_cgeqrf. Q is of order M.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q^*H from the left; = PlasmaRight: apply Q or Q^*H from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjugate transpose, apply Q^*H . Currently only PlasmaConjTrans is supported.
- ← *M* The number of rows of the matrix C. $M \geq 0$.
- ← *N* The number of columns of the matrix C. $N \geq 0$.
- ← *K* The number of columns of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_cgeqrf.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$;
- ← *T* Auxiliary factorization data, computed by PLASMA_cgeqrf.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by Q^*B or Q^*H^*B .
- ← *LDB* The leading dimension of the array C. $LDB \geq \max(1, M)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_cunmqr_Tile](#)
[PLASMA_cunmqr_Tile_Async](#)
[PLASMA_cunmqr](#)
[PLASMA_dunmqr](#)
[PLASMA_sunmqr](#)
[PLASMA_cgeqrf](#)

3.3 Simple Interface - Double Real

Functions/Subroutines

- int [PLASMA_dgelqf](#) (int M, int N, double *A, int LDA, double *T)
- int [PLASMA_dgelqs](#) (int M, int N, int NRHS, double *A, int LDA, double *T, double *B, int LDB)
- int [PLASMA_dgels](#) (PLASMA_enum trans, int M, int N, int NRHS, double *A, int LDA, double *T, double *B, int LDB)
- int [PLASMA_dgemm](#) (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, double alpha, double *A, int LDA, double *B, int LDB, double beta, double *C, int LDC)
- int [PLASMA_dgeqrf](#) (int M, int N, double *A, int LDA, double *T)
- int [PLASMA_dgeqrs](#) (int M, int N, int NRHS, double *A, int LDA, double *T, double *B, int LDB)
- int [PLASMA_dgesv](#) (int N, int NRHS, double *A, int LDA, double *L, int *IPIV, double *B, int LDB)
- int [PLASMA_dgetrf](#) (int M, int N, double *A, int LDA, double *L, int *IPIV)
- int [PLASMA_dgetrs](#) (PLASMA_enum trans, int N, int NRHS, double *A, int LDA, double *L, int *IPIV, double *B, int LDB)
- double [PLASMA_dlange](#) (PLASMA_enum norm, int M, int N, double *A, int LDA, double *work)
- double [PLASMA_dlansy](#) (PLASMA_enum norm, PLASMA_enum uplo, int N, double *A, int LDA, double *work)
- int [PLASMA_dlauum](#) (PLASMA_enum uplo, int N, double *A, int LDA)
- int [PLASMA_dorglq](#) (int M, int N, int K, double *A, int LDA, double *T, double *B, int LDB)
- int [PLASMA_dorgqr](#) (int M, int N, int K, double *A, int LDA, double *T, double *Q, int LDQ)
- int [PLASMA_dormlq](#) (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, double *A, int LDA, double *T, double *B, int LDB)
- int [PLASMA_dormqr](#) (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, double *A, int LDA, double *T, double *B, int LDB)
- int [PLASMA_dpigsy](#) (double bump, int N, double *A, int LDA, unsigned long long int seed)
- int [PLASMA_dplrnt](#) (int M, int N, double *A, int LDA, unsigned long long int seed)
- int [PLASMA_dposv](#) (PLASMA_enum uplo, int N, int NRHS, double *A, int LDA, double *B, int LDB)
- int [PLASMA_dpotr](#) (PLASMA_enum uplo, int N, double *A, int LDA)
- int [PLASMA_dpotri](#) (PLASMA_enum uplo, int N, double *A, int LDA)
- int [PLASMA_dpotrs](#) (PLASMA_enum uplo, int N, int NRHS, double *A, int LDA, double *B, int LDB)
- int [PLASMA_dsgevs](#) (int N, int NRHS, double *A, int LDA, double *B, int LDB, double *X, int LDX, int *ITER)
- int [PLASMA_dsposv](#) (PLASMA_enum uplo, int N, int NRHS, double *A, int LDA, double *B, int LDB, double *X, int LDX, int *ITER)
- int [PLASMA_dsungesv](#) (PLASMA_enum trans, int N, int NRHS, double *A, int LDA, double *B, int LDB, double *X, int LDX, int *ITER)
- int [PLASMA_dsymm](#) (PLASMA_enum side, PLASMA_enum uplo, int M, int N, double alpha, double *A, int LDA, double *B, int LDB, double beta, double *C, int LDC)
- int [PLASMA_dsyr2k](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, double alpha, double *A, int LDA, double *B, int LDB, double beta, double *C, int LDC)
- int [PLASMA_dsyrrk](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, double alpha, double *A, int LDA, double beta, double *C, int LDC)
- int [PLASMA_dtrmm](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, double alpha, double *A, int LDA, double *B, int LDB)
- int [PLASMA_dtrsm](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, double alpha, double *A, int LDA, double *B, int LDB)

- int [PLASMA_dtrsmpl](#) (int *N*, int *NRHS*, double **A*, int *LDA*, double **L*, int **IPIV*, double **B*, int *LDB*)
- int [PLASMA_dtrtri](#) (PLASMA_enum *uplo*, PLASMA_enum *diag*, int *N*, double **A*, int *LDA*)
- int [PLASMA_dLapack_to_Tile](#) (double **Af77*, int *LDA*, PLASMA_desc **A*)
- int [PLASMA_dTile_to_Lapack](#) (PLASMA_desc **A*, double **Af77*, int *LDA*)

3.3.1 Detailed Description

This is the group of double real functions using the simple user interface.

3.3.2 Function/Subroutine Documentation

3.3.2.1 int PLASMA_dgelqf (int *M*, int *N*, double **A*, int *LDA*, double **T*)

PLASMA_dgelqf - Computes the tile LQ factorization of a complex M-by-N matrix A: $A = L * Q$.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if $M \leq N$); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- *T* On exit, auxiliary factorization data, required by PLASMA_dgelqs to solve the system of equations.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dgelqf_Tile](#)
[PLASMA_dgelqf_Tile_Async](#)
[PLASMA_cgelqf](#)
[PLASMA_dgelqf](#)
[PLASMA_sgelqf](#)
[PLASMA_dgelqs](#)

3.3.2.2 int PLASMA_dgelqs (int *M*, int *N*, int *NRHS*, double **A*, int *LDA*, double **T*, double **B*, int *LDB*)

PLASMA_dgelqs - Compute a minimum-norm solution $\min \|A * X - B\|$ using the LQ factorization $A = L * Q$ computed by PLASMA_dgelqf.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq M \geq 0$.
- ← *NRHS* The number of columns of B. $NRHS \geq 0$.
- ← *A* Details of the LQ factorization of the original matrix A as returned by PLASMA_dgelqf.
- ← *LDA* The leading dimension of the array A. $LDA \geq M$.
- ← *T* Auxiliary factorization data, computed by PLASMA_dgelqf.
- ↔ *B* On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq N$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dgelqs_Tile](#)
[PLASMA_dgelqs_Tile_Async](#)
[PLASMA_cgelqs](#)
[PLASMA_dgelqs](#)
[PLASMA_sgelqs](#)
[PLASMA_dgelqf](#)

3.3.2.3 int PLASMA_dgels (PLASMA_enum *trans*, int *M*, int *N*, int *NRHS*, double * *A*, int *LDA*, double * *T*, double * *B*, int *LDB*)

PLASMA_dgels - solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and $M \geq N$: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize $\| B - A * X \|$.

trans = PlasmaNoTrans and $M < N$: find the minimum norm solution of an underdetermined system $A * X = B$.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Parameters

- ← *trans* Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves A^T . Currently only PlasmaNoTrans is supported.
- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.

- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS ≥ 0 .
- ↔ **A** On entry, the M-by-N matrix A. On exit, if $M \geq N$, A is overwritten by details of its QR factorization as returned by PLASMA_dgeqrf; if $M < N$, A is overwritten by details of its LQ factorization as returned by PLASMA_dgelqf.
- ← **LDA** The leading dimension of the array A. LDA $\geq \max(1, M)$.
- **T** On exit, auxiliary factorization data.
- ↔ **B** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;
- ← **LDB** The leading dimension of the array B. LDB $\geq \max(1, M, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0** if -i, the i-th argument had an illegal value

See also

- [PLASMA_dgels_Tile](#)
- [PLASMA_dgels_Tile_Async](#)
- [PLASMA_cgels](#)
- [PLASMA_dgels](#)
- [PLASMA_sgels](#)

3.3.2.4 int PLASMA_dgemm (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, double alpha, double *A, int LDA, double *B, int LDB, double beta, double *C, int LDC)

PLASMA_dgemm - Performs one of the matrix-matrix operations

$$C = \alpha[op(A) \times op(B)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = X' or op(X) = g(X')

alpha and beta are scalars, and A, B and C are matrices, with op(A) an m by k matrix, op(B) a k by n matrix and C an m by n matrix.

Parameters

- ← **transA** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaTrans: A is ugate transposed.

- ← **transB** Specifies whether the matrix B is transposed, not transposed or ugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaTrans: B is ugate transposed.
- ← **M** M specifies the number of rows of the matrix op(A) and of the matrix C. $M \geq 0$.
- ← **N** N specifies the number of columns of the matrix op(B) and of the matrix C. $N \geq 0$.
- ← **K** K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). $K \geq 0$.
- ← **alpha** alpha specifies the scalar alpha
- ← **A** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1,M)$.
- ← **B** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1,N)$.
- ← **beta** beta specifies the scalar beta
- ↔ **C** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix ($\alpha * \text{op}(A) * \text{op}(B) + \beta * C$)
- ← **LDC** The leading dimension of the array C. $LDC \geq \max(1,M)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_dgemm_Tile](#)
[PLASMA_cgemm](#)
[PLASMA_dgemm](#)
[PLASMA_sgemm](#)

3.3.2.5 int PLASMA_dgeqrf (int M, int N, double * A, int LDA, double * T)

PLASMA_dgeqrf - Computes the tile QR factorization of a complex M-by-N matrix A: $A = Q * R$.

Parameters

- ← **M** The number of rows of the matrix A. $M \geq 0$.
- ← **N** The number of columns of the matrix A. $N \geq 0$.
- ↔ **A** On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the $\min(M,N)$ -by-N upper trapezoidal matrix R (R is upper triangular if $M \geq N$); the elements below the diagonal represent the unitary matrix Q as a product of elementary reflectors stored by tiles.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1,M)$.
- **T** On exit, auxiliary factorization data, required by PLASMA_dgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dgeqrf_Tile](#)
[PLASMA_dgeqrf_Tile_Async](#)
[PLASMA_cgeqrf](#)
[PLASMA_dgeqrf](#)
[PLASMA_sgeqrf](#)
[PLASMA_dgeqrs](#)

3.3.2.6 int PLASMA_dgeqrs (int M, int N, int NRHS, double * A, int LDA, double * T, double * B, int LDB)

PLASMA_dgeqrs - Compute a minimum-norm solution $\min \|A \cdot X - B\|$ using the RQ factorization $A = R \cdot Q$ computed by PLASMA_dgeqrf.

Parameters

← *M* The number of rows of the matrix A. $M \geq 0$.
 ← *N* The number of columns of the matrix A. $N \geq M \geq 0$.
 ← *NRHS* The number of columns of B. $NRHS \geq 0$.
 ↔ *A* Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.
 ← *LDA* The leading dimension of the array A. $LDA \geq M$.
 ← *T* Auxiliary factorization data, computed by PLASMA_dgeqrf.
 ↔ *B* On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.
 ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dgeqrs_Tile](#)
[PLASMA_dgeqrs_Tile_Async](#)
[PLASMA_cgeqrs](#)
[PLASMA_dgeqrs](#)
[PLASMA_sgeqrs](#)
[PLASMA_dgeqrf](#)

3.3.2.7 `int PLASMA_dgesv (int N, int NRHS, double * A, int LDA, double * L, int * IPIV, double * B, int LDB)`

`PLASMA_dgesv` - Computes the solution to a system of linear equations $A * X = B$, where A is an N -by- N matrix and X and B are N -by- $NRHS$ matrices. The tile LU decomposition with partial tile pivoting and row interchanges is used to factor A . The factored form of A is then used to solve the system of equations $A * X = B$.

Parameters

- ← N The number of linear equations, i.e., the order of the matrix A . $N \geq 0$.
- ← $NRHS$ The number of right hand sides, i.e., the number of columns of the matrix B . $NRHS \geq 0$.
- ↔ A On entry, the N -by- N coefficient matrix A . On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).
- ← LDA The leading dimension of the array A . $LDA \geq \max(1, N)$.
- L On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.
- $IPIV$ On exit, the pivot indices that define the permutations (not equivalent to LAPACK).
- ↔ B On entry, the N -by- $NRHS$ matrix of right hand side matrix B . On exit, if return value = 0, the N -by- $NRHS$ solution matrix X .
- ← LDB The leading dimension of the array B . $LDB \geq \max(1, N)$.

Returns

Return values

- `PLASMA_SUCCESS` successful exit
- < 0 if $-i$, the i -th argument had an illegal value
- > 0 if i , $U(i, i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_dgesv_Tile](#)
[PLASMA_dgesv_Tile_Async](#)
[PLASMA_cgesv](#)
[PLASMA_dgesv](#)
[PLASMA_sgesv](#)
[PLASMA_dcgsv](#)

3.3.2.8 `int PLASMA_dgetrf (int M, int N, double * A, int LDA, double * L, int * IPIV)`

`PLASMA_dgetrf` - Computes an LU factorization of a general M -by- N matrix A using the tile LU algorithm with partial tile pivoting with row interchanges.

Parameters

- ← M The number of rows of the matrix A . $M \geq 0$.
- ← N The number of columns of the matrix A . $N \geq 0$.

- ↔ **A** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, M)$.
- **L** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_dgetrs to solve the system of equations.
- **IPIV** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

See also

[PLASMA_dgetrf_Tile](#)
[PLASMA_dgetrf_Tile_Async](#)
[PLASMA_cgetrf](#)
[PLASMA_dgetrf](#)
[PLASMA_sgetrf](#)
[PLASMA_dgetrs](#)

3.3.2.9 int PLASMA_dgetrs (PLASMA_enum trans, int N, int NRHS, double * A, int LDA, double * L, int * IPIV, double * B, int LDB)

PLASMA_dgetrs - Solves a system of linear equations $A * X = B$, with a general N-by-N matrix A using the tile LU factorization computed by PLASMA_dgetrf.

Parameters

- ← **trans** Intended to specify the the form of the system of equations: = PlasmaNoTrans: $A * X = B$ (No transpose) = PlasmaTrans: $A * T * X = B$ (Transpose) = PlasmaTrans: $A \backslash * \backslash * T * X = B$ (Conjugate transpose) Currently only PlasmaNoTrans is supported.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The tile factors L and U from the factorization, computed by PLASMA_dgetrf.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_dgetrf.
- ← **IPIV** The pivot indices from PLASMA_dgetrf (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

Returns

<0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dgetrs_Tile](#)
[PLASMA_dgetrs_Tile_Async](#)
[PLASMA_cgetrs](#)
[PLASMA_dgetrs](#)
[PLASMA_sgetrs](#)
[PLASMA_dgetrf](#)

3.3.2.10 double PLASMA_dlange (PLASMA_enum *norm*, int *M*, int *N*, double * *A*, int *LDA*, double * *work*)

PLASMA_dlange returns the value

$dlange = (\max(\text{abs}(A(i,j))), \text{NORM} = \text{PlasmaMaxNorm} ((\text{norm1}(A), \text{NORM} = \text{PlasmaOneNorm} ((\text{normI}(A), \text{NORM} = \text{PlasmaInfNorm} ((\text{normF}(A), \text{NORM} = \text{PlasmaFrobeniusNorm}$

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that $\max(\text{abs}(A(i,j)))$ is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← *M* The number of rows of the matrix A. $M \geq 0$. When $M = 0$, the returned value is set to zero.
- ← *N* The number of columns of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The M-by-N matrix A.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,M)$.
- ← *work* double precision array of dimension $(\text{MAX}(1,\text{LWORK}))$, where $\text{LWORK} \geq M$ when $\text{NORM} = \text{PlasmaInfNorm}$; otherwise, WORK is not referenced.

Returns**Return values**

the norm described above.

See also

[PLASMA_dlange_Tile](#)
[PLASMA_dlange_Tile_Async](#)
[PLASMA_clange](#)
[PLASMA_dlange](#)
[PLASMA_slange](#)

3.3.2.11 double PLASMA_dlansy (PLASMA_enum *norm*, PLASMA_enum *uplo*, int *N*, double * *A*, int *LDA*, double * *work*)

PLASMA_dlansy returns the value

$dlansy = (\max(\text{abs}(A(i,j))), \text{NORM} = \text{PlasmaMaxNorm} ((\text{norm1}(A), \text{NORM} = \text{PlasmaOneNorm} ((\text{normI}(A), \text{NORM} = \text{PlasmaInfNorm} ((\text{normF}(A), \text{NORM} = \text{PlasmaFrobeniusNorm}$

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that $\max(\text{abs}(A(i,j)))$ is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The number of columns/rows of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The N-by-N matrix A.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,N)$.
- ← *work* double precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

[PLASMA_dlansy_Tile](#)
[PLASMA_dlansy_Tile_Async](#)
[PLASMA_clansy](#)
[PLASMA_dlansy](#)
[PLASMA_slansy](#)

3.3.2.12 int PLASMA_dLapack_to_Tile (double * *Af77*, int *LDA*, PLASMA_desc * *A*)

PLASMA_dLapack_to_Tile - Conversion from LAPACK layout to tile layout.

Parameters

- ← *Af77* LAPACK matrix.
- ← *LDA* The leading dimension of the matrix *Af77*.
- ↔ *A* Descriptor of the PLASMA matrix in tile layout. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, *A->mat* is not used and set to *Af77* when returns, else if PLASMA_TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, *A->mat* has to be allocated before.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dLapack_to_Tile_Async](#)

[PLASMA_dTile_to_Lapack](#)

[PLASMA_cLapack_to_Tile](#)

[PLASMA_dLapack_to_Tile](#)

[PLASMA_sLapack_to_Tile](#)

3.3.2.13 int PLASMA_dlauum (PLASMA_enum uplo, int N, double * A, int LDA)

PLASMA_dlauum - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← *N* The order of the triangular factor U or L. $N \geq 0$.

↔ *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit

<0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dlauum_Tile](#)

[PLASMA_dlauum_Tile_Async](#)

[PLASMA_clauum](#)

[PLASMA_dlauum](#)

[PLASMA_slauum](#)

[PLASMA_dpstri](#)

3.3.2.14 `int PLASMA_dorglq (int M, int N, int K, double * A, int LDA, double * T, double * B, int LDB)`

`PLASMA_dorglq` - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by `PLASMA_dgelqf`.

Parameters

- ← *M* The number of rows of the matrix Q. $M \geq 0$.
- ← *N* The number of columns of the matrix Q. $N \geq M$.
- ← *K* The number of rows of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the LQ factorization of the original matrix A as returned by `PLASMA_dgelqf`.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- ← *T* Auxiliary factorization data, computed by `PLASMA_dgelqf`.
- *B* On exit, the M-by-N matrix Q.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, M)$.

Returns

Return values

- `PLASMA_SUCCESS` successful exit
- `PLASMA_SUCCESS` <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dorglq_Tile](#)
[PLASMA_dorglq_Tile_Async](#)
[PLASMA_cunglq](#)
[PLASMA_dunglq](#)
[PLASMA_sunglq](#)
[PLASMA_dgelqf](#)

3.3.2.15 `int PLASMA_dorgqr (int M, int N, int K, double * A, int LDA, double * T, double * Q, int LDQ)`

`PLASMA_dorgqr` - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by `PLASMA_dgeqrf`.

Parameters

- ← *M* The number of rows of the matrix Q. $M \geq 0$.
- ← *N* The number of columns of the matrix Q. $N \geq M$.
- ← *K* The number of columns of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the QR factorization of the original matrix A as returned by `PLASMA_dgeqrf`.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

- ← *T* Auxiliary factorization data, computed by PLASMA_dgeqrf.
- *Q* On exit, the M-by-N matrix Q.
- ← *LDQ* The leading dimension of the array Q. LDQ $\geq \max(1, M)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dorgqr_Tile](#)
[PLASMA_dorgqr_Tile_Async](#)
[PLASMA_cungqr](#)
[PLASMA_dungqr](#)
[PLASMA_sungqr](#)
[PLASMA_dgeqrf](#)

3.3.2.16 int PLASMA_dormlq (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, double * A, int LDA, double * T, double * B, int LDB)

PLASMA_dormlq - overwrites the general M-by-N matrix C with $Q \cdot C$, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_dgelqf. Q is of order M.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q^T from the left; = PlasmaRight: apply Q or Q^T from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q^T . Currently only PlasmaTrans is supported.
- ← *M* The number of rows of the matrix C. $M \geq 0$.
- ← *N* The number of columns of the matrix C. $N \geq 0$.
- ← *K* The number of rows of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the LQ factorization of the original matrix A as returned by PLASMA_dgelqf.
- ← *LDA* The leading dimension of the array A. LDA $\geq \max(1, K)$.
- ← *T* Auxiliary factorization data, computed by PLASMA_dgelqf.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by $Q \cdot B$ or $Q^T \cdot B$.
- ← *LDB* The leading dimension of the array C. LDB $\geq \max(1, M)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit

<0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dormlq_Tile](#)
[PLASMA_dormlq_Tile_Async](#)
[PLASMA_cunmlq](#)
[PLASMA_dunmlq](#)
[PLASMA_sunmlq](#)
[PLASMA_dgelqf](#)

3.3.2.17 int PLASMA_dormqr (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, double * A, int LDA, double * T, double * B, int LDB)

PLASMA_dormqr - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_dgeqrf. Q is of order M.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q^*T from the left; = PlasmaRight: apply Q or Q^*T from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q^*T . Currently only PlasmaTrans is supported.
- ← *M* The number of rows of the matrix C. $M \geq 0$.
- ← *N* The number of columns of the matrix C. $N \geq 0$.
- ← *K* The number of columns of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$;
- ← *T* Auxiliary factorization data, computed by PLASMA_dgeqrf.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by Q^*B or Q^*T^*B .
- ← *LDB* The leading dimension of the array C. $LDB \geq \max(1, M)$.

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dormqr_Tile](#)
[PLASMA_dormqr_Tile_Async](#)
[PLASMA_cunmqr](#)
[PLASMA_dunmqr](#)
[PLASMA_sunmqr](#)
[PLASMA_dgeqrf](#)

3.3.2.18 int PLASMA_dplgsy (double *bump*, int *N*, double * *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_dplgsy - Generate a random hermitian matrix by tiles.

Parameters

- ← *bump* The value to add to the diagonal to be sure to have a positive definite matrix.
- ← *N* The order of the matrix A. $N \geq 0$.
- *A* On exit, The random hermitian matrix A generated.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dplgsy_Tile](#)
[PLASMA_dplgsy_Tile_Async](#)
[PLASMA_cpigsy](#)
[PLASMA_dpigsy](#)
[PLASMA_spigsy](#)
[PLASMA_dplrnt](#)
[PLASMA_dpigsy](#)

3.3.2.19 int PLASMA_dplrnt (int *M*, int *N*, double * *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_dplrnt - Generate a random matrix by tiles.

Parameters

- ← *M* The number of rows of A.
- ← *N* The order of the matrix A. $N \geq 0$.
- *A* On exit, The random matrix A generated.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dplrnt_Tile](#)
[PLASMA_dplrnt_Tile_Async](#)
[PLASMA_cplrnt](#)
[PLASMA_dplrnt](#)
[PLASMA_splrnt](#)
[PLASMA_dplgsy](#)
[PLASMA_dplgsy](#)

3.3.2.20 int PLASMA_dposv (PLASMA_enum *uplo*, int *N*, int *NRHS*, double * *A*, int *LDA*, double * *B*, int *LDB*)

PLASMA_dposv - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$$A = \begin{cases} U^H \times U, & \text{if } uplo = PlasmaUpper \\ L \times L^H, & \text{if } uplo = PlasmaLower \end{cases}$$

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations $A * X = B$.

Parameters

- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ↔ *A* On entry, the symmetric positive definite (or Hermitian) matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If *UPLO* = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_dposv_Tile](#)
[PLASMA_dposv_Tile_Async](#)
[PLASMA_cposv](#)
[PLASMA_dposv](#)
[PLASMA_sposv](#)

3.3.2.21 int PLASMA_dpotrf (PLASMA_enum uplo, int N, double * A, int LDA)

PLASMA_dpotrf - Computes the Cholesky factorization of a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A. The factorization has the form

$$A = \begin{cases} U^H \times U, & \text{if } uplo = PlasmaUpper \\ L \times L^H, & \text{if } uplo = PlasmaLower \end{cases}$$

where U is an upper triangular matrix and L is a lower triangular matrix.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The order of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the symmetric positive definite (or Hermitian) matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If *UPLO* = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_dpotrf_Tile](#)
[PLASMA_dpotrf_Tile_Async](#)
[PLASMA_cpotrf](#)
[PLASMA_dpotrf](#)
[PLASMA_spotrf](#)
[PLASMA_dpots](#)

3.3.2.22 int PLASMA_dpotri (PLASMA_enum *uplo*, int *N*, double * *A*, int *LDA*)

PLASMA_dpotri - Computes the inverse of a complex Hermitian positive definite matrix *A* using the Cholesky factorization $A = U^*U$ or $A = LL^*$ computed by PLASMA_dpotrf.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.
- ← *N* The order of the matrix *A*. $N \geq 0$.
- ↔ *A* On entry, the triangular factor *U* or *L* from the Cholesky factorization $A = U^*U$ or $A = LL^*$, as computed by PLASMA_dpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of *A*, overwriting the input factor *U* or *L*.
- ← *LDA* The leading dimension of the array *A*. $LDA \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if *i*, the *i*-th argument had an illegal value
- >0 if *i*, the (*i*,*i*) element of the factor *U* or *L* is zero, and the inverse could not be computed.

See also

[PLASMA_dpotri_Tile](#)
[PLASMA_dpotri_Tile_Async](#)
[PLASMA_cpotri](#)
[PLASMA_dpotri](#)
[PLASMA_spotri](#)
[PLASMA_dpotrf](#)

3.3.2.23 int PLASMA_dpotrs (PLASMA_enum *uplo*, int *N*, int *NRHS*, double * *A*, int *LDA*, double * *B*, int *LDB*)

PLASMA_dpotrs - Solves a system of linear equations $A * X = B$ with a symmetric positive definite (or Hermitian positive definite in the complex case) matrix *A* using the Cholesky factorization $A = U^*U$ or $A = LL^*$ computed by PLASMA_dpotrf.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.
- ← *N* The order of the matrix *A*. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix *B*. $NRHS \geq 0$.
- ← *A* The triangular factor *U* or *L* from the Cholesky factorization $A = U^*U$ or $A = LL^*$, computed by PLASMA_dpotrf.
- ← *LDA* The leading dimension of the array *A*. $LDA \geq \max(1, N)$.
- ↔ *B* On entry, the *N*-by-*NRHS* right hand side matrix *B*. On exit, if return value = 0, the *N*-by-*NRHS* solution matrix *X*.
- ← *LDB* The leading dimension of the array *B*. $LDB \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dpotsr_Tile](#)
[PLASMA_dpotsr_Tile_Async](#)
[PLASMA_cpotsr](#)
[PLASMA_dpotsr](#)
[PLASMA_spotsr](#)
[PLASMA_dpotrf](#)

3.3.2.24 int PLASMA_dsgesv (int *N*, int *NRHS*, double * *A*, int *LDA*, double * *B*, int *LDB*, double * *X*, int *LDX*, int * *ITER*)

PLASMA_dsgesv - Computes the solution to a system of linear equations $A * X = B$, where *A* is an *N*-by-*N* matrix and *X* and *B* are *N*-by-*NRHS* matrices.

PLASMA_dsgesv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX*16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX*16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX*16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if $ITER > ITERMAX$ or for all the RHS we have: $RNRM < N * XNRM * ANRM * EPS * BWDMAX$ where:

- *ITER* is the number of the current iteration in the iterative refinement process
- *RNRM* is the infinity-norm of the residual
- *XNRM* is the infinity-norm of the solution
- *ANRM* is the infinity-operator-norm of the matrix *A*
- *EPS* is the machine epsilon returned by DLAMCH('Epsilon').

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

The values *ITERMAX* and *BWDMAX* are fixed to 30 and 1.0D+00 respectively.

Parameters

- ← *N* The number of linear equations, i.e., the order of the matrix *A*. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix *B*. $NRHS \geq 0$.
- ← *A* The *N*-by-*N* coefficient matrix *A*. This matrix is not modified.
- ← *LDA* The leading dimension of the array *A*. $LDA \geq \max(1, N)$.

- ← **B** The N-by-NRHS matrix of right hand side matrix B.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1,N)$.
- **X** If return value = 0, the N-by-NRHS solution matrix X.
- ← **LDX** The leading dimension of the array B. $LDX \geq \max(1,N)$.
- **ITER** The number of the current iteration in the iterative refinement process

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_dsgesv_Tile](#)
[PLASMA_dsgesv_Tile_Async](#)
[PLASMA_dsgesv](#)
[PLASMA_dgesv](#)

3.3.2.25 int PLASMA_dsposv (PLASMA_enum uplo, int N, int NRHS, double * A, int LDA, double * B, int LDB, double * X, int LDX, int * ITER)

PLASMA_dsposv - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$A = U**H * U$, if uplo = PlasmaUpper, or $A = L * L**H$, if uplo = PlasmaLower,

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations $A * X = B$.

PLASMA_dsposv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX*16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX*16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX*16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if $ITER > ITERMAX$ or for all the RHS we have: $RNRM < N * XNRM * ANRM * EPS * BWDMAX$ where:

- ITER is the number of the current iteration in the iterative refinement process
- RNRM is the infinity-norm of the residual
- XNRM is the infinity-norm of the solution
- ANRM is the infinity-operator-norm of the matrix A

- EPS is the machine epsilon returned by DLAMCH('Epsilon').

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

The values ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

Parameters

- ← *N* The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← *A* The N-by-N symmetric positive definite (or Hermitian) coefficient matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. This matrix is not modified.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← *B* The N-by-NRHS matrix of right hand side matrix B.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.
- *X* If return value = 0, the N-by-NRHS solution matrix X.
- ← *LDX* The leading dimension of the array B. $LDX \geq \max(1, N)$.
- *ITER* The number of the current iteration in the iterative refinement process

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_dsposv_Tile](#)
[PLASMA_dsposv_Tile_Async](#)
[PLASMA_dsposv](#)
[PLASMA_zposv](#)

3.3.2.26 int PLASMA_dsungesv (PLASMA_enum trans, int N, int NRHS, double * A, int LDA, double * B, int LDB, double * X, int LDX, int * ITER)

PLASMA_dsungesv - Solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and $M \geq N$: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize $\|B - A * X\|$.

trans = PlasmaNoTrans and $M < N$: find the minimum norm solution of an underdetermined system $A * X = B$.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M -by- $NRHS$ right hand side matrix B and the N -by- $NRHS$ solution matrix X .

`PLASMA_dsungesv` first attempts to factorize the matrix in `COMPLEX` and use this factorization within an iterative refinement procedure to produce a solution with `COMPLEX*16` normwise backward error quality (see below). If the approach fails the method switches to a `COMPLEX*16` factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio `COMPLEX` performance over `COMPLEX*16` performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to `ILAENV` in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if $ITER > ITERMAX$ or for all the RHS we have: $RNRM < N * XNRM * ANRM * EPS * BWDMAX$ where:

- $ITER$ is the number of the current iteration in the iterative refinement process
- $RNRM$ is the infinity-norm of the residual
- $XNRM$ is the infinity-norm of the solution
- $ANRM$ is the infinity-operator-norm of the matrix A
- EPS is the machine epsilon returned by `DLAMCH('Epsilon')`.

Actually, in its current state (`PLASMA 2.1.0`), the test is slightly relaxed.

The values $ITERMAX$ and $BWDMAX$ are fixed to 30 and 1.0D+00 respectively.

We follow Bjorck's algorithm proposed in "Iterative Refinement of Linear Least Squares solutions I", *BIT*, 7:257-278, 1967.4

Parameters

- ← **trans** Intended usage: = `PlasmaNoTrans`: the linear system involves A ; = `PlasmaTrans`: the linear system involves $A**H$. Currently only `PlasmaNoTrans` is supported.
- ← M The number of rows of the matrix A . $M \geq 0$.
- ← N The number of columns of the matrix A . $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrices B and X . $NRHS \geq 0$.
- ← A The M -by- N matrix A . This matrix is not modified.
- ← **LDA** The leading dimension of the array A . $LDA \geq \max(1, M)$.
- ← B The M -by- $NRHS$ matrix B of right hand side vectors, stored columnwise. Not modified.
- ← **LDB** The leading dimension of the array B . $LDB \geq \max(1, M, N)$.
- X If return value = 0, the solution vectors, stored columnwise. if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements $N+1$ to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;
- ← **LDX** The leading dimension of the array B . $LDB \geq \max(1, M, N)$.
- **ITER** The number of the current iteration in the iterative refinement process

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dsungesv_Tile](#)
[PLASMA_dsungesv_Tile_Async](#)
[PLASMA_dsungesv](#)
[PLASMA_zgels](#)

3.3.2.27 int PLASMA_dsymm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, double alpha, double * A, int LDA, double * B, int LDB, double beta, double * C, int LDC)

PLASMA_dsymm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an symmetric matrix and B and C are m by n matrices.

Parameters

← *side* Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← *uplo* Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← *M* Specifies the number of rows of the matrix C. $M \geq 0$.

← *N* Specifies the number of columns of the matrix C. $N \geq 0$.

← *alpha* Specifies the scalar alpha.

← *A* A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← *LDA* The leading dimension of the array A. $LDA \geq \max(1,ka)$.

← *B* B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← *LDB* The leading dimension of the array B. $LDB \geq \max(1,M)$.

← *beta* Specifies the scalar beta.

↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← *LDC* The leading dimension of the array C. $LDC \geq \max(1,M)$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dsymm_Tile](#)

[PLASMA_csymm](#)

[PLASMA_dsymm](#)

[PLASMA_ssymm](#)

3.3.2.28 int PLASMA_dsyr2k (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, double *alpha*, double * *A*, int *LDA*, double * *B*, int *LDB*, double *beta*, double * *C*, int *LDC*)

PLASMA_dsyr2k - Performs one of the symmetric rank 2k operations

$$C = \alpha[op(A) \times g(op(B)')] + \alpha[op(B) \times g(op(A)')] + \beta C$$

, or

$$C = \alpha[g(op(A)') \times op(B)] + \alpha[g(op(B)') \times op(A)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = g(X')

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← **trans** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times g(op(B)')] + \alpha[op(B) \times g(op(A)')] + \beta C$$

= PlasmaTrans:

$$C = \alpha[g(op(A)') \times op(B)] + \alpha[g(op(B)') \times op(A)] + \beta C$$

← **N** N specifies the order of the matrix C. N must be at least zero.

← **K** K specifies the number of columns of the A and B matrices with trans = PlasmaNoTrans. K specifies the number of rows of the A and B matrices with trans = PlasmaTrans.

← **alpha** alpha specifies the scalar alpha.

← **A** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← **LDA** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← **B** B is a LDB-by-kb matrix, where kb is K when trans = PlasmaNoTrans, and is N otherwise.

← **LDB** The leading dimension of the array B. LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).

- ← **beta** beta specifies the scalar beta.
- ↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.
- ← **LDC** The leading dimension of the array C. $LDC \geq \max(1, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_dsyr2k_Tile](#)
[PLASMA_csyr2k](#)
[PLASMA_dsyr2k](#)
[PLASMA_ssyr2k](#)

3.3.2.29 int PLASMA_dsyrk (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, double alpha, double * A, int LDA, double beta, double * C, int LDC)

PLASMA_dsyrk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times g(op(A)')] + \beta C$$

,

where $op(X)$ is one of

$op(X) = X$ or $op(X) = g(X')$

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

- ← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← **trans** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans : A is transposed.
- ← **N** N specifies the order of the matrix C. N must be at least zero.
- ← **K** K specifies the number of columns of the matrix $op(A)$.
- ← **alpha** alpha specifies the scalar alpha.
- ← **A** A is a LDA-by-ka matrix, where ka is K when $trans = PlasmaNoTrans$, and is N otherwise.
- ← **LDA** The leading dimension of the array A. LDA must be at least $\max(1, N)$, otherwise LDA must be at least $\max(1, K)$.
- ← **beta** beta specifies the scalar beta
- ↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.
- ← **LDC** The leading dimension of the array C. $LDC \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dsyrc_Tile](#)

[PLASMA_csyrc](#)

[PLASMA_dsyrc](#)

[PLASMA_ssyrc](#)

3.3.2.30 int PLASMA_dTile_to_Lapack (PLASMA_desc * A, double * Af77, int LDA)

PLASMA_Tile_to_Lapack - Conversion from tile layout to LAPACK layout.

Parameters

← *A* Descriptor of the PLASMA matrix in tile layout.

↔ *Af77* LAPACK matrix. If *PLASMA_TRANSLATION_MODE* is set to *PLASMA_INPLACE*, *Af77* has to be *A->mat*, else if *PLASMA_TRANSLATION_MODE* is set to *PLASMA_OUTOFPLACE*, *Af77* has to be allocated before.

← *LDA* The leading dimension of the matrix *Af77*.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dTile_to_Lapack_Async](#)

[PLASMA_dLapack_to_Tile](#)

[PLASMA_cTile_to_Lapack](#)

[PLASMA_dTile_to_Lapack](#)

[PLASMA_sTile_to_Lapack](#)

3.3.2.31 int PLASMA_dtrmm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, double alpha, double * A, int LDA, double * B, int LDB)

PLASMA_dtrmm - Computes $B = \alpha * \text{op}(A) * B$ or $B = \alpha * B * \text{op}(A)$.

Parameters

← *side* Specifies whether *A* appears on the left or on the right of *X*: = PlasmaLeft: $A * X = B =$
PlasmaRight: $X * A = B$

- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **transA** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.
- ← **diag** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The triangular matrix A. If `uplo = PlasmaUpper`, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If `uplo = PlasmaLower`, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If `diag = PlasmaUnit`, the diagonal elements of A are also not referenced and are assumed to be 1.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dtrmm_Tile](#)
[PLASMA_dtrmm_Tile_Async](#)
[PLASMA_ctrmm](#)
[PLASMA_dtrmm](#)
[PLASMA_strmm](#)

3.3.2.32 int PLASMA_dtrsm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, double alpha, double * A, int LDA, double * B, int LDB)

PLASMA_dtrsm - Computes triangular solve $A * X = B$ or $X * A = B$.

Parameters

- ← **side** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **transA** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

- ← **diag** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The triangular matrix A. If `uplo = PlasmaUpper`, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If `uplo = PlasmaLower`, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If `diag = PlasmaUnit`, the diagonal elements of A are also not referenced and are assumed to be 1.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dtrsm_Tile](#)
[PLASMA_dtrsm_Tile_Async](#)
[PLASMA_ctrsm](#)
[PLASMA_dtrsm](#)
[PLASMA_strsm](#)

3.3.2.33 int PLASMA_dtrsmpl (int N, int NRHS, double * A, int LDA, double * L, int * IPIV, double * B, int LDB)

PLASMA_dtrsmpl - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix.

Parameters

- ← **N** The order of the matrix A. $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← **A** The tile factor L from the factorization, computed by PLASMA_dgetrf.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_dgetrf.
- ← **IPIV** The pivot indices from PLASMA_dgetrf (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_dtrsmpi_Tile](#)
[PLASMA_dtrsmpi_Tile_Async](#)
[PLASMA_ctrsmpl](#)
[PLASMA_dtrsmpi](#)
[PLASMA_strsmpl](#)
[PLASMA_dgetrf](#)

3.3.2.34 int PLASMA_dtrtri (PLASMA_enum *uplo*, PLASMA_enum *diag*, int *N*, double * *A*, int *LDA*)

PLASMA_dtrtri - Computes the inverse of a complex upper or lower triangular matrix A.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
 ← *diag* = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.
 ← *N* The order of the matrix A. $N \geq 0$.
 ↔ *A* On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.
 ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value
 >0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

[PLASMA_dtrtri_Tile](#)
[PLASMA_dtrtri_Tile_Async](#)
[PLASMA_ctrtri](#)
[PLASMA_dtrtri](#)
[PLASMA_strtri](#)
[PLASMA_dpotri](#)

3.4 Simple Interface - Single Real

Functions/Subroutines

- int [PLASMA_sgelqf](#) (int M, int N, float *A, int LDA, float *T)
- int [PLASMA_sgelqs](#) (int M, int N, int NRHS, float *A, int LDA, float *T, float *B, int LDB)
- int [PLASMA_sgels](#) (PLASMA_enum trans, int M, int N, int NRHS, float *A, int LDA, float *T, float *B, int LDB)
- int [PLASMA_sgemm](#) (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, float alpha, float *A, int LDA, float *B, int LDB, float beta, float *C, int LDC)
- int [PLASMA_sgeqrf](#) (int M, int N, float *A, int LDA, float *T)
- int [PLASMA_sgeqrs](#) (int M, int N, int NRHS, float *A, int LDA, float *T, float *B, int LDB)
- int [PLASMA_sgesv](#) (int N, int NRHS, float *A, int LDA, float *L, int *IPIV, float *B, int LDB)
- int [PLASMA_sgetrf](#) (int M, int N, float *A, int LDA, float *L, int *IPIV)
- int [PLASMA_sgetrs](#) (PLASMA_enum trans, int N, int NRHS, float *A, int LDA, float *L, int *IPIV, float *B, int LDB)
- float [PLASMA_slange](#) (PLASMA_enum norm, int M, int N, float *A, int LDA, float *work)
- float [PLASMA_slansy](#) (PLASMA_enum norm, PLASMA_enum uplo, int N, float *A, int LDA, float *work)
- int [PLASMA_slauum](#) (PLASMA_enum uplo, int N, float *A, int LDA)
- int [PLASMA_sorglq](#) (int M, int N, int K, float *A, int LDA, float *T, float *B, int LDB)
- int [PLASMA_sorgqr](#) (int M, int N, int K, float *A, int LDA, float *T, float *Q, int LDQ)
- int [PLASMA_sormlq](#) (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, float *A, int LDA, float *T, float *B, int LDB)
- int [PLASMA_sormqr](#) (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, float *A, int LDA, float *T, float *B, int LDB)
- int [PLASMA_splgsy](#) (float bump, int N, float *A, int LDA, unsigned long long int seed)
- int [PLASMA_splrnt](#) (int M, int N, float *A, int LDA, unsigned long long int seed)
- int [PLASMA_sposv](#) (PLASMA_enum uplo, int N, int NRHS, float *A, int LDA, float *B, int LDB)
- int [PLASMA_spotrf](#) (PLASMA_enum uplo, int N, float *A, int LDA)
- int [PLASMA_spotri](#) (PLASMA_enum uplo, int N, float *A, int LDA)
- int [PLASMA_spotrs](#) (PLASMA_enum uplo, int N, int NRHS, float *A, int LDA, float *B, int LDB)
- int [PLASMA_ssymm](#) (PLASMA_enum side, PLASMA_enum uplo, int M, int N, float alpha, float *A, int LDA, float *B, int LDB, float beta, float *C, int LDC)
- int [PLASMA_ssyr2k](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, float alpha, float *A, int LDA, float *B, int LDB, float beta, float *C, int LDC)
- int [PLASMA_ssyrk](#) (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, float alpha, float *A, int LDA, float beta, float *C, int LDC)
- int [PLASMA_strmm](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, float alpha, float *A, int LDA, float *B, int LDB)
- int [PLASMA_strsm](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, float alpha, float *A, int LDA, float *B, int LDB)
- int [PLASMA_strsmpl](#) (int N, int NRHS, float *A, int LDA, float *L, int *IPIV, float *B, int LDB)
- int [PLASMA_strtri](#) (PLASMA_enum uplo, PLASMA_enum diag, int N, float *A, int LDA)
- int [PLASMA_sLapack_to_Tile](#) (float *Af77, int LDA, PLASMA_desc *A)
- int [PLASMA_sTile_to_Lapack](#) (PLASMA_desc *A, float *Af77, int LDA)

3.4.1 Detailed Description

This is the group of single real functions using the simple user interface.

3.4.2 Function/Subroutine Documentation

3.4.2.1 `int PLASMA_sgelqf (int M, int N, float * A, int LDA, float * T)`

PLASMA_sgelqf - Computes the tile LQ factorization of a complex M-by-N matrix A: $A = L * Q$.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if $M \leq N$); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- *T* On exit, auxiliary factorization data, required by PLASMA_sgelqs to solve the system of equations.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_sgelqf_Tile](#)
[PLASMA_sgelqf_Tile_Async](#)
[PLASMA_cgelqf](#)
[PLASMA_dgelqf](#)
[PLASMA_sgelqf](#)
[PLASMA_sgelqs](#)

3.4.2.2 `int PLASMA_sgelqs (int M, int N, int NRHS, float * A, int LDA, float * T, float * B, int LDB)`

PLASMA_sgelqs - Compute a minimum-norm solution $\min \| A * X - B \|$ using the LQ factorization $A = L * Q$ computed by PLASMA_sgelqf.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq M \geq 0$.
- ← *NRHS* The number of columns of B. $NRHS \geq 0$.
- ← *A* Details of the LQ factorization of the original matrix A as returned by PLASMA_sgelqf.
- ← *LDA* The leading dimension of the array A. $LDA \geq M$.
- ← *T* Auxiliary factorization data, computed by PLASMA_sgelqf.
- ↔ *B* On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

← **LDB** The leading dimension of the array B. $LDB \geq N$.

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_sgelqs_Tile](#)
[PLASMA_sgelqs_Tile_Async](#)
[PLASMA_cgelqs](#)
[PLASMA_dgelqs](#)
[PLASMA_sgelqs](#)
[PLASMA_sgelqf](#)

3.4.2.3 int PLASMA_sgels (PLASMA_enum trans, int M, int N, int NRHS, float * A, int LDA, float * T, float * B, int LDB)

PLASMA_sgels - solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and $M \geq N$: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize $\|B - A*X\|$.

trans = PlasmaNoTrans and $M < N$: find the minimum norm solution of an underdetermined system $A * X = B$.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Parameters

← **trans** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves $A \setminus * T$. Currently only PlasmaNoTrans is supported.

← **M** The number of rows of the matrix A. $M \geq 0$.

← **N** The number of columns of the matrix A. $N \geq 0$.

← **NRHS** The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

↔ **A** On entry, the M-by-N matrix A. On exit, if $M \geq N$, A is overwritten by details of its QR factorization as returned by PLASMA_sgeqrf; if $M < N$, A is overwritten by details of its LQ factorization as returned by PLASMA_sgelqf.

← **LDA** The leading dimension of the array A. $LDA \geq \max(1, M)$.

→ **T** On exit, auxiliary factorization data.

↔ **B** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;

← **LDB** The leading dimension of the array B. $LDB \geq \max(1, M, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_sgels_Tile](#)
[PLASMA_sgels_Tile_Async](#)
[PLASMA_cgels](#)
[PLASMA_dgels](#)
[PLASMA_sgels](#)

3.4.2.4 int PLASMA_sgemm (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, float alpha, float *A, int LDA, float *B, int LDB, float beta, float *C, int LDC)

PLASMA_sgemm - Performs one of the matrix-matrix operations

$$C = \alpha[op(A) \times op(B)] + \beta C$$

where $op(X)$ is one of

$op(X) = X$ or $op(X) = X'$ or $op(X) = g(X')$

alpha and beta are scalars, and A, B and C are matrices, with $op(A)$ an m by k matrix, $op(B)$ a k by n matrix and C an m by n matrix.

Parameters

- ← **transA** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaTrans: A is ugate transposed.
- ← **transB** Specifies whether the matrix B is transposed, not transposed or ugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaTrans: B is ugate transposed.
- ← **M** M specifies the number of rows of the matrix $op(A)$ and of the matrix C. $M \geq 0$.
- ← **N** N specifies the number of columns of the matrix $op(B)$ and of the matrix C. $N \geq 0$.
- ← **K** K specifies the number of columns of the matrix $op(A)$ and the number of rows of the matrix $op(B)$. $K \geq 0$.
- ← **alpha** alpha specifies the scalar alpha
- ← **A** A is a LDA-by-ka matrix, where ka is K when $transA = PlasmaNoTrans$, and is M otherwise.
- ← **LDA** The leading dimension of the array A. $LDA \geq \max(1, M)$.
- ← **B** B is a LDB-by-kb matrix, where kb is N when $transB = PlasmaNoTrans$, and is K otherwise.
- ← **LDB** The leading dimension of the array B. $LDB \geq \max(1, N)$.

- ← *beta* beta specifies the scalar beta
- ↔ *C* *C* is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix ($\alpha * \text{op}(A) * \text{op}(B) + \text{beta} * C$)
- ← *LDC* The leading dimension of the array C. $LDC \geq \max(1, M)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_sgemm_Tile](#)
[PLASMA_cgemm](#)
[PLASMA_dgemm](#)
[PLASMA_sgemm](#)

3.4.2.5 int PLASMA_sgeqrf (int M, int N, float * A, int LDA, float * T)

PLASMA_sgeqrf - Computes the tile QR factorization of a complex M-by-N matrix A: $A = Q * R$.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the $\min(M, N)$ -by-N upper trapezoidal matrix R (R is upper triangular if $M \geq N$); the elements below the diagonal represent the unitary matrix Q as a product of elementary reflectors stored by tiles.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- *T* On exit, auxiliary factorization data, required by PLASMA_sgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_sgeqrf_Tile](#)
[PLASMA_sgeqrf_Tile_Async](#)
[PLASMA_cgeqrf](#)
[PLASMA_dgeqrf](#)
[PLASMA_sgeqrf](#)
[PLASMA_sgeqrs](#)

3.4.2.6 `int PLASMA_sgeqrs (int M, int N, int NRHS, float * A, int LDA, float * T, float * B, int LDB)`

`PLASMA_sgeqrs` - Compute a minimum-norm solution $\min \| A * X - B \|$ using the RQ factorization $A = R * Q$ computed by `PLASMA_sgeqrf`.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq M \geq 0$.
- ← *NRHS* The number of columns of B. $NRHS \geq 0$.
- ↔ *A* Details of the QR factorization of the original matrix A as returned by `PLASMA_sgeqrf`.
- ← *LDA* The leading dimension of the array A. $LDA \geq M$.
- ← *T* Auxiliary factorization data, computed by `PLASMA_sgeqrf`.
- ↔ *B* On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns

Return values

- `PLASMA_SUCCESS` successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_sgeqrs_Tile](#)
[PLASMA_sgeqrs_Tile_Async](#)
[PLASMA_cgeqrs](#)
[PLASMA_dgeqrs](#)
[PLASMA_sgeqrs](#)
[PLASMA_sgeqrf](#)

3.4.2.7 `int PLASMA_sgesv (int N, int NRHS, float * A, int LDA, float * L, int * IPIV, float * B, int LDB)`

`PLASMA_sgesv` - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The tile LU decomposition with partial tile pivoting and row interchanges is used to factor A. The factored form of A is then used to solve the system of equations $A * X = B$.

Parameters

- ← *N* The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ↔ *A* On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.

- *L* On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.
- *IPIV* On exit, the pivot indices that define the permutations (not equivalent to LAPACK).
- ↔ *B* On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1,N)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_sgesv_Tile](#)
[PLASMA_sgesv_Tile_Async](#)
[PLASMA_cgesv](#)
[PLASMA_dgesv](#)
[PLASMA_sgesv](#)
[PLASMA_scgesv](#)

3.4.2.8 int PLASMA_sgetrf (int M, int N, float * A, int LDA, float * L, int * IPIV)

PLASMA_sgetrf - Computes an LU factorization of a general M-by-N matrix A using the tile LU algorithm with partial tile pivoting with row interchanges.

Parameters

- ← *M* The number of rows of the matrix A. $M \geq 0$.
- ← *N* The number of columns of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,M)$.
- *L* On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_sgetrs to solve the system of equations.
- *IPIV* The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

>0 if i , $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

See also

[PLASMA_sgetrf_Tile](#)
[PLASMA_sgetrf_Tile_Async](#)
[PLASMA_cgetrf](#)
[PLASMA_dgetrf](#)
[PLASMA_sgetrf](#)
[PLASMA_sgetrs](#)

3.4.2.9 int PLASMA_sgetrs (PLASMA_enum trans, int N, int NRHS, float * A, int LDA, float * L, int * IPIV, float * B, int LDB)

PLASMA_sgetrs - Solves a system of linear equations $A * X = B$, with a general N -by- N matrix A using the tile LU factorization computed by PLASMA_sgetrf.

Parameters

- ← **trans** Intended to specify the the form of the system of equations: = PlasmaNoTrans: $A * X = B$ (No transpose) = PlasmaTrans: $A^{**T} * X = B$ (Transpose) = PlasmaTrans: $A \setminus * \setminus * T * X = B$ (Conjugate transpose) Currently only PlasmaNoTrans is supported.
- ← **N** The order of the matrix A . $N \geq 0$.
- ← **NRHS** The number of right hand sides, i.e., the number of columns of the matrix B . $NRHS \geq 0$.
- ← **A** The tile factors L and U from the factorization, computed by PLASMA_sgetrf.
- ← **LDA** The leading dimension of the array A . $LDA \geq \max(1,N)$.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_sgetrf.
- ← **IPIV** The pivot indices from PLASMA_sgetrf (not equivalent to LAPACK).
- ← **B** On entry, the N -by- $NRHS$ matrix of right hand side matrix B . On exit, the solution matrix X .
- ← **LDB** The leading dimension of the array B . $LDB \geq \max(1,N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

Returns

<0 if $-i$, the i -th argument had an illegal value

See also

[PLASMA_sgetrs_Tile](#)
[PLASMA_sgetrs_Tile_Async](#)
[PLASMA_cgetrs](#)
[PLASMA_dgetrs](#)
[PLASMA_sgetrs](#)
[PLASMA_sgetrf](#)

3.4.2.10 float PLASMA_slange (PLASMA_enum *norm*, int *M*, int *N*, float * *A*, int *LDA*, float * *work*)

PLASMA_slange returns the value

slange = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← *M* The number of rows of the matrix A. $M \geq 0$. When $M = 0$, the returned value is set to zero.
- ← *N* The number of columns of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The M-by-N matrix A.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.
- ← *work* float precision array of dimension (MAX(1,LWORK)), where $LWORK \geq M$ when $NORM = PlasmaInfNorm$; otherwise, WORK is not referenced.

Returns

Return values

the norm described above.

See also

[PLASMA_slange_Tile](#)
[PLASMA_slange_Tile_Async](#)
[PLASMA_clange](#)
[PLASMA_dlange](#)
[PLASMA_slange](#)

3.4.2.11 float PLASMA_slansy (PLASMA_enum *norm*, PLASMA_enum *uplo*, int *N*, float * *A*, int *LDA*, float * *work*)

PLASMA_slansy returns the value

slansy = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

- ← *norm* = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The number of columns/rows of the matrix A. $N \geq 0$. When $N = 0$, the returned value is set to zero.
- ← *A* The N-by-N matrix A.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ← *work* float precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

[PLASMA_slansy_Tile](#)
[PLASMA_slansy_Tile_Async](#)
[PLASMA_clansy](#)
[PLASMA_dlansy](#)
[PLASMA_slansy](#)

3.4.2.12 int PLASMA_sLapack_to_Tile (float *Af77, int LDA, PLASMA_desc *A)

PLASMA_sLapack_to_Tile - Conversion from LAPACK layout to tile layout.

Parameters

- ← *Af77* LAPACK matrix.
- ← *LDA* The leading dimension of the matrix Af77.
- ↔ *A* Descriptor of the PLASMA matrix in tile layout. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, A->mat is not used and set to Af77 when returns, else if PLASMA_TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, A->mat has to be allocated before.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_sLapack_to_Tile_Async](#)
[PLASMA_sTile_to_Lapack](#)
[PLASMA_cLapack_to_Tile](#)
[PLASMA_dLapack_to_Tile](#)
[PLASMA_sLapack_to_Tile](#)

3.4.2.13 `int PLASMA_slauum (PLASMA_enum uplo, int N, float * A, int LDA)`

`PLASMA_slauum` - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A .

If `UPLO = 'U'` or `'u'` then the upper triangle of the result is stored, overwriting the factor U in A . If `UPLO = 'L'` or `'l'` then the lower triangle of the result is stored, overwriting the factor L in A .

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← N The order of the triangular factor U or L . $N \geq 0$.
- ↔ A On entry, the triangular factor U or L . On exit, if `UPLO = 'U'`, the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if `UPLO = 'L'`, the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.
- ← *LDA* The leading dimension of the array A . $LDA \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if $-i$, the i -th argument had an illegal value

See also

[PLASMA_slauum_Tile](#)
[PLASMA_slauum_Tile_Async](#)
[PLASMA_clauum](#)
[PLASMA_dlauum](#)
[PLASMA_slauum](#)
[PLASMA_spotri](#)

3.4.2.14 `int PLASMA_sorglq (int M, int N, int K, float * A, int LDA, float * T, float * B, int LDB)`

`PLASMA_sorglq` - Generates an M -by- N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by `PLASMA_sgelqf`.

Parameters

- ← M The number of rows of the matrix Q . $M \geq 0$.
- ← N The number of columns of the matrix Q . $N \geq M$.
- ← K The number of rows of elementary tile reflectors whose product defines the matrix Q . $M \geq K \geq 0$.
- ← A Details of the LQ factorization of the original matrix A as returned by `PLASMA_sgelqf`.
- ← *LDA* The leading dimension of the array A . $LDA \geq \max(1, M)$.
- ← T Auxiliary factorization data, computed by `PLASMA_sgelqf`.
- B On exit, the M -by- N matrix Q .

← *LDA* The leading dimension of the array B. $LDB \geq \max(1, M)$.

Returns

Return values

PLASMA_SUCCESS successful exit

PLASMA_SUCCESS <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_sorglq_Tile](#)
[PLASMA_sorglq_Tile_Async](#)
[PLASMA_cunglq](#)
[PLASMA_dunglq](#)
[PLASMA_sunglq](#)
[PLASMA_sgelqf](#)

3.4.2.15 int PLASMA_sorgqr (int *M*, int *N*, int *K*, float * *A*, int *LDA*, float * *T*, float * *Q*, int *LDQ*)

PLASMA_sorgqr - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_sgeqrf.

Parameters

← *M* The number of rows of the matrix Q. $M \geq 0$.

← *N* The number of columns of the matrix Q. $N \geq M$.

← *K* The number of columns of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.

← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_sgeqrf.

← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

← *T* Auxiliary factorization data, computed by PLASMA_sgeqrf.

→ *Q* On exit, the M-by-N matrix Q.

← *LDQ* The leading dimension of the array Q. $LDQ \geq \max(1, M)$.

Returns

Return values

PLASMA_SUCCESS successful exit

<0 if -i, the i-th argument had an illegal value

See also

[PLASMA_sorgqr_Tile](#)
[PLASMA_sorgqr_Tile_Async](#)
[PLASMA_cungqr](#)
[PLASMA_dungqr](#)
[PLASMA_sungqr](#)
[PLASMA_sgeqrf](#)

3.4.2.16 `int PLASMA_sormlq (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, float * A, int LDA, float * T, float * B, int LDB)`

`PLASMA_sormlq` - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by `PLASMA_sgelqf`. Q is of order M.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q^T from the left; = PlasmaRight: apply Q or Q^T from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q^T . Currently only PlasmaTrans is supported.
- ← *M* The number of rows of the matrix C. $M \geq 0$.
- ← *N* The number of columns of the matrix C. $N \geq 0$.
- ← *K* The number of rows of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the LQ factorization of the original matrix A as returned by `PLASMA_sgelqf`.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, K)$.
- ← *T* Auxiliary factorization data, computed by `PLASMA_sgelqf`.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by $Q \cdot B$ or $Q^T \cdot B$.
- ← *LDB* The leading dimension of the array C. $LDB \geq \max(1, M)$.

Returns

Return values

- `PLASMA_SUCCESS` successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_sormlq_Tile](#)
[PLASMA_sormlq_Tile_Async](#)
[PLASMA_cunmlq](#)
[PLASMA_dunmlq](#)
[PLASMA_sunmlq](#)
[PLASMA_sgelqf](#)

3.4.2.17 `int PLASMA_sormqr (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, float * A, int LDA, float * T, float * B, int LDB)`

`PLASMA_sormqr` - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by `PLASMA_sgeqrf`. Q is of order M.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q^T from the left; = PlasmaRight: apply Q or Q^T from the right. Currently only PlasmaLeft is supported.

- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q^*T . Currently only PlasmaTrans is supported.
- ← *M* The number of rows of the matrix C. $M \geq 0$.
- ← *N* The number of columns of the matrix C. $N \geq 0$.
- ← *K* The number of columns of elementary tile reflectors whose product defines the matrix Q. $M \geq K \geq 0$.
- ← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_sgeqrf.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$;
- ← *T* Auxiliary factorization data, computed by PLASMA_sgeqrf.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by Q^*B or Q^*T^*B .
- ← *LDB* The leading dimension of the array C. $LDB \geq \max(1, M)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

- [PLASMA_sormqr_Tile](#)
- [PLASMA_sormqr_Tile_Async](#)
- [PLASMA_cunmqr](#)
- [PLASMA_dunmqr](#)
- [PLASMA_sunmqr](#)
- [PLASMA_sgeqrf](#)

3.4.2.18 int PLASMA_splgsy (float bump, int N, float * A, int LDA, unsigned long long int seed)

PLASMA_splgsy - Generate a random hermitian matrix by tiles.

Parameters

- ← *bump* The value to add to the diagonal to be sure to have a positive definite matrix.
- ← *N* The order of the matrix A. $N \geq 0$.
- *A* On exit, The random hermitian matrix A generated.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_splgsy_Tile](#)
[PLASMA_splgsy_Tile_Async](#)
[PLASMA_cplgsy](#)
[PLASMA_dplgsy](#)
[PLASMA_splgsy](#)
[PLASMA_splrnt](#)
[PLASMA_splgsy](#)

3.4.2.19 int PLASMA_splrnt (int M, int N, float * A, int LDA, unsigned long long int seed)

PLASMA_splrnt - Generate a random matrix by tiles.

Parameters

← *M* The number of rows of A.
 ← *N* The order of the matrix A. $N \geq 0$.
 → *A* On exit, The random matrix A generated.
 ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, M)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_splrnt_Tile](#)
[PLASMA_splrnt_Tile_Async](#)
[PLASMA_cplrnt](#)
[PLASMA_dplrnt](#)
[PLASMA_splrnt](#)
[PLASMA_splgsy](#)
[PLASMA_splgsy](#)

3.4.2.20 int PLASMA_sposv (PLASMA_enum uplo, int N, int NRHS, float * A, int LDA, float * B, int LDB)

PLASMA_sposv - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$$A = \begin{cases} U^H \times U, & \text{if uplo=PlasmaUpper} \\ L \times L^H, & \text{if uplo=PlasmaLower} \end{cases}$$

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations $A * X = B$.

Parameters

- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ↔ *A* On entry, the symmetric positive definite (or Hermitian) matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If *UPLO* = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U^H U$ or $A = L L^H$.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_sposv_Tile](#)
[PLASMA_sposv_Tile_Async](#)
[PLASMA_cposv](#)
[PLASMA_dposv](#)
[PLASMA_sposv](#)

3.4.2.21 int PLASMA_spotrf (PLASMA_enum uplo, int N, float * A, int LDA)

PLASMA_spotrf - Computes the Cholesky factorization of a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A. The factorization has the form

$$A = \begin{cases} U^H \times U, & \text{if } uplo = PlasmaUpper \\ L \times L^H, & \text{if } uplo = PlasmaLower \end{cases}$$

where U is an upper triangular matrix and L is a lower triangular matrix.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The order of the matrix A. $N \geq 0$.

↔ **A** On entry, the symmetric positive definite (or Hermitian) matrix A. If `uplo = PlasmaUpper`, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If `UPLO = 'L'`, the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U^*U$ or $A = LL^*$.

← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

<0 if -i, the i-th argument had an illegal value

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_spotrf_Tile](#)
[PLASMA_spotrf_Tile_Async](#)
[PLASMA_cpotrf](#)
[PLASMA_dpotrf](#)
[PLASMA_spotrf](#)
[PLASMA_spotrs](#)

3.4.2.22 int PLASMA_spotri (PLASMA_enum uplo, int N, float * A, int LDA)

PLASMA_spotri - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U^*U$ or $A = LL^*$ computed by **PLASMA_spotrf**.

Parameters

← **uplo** = `PlasmaUpper`: Upper triangle of A is stored; = `PlasmaLower`: Lower triangle of A is stored.

← **N** The order of the matrix A. $N \geq 0$.

↔ **A** On entry, the triangular factor U or L from the Cholesky factorization $A = U^*U$ or $A = LL^*$, as computed by **PLASMA_spotrf**. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

← **LDA** The leading dimension of the array A. $LDA \geq \max(1, N)$.

Returns

Return values

PLASMA_SUCCESS successful exit

<0 if -i, the i-th argument had an illegal value

>0 if i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

See also

[PLASMA_spotri_Tile](#)
[PLASMA_spotri_Tile_Async](#)
[PLASMA_cpotri](#)
[PLASMA_dpotri](#)
[PLASMA_spotri](#)
[PLASMA_spotrf](#)

3.4.2.23 `int PLASMA_spotrs (PLASMA_enum uplo, int N, int NRHS, float * A, int LDA, float * B, int LDB)`

PLASMA_spotrs - Solves a system of linear equations $A * X = B$ with a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A using the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$ computed by PLASMA_spotrf.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *N* The order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← *A* The triangular factor U or L from the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$, computed by PLASMA_spotrf.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1, N)$.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_spotrs_Tile](#)
[PLASMA_spotrs_Tile_Async](#)
[PLASMA_cpotrs](#)
[PLASMA_dpotrs](#)
[PLASMA_spotrs](#)
[PLASMA_spotrf](#)

3.4.2.24 `int PLASMA_ssymm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, float alpha, float * A, int LDA, float * B, int LDB, float beta, float * C, int LDC)`

PLASMA_ssymm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an symmetric matrix and B and C are m by n matrices.

Parameters

← *side* Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← *uplo* Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← *M* Specifies the number of rows of the matrix C. $M \geq 0$.

← *N* Specifies the number of columns of the matrix C. $N \geq 0$.

← *alpha* Specifies the scalar alpha.

← *A* A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← *LDA* The leading dimension of the array A. $LDA \geq \max(1,ka)$.

← *B* B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← *LDB* The leading dimension of the array B. $LDB \geq \max(1,M)$.

← *beta* Specifies the scalar beta.

↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← *LDC* The leading dimension of the array C. $LDC \geq \max(1,M)$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_ssymm_Tile](#)
[PLASMA_csymm](#)
[PLASMA_dsymm](#)
[PLASMA_ssymm](#)

3.4.2.25 int PLASMA_ssy2k (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, float *alpha*, float * *A*, int *LDA*, float * *B*, int *LDB*, float *beta*, float * *C*, int *LDC*)

PLASMA_ssy2k - Performs one of the symmetric rank 2k operations

$$C = \alpha[op(A) \times g(op(B)')] + \alpha[op(B) \times g(op(A)')] + \beta C$$

, or

$$C = \alpha[g(op(A)') \times op(B)] + \alpha[g(op(B)') \times op(A)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = g(X')

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← **trans** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times g(op(B)')] + \alpha[op(B) \times g(op(A)')] + \beta C$$

= PlasmaTrans:

$$C = \alpha[g(op(A)') \times op(B)] + \alpha[g(op(B)') \times op(A)] + \beta C$$

← **N** N specifies the order of the matrix C. N must be at least zero.

← **K** K specifies the number of columns of the A and B matrices with trans = PlasmaNoTrans. K specifies the number of rows of the A and B matrices with trans = PlasmaTrans.

← **alpha** alpha specifies the scalar alpha.

← **A** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← **LDA** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← **B** B is a LDB-by-kb matrix, where kb is K when trans = PlasmaNoTrans, and is N otherwise.

← **LDB** The leading dimension of the array B. LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).

← **beta** beta specifies the scalar beta.

↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← **LDC** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_ssy2k_Tile](#)

[PLASMA_csyr2k](#)

[PLASMA_dsyr2k](#)

[PLASMA_ssy2k](#)

3.4.2.26 int PLASMA_ssyrc (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, float *alpha*, float * *A*, int *LDA*, float *beta*, float * *C*, int *LDC*)

PLASMA_ssyrc - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times g(op(A)')] + \beta C$$

where op(X) is one of

op(X) = X or op(X) = g(X')

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans : A is transposed.
- ← *N* N specifies the order of the matrix C. N must be at least zero.
- ← *K* K specifies the number of columns of the matrix op(A).
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← *LDA* The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.
- ← *LDC* The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_ssyrc_Tile](#)
[PLASMA_csyrc](#)
[PLASMA_dsyrc](#)
[PLASMA_ssyrc](#)

3.4.2.27 int PLASMA_sTile_to_Lapack (PLASMA_desc * *A*, float * *Af77*, int *LDA*)

PLASMA_Tile_to_Lapack - Conversion from tile layout to LAPACK layout.

Parameters

- ← *A* Descriptor of the PLASMA matrix in tile layout.

- ↔ *Af77* LAPACK matrix. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, *Af77* has to be A->mat, else if PLASMA_TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, *Af77* has to be allocated before.
- ← *LDA* The leading dimension of the matrix *Af77*.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_sTile_to_Lapack_Async](#)
[PLASMA_sLapack_to_Tile](#)
[PLASMA_cTile_to_Lapack](#)
[PLASMA_dTile_to_Lapack](#)
[PLASMA_sTile_to_Lapack](#)

3.4.2.28 int PLASMA_strmm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, float *alpha*, float * *A*, int *LDA*, float * *B*, int *LDB*)

PLASMA_strmm - Computes $B = \alpha * \text{op}(A) * B$ or $B = \alpha * B * \text{op}(A)$.

Parameters

- ← *side* Specifies whether *A* appears on the left or on the right of X : = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ← *uplo* Specifies whether the matrix *A* is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.
- ← *transA* Specifies whether the matrix *A* is transposed, not transposed or ugate transposed: = PlasmaNoTrans: *A* is transposed; = PlasmaTrans: *A* is not transposed; = PlasmaTrans: *A* is ugate transposed.
- ← *diag* Specifies whether or not *A* is unit triangular: = PlasmaNonUnit: *A* is non unit; = PlasmaUnit: *A* is unit.
- ← *N* The order of the matrix *A*. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix *B*. $NRHS \geq 0$.
- ← *A* The triangular matrix *A*. If *uplo* = PlasmaUpper, the leading *N*-by-*N* upper triangular part of the array *A* contains the upper triangular matrix, and the strictly lower triangular part of *A* is not referenced. If *uplo* = PlasmaLower, the leading *N*-by-*N* lower triangular part of the array *A* contains the lower triangular matrix, and the strictly upper triangular part of *A* is not referenced. If *diag* = PlasmaUnit, the diagonal elements of *A* are also not referenced and are assumed to be 1.
- ← *LDA* The leading dimension of the array *A*. $LDA \geq \max(1, N)$.
- ↔ *B* On entry, the *N*-by-*NRHS* right hand side matrix *B*. On exit, if return value = 0, the *N*-by-*NRHS* solution matrix *X*.
- ← *LDB* The leading dimension of the array *B*. $LDB \geq \max(1, N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_strmm_Tile](#)
[PLASMA_strmm_Tile_Async](#)
[PLASMA_ctrmm](#)
[PLASMA_dtrmm](#)
[PLASMA_strmm](#)

3.4.2.29 int PLASMA_strsm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, float *alpha*, float * *A*, int *LDA*, float * *B*, int *LDB*)

PLASMA_strsm - Computes triangular solve $A*X = B$ or $X*A = B$.

Parameters

- ← *side* Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A*X = B$ = PlasmaRight: $X*A = B$
- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *transA* Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.
- ← *diag* Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← *N* The order of the matrix A. $N \geq 0$.
- ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- ← *A* The triangular matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If *uplo* = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If *diag* = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,N)$.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
- ← *LDB* The leading dimension of the array B. $LDB \geq \max(1,N)$.

Returns

Return values

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_strsm_Tile](#)
[PLASMA_strsm_Tile_Async](#)
[PLASMA_ctrsm](#)
[PLASMA_dtrsm](#)
[PLASMA_strsm](#)

3.4.2.30 int PLASMA_strsmpl (int *N*, int *NRHS*, float * *A*, int *LDA*, float * *L*, int * *IPIV*, float * *B*, int *LDB*)

PLASMA_strsmpl - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix.

Parameters

← *N* The order of the matrix A. $N \geq 0$.
 ← *NRHS* The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
 ← *A* The tile factor L from the factorization, computed by PLASMA_sgetrf.
 ← *LDA* The leading dimension of the array A. $LDA \geq \max(1,N)$.
 ← *L* Auxiliary factorization data, related to the tile L factor, computed by PLASMA_sgetrf.
 ← *IPIV* The pivot indices from PLASMA_sgetrf (not equivalent to LAPACK).
 ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.
 ← *LDB* The leading dimension of the array B. $LDB \geq \max(1,N)$.

Returns**Return values**

PLASMA_SUCCESS successful exit
 <0 if -i, the i-th argument had an illegal value

See also

[PLASMA_strsmpl_Tile](#)
[PLASMA_strsmpl_Tile_Async](#)
[PLASMA_ctrsmpl](#)
[PLASMA_dtrsmpl](#)
[PLASMA_strsmpl](#)
[PLASMA_sgetrf](#)

3.4.2.31 int PLASMA_strtri (PLASMA_enum *uplo*, PLASMA_enum *diag*, int *N*, float * *A*, int *LDA*)

PLASMA_strtri - Computes the inverse of a complex upper or lower triangular matrix A.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *diag* = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.
- ← *N* The order of the matrix A. $N \geq 0$.
- ↔ *A* On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.
- ← *LDA* The leading dimension of the array A. $LDA \geq \max(1, N)$.

Returns

Return values

- PLASMA_SUCCESS* successful exit
- <0 if -i, the i-th argument had an illegal value
- >0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

[PLASMA_strtri_Tile](#)
[PLASMA_strtri_Tile_Async](#)
[PLASMA_ctrtri](#)
[PLASMA_dtrtri](#)
[PLASMA_strtri](#)
[PLASMA_spotri](#)

3.5 Advanced Interface: Synchronous - Double Complex

Functions/Subroutines

- int [PLASMA_zcgels_Tile](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_desc *X, int *ITER)
- int [PLASMA_zcgsv_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_desc *X, int *ITER)
- int [PLASMA_zcposv_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_desc *X, int *ITER)
- int [PLASMA_zcungsv_Tile](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_desc *X, int *ITER)
- int [PLASMA_zgelqf_Tile](#) (PLASMA_desc *A, PLASMA_desc *T)
- int [PLASMA_zgelqs_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_zgels_Tile](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_zgemm_Tile](#) (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex64_t beta, PLASMA_desc *C)
- int [PLASMA_zgeqrf_Tile](#) (PLASMA_desc *A, PLASMA_desc *T)
- int [PLASMA_zgeqrs_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_zgesv_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int [PLASMA_zgetrf_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV)
- int [PLASMA_zgetrs_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int [PLASMA_zhemm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex64_t beta, PLASMA_desc *C)
- int [PLASMA_zher2k_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, double beta, PLASMA_desc *C)
- int [PLASMA_zherk_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc *A, double beta, PLASMA_desc *C)
- double [PLASMA_zlange_Tile](#) (PLASMA_enum norm, PLASMA_desc *A, double *work)
- double [PLASMA_zlanhe_Tile](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, double *work)
- double [PLASMA_zlansy_Tile](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, double *work)
- int [PLASMA_zlauum_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_zplghe_Tile](#) (double bump, PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_zpLgsy_Tile](#) (PLASMA_Complex64_t bump, PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_zplrnt_Tile](#) (PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_zposv_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_zpotrf_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_zpotri_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_zpotrs_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_zsymm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex64_t beta, PLASMA_desc *C)
- int [PLASMA_zsyr2k_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex64_t beta, PLASMA_desc *C)
- int [PLASMA_zsyrk_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_Complex64_t beta, PLASMA_desc *C)

- int `PLASMA_ztrmm_Tile` (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B)
- int `PLASMA_ztrsm_Tile` (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B)
- int `PLASMA_ztrsml_Tile` (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int `PLASMA_ztrtri_Tile` (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A)
- int `PLASMA_zunglq_Tile` (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int `PLASMA_zungqr_Tile` (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q)
- int `PLASMA_zunmlq_Tile` (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int `PLASMA_zunmqr_Tile` (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)

3.5.1 Detailed Description

This is the group of double complex functions using the advanced synchronous user interface.

3.5.2 Function/Subroutine Documentation

3.5.2.1 int `PLASMA_zcgels_Tile` (PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*, PLASMA_desc * *X*, int * *ITER*)

`PLASMA_zcgels_Tile` - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Tile equivalent of `PLASMA_zcgelsv()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *trans* Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A**H. Currently only PlasmaNoTrans is supported.
- ↔ *A*
 - If the iterative refinement converged, A is not modified;
 - otherwise, it fell back to double precision solution, and on exit the M-by-N matrix A contains: if $M \geq N$, A is overwritten by details of its QR factorization as returned by `PLASMA_zgeqrf`; if $M < N$, A is overwritten by details of its LQ factorization as returned by `PLASMA_zgelqf`.
- *T* On exit:
 - if the iterative refinement converged, T is not modified;
 - otherwise, it fell back to double precision solution, and then T is an auxiliary factorization data.
- ↔ *B* On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zcgels](#)
[PLASMA_zcgels_Tile_Async](#)
[PLASMA_dsgels_Tile](#)
[PLASMA_zgels_Tile](#)

3.5.2.2 int PLASMA_zcgesv_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_desc * X, int * ITER)

PLASMA_zcgesv_Tile - Solves a system of linear equations using the tile LU factorization and mixed-precision iterative refinement. Tile equivalent of [PLASMA_zcgesv\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ **A** On entry, the N-by-N coefficient matrix A.

- If the iterative refinement converged, A is not modified;
- otherwise, it fell back to double precision solution, and then A contains the tile L and U factors from the factorization (not equivalent to LAPACK).

→ **L** On exit:

- if the iterative refinement converged, L is not modified;
- otherwise, it fell back to double precision solution, and then L is an auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations (not equivalent to LAPACK).

→ **IPIV** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

>0 if $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_zcgesv](#)
[PLASMA_zcgesv_Tile_Async](#)
[PLASMA_dsgesv_Tile](#)
[PLASMA_zgesv_Tile](#)

3.5.2.3 `int PLASMA_zcposv_Tile (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_desc * X, int * ITER)`

`PLASMA_zcposv_Tile` - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization and mixed-precision iterative refinement. Tile equivalent of `PLASMA_zcposv()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ↔ *A* On entry, the N-by-N symmetric positive definite (or Hermitian) coefficient matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If *UPLO* = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.
 - If the iterative refinement converged, A is not modified;
 - otherwise, it failed backed to double precision solution,
- ↔ *B* On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

>0 if *i*, the leading minor of order *i* of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_zcposv](#)
[PLASMA_zcposv_Tile_Async](#)
[PLASMA_dsposv_Tile](#)
[PLASMA_zposv_Tile](#)

3.5.2.4 `int PLASMA_zcungesv_Tile (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_desc * X, int * ITER)`

`PLASMA_zcungesv_Tile` - Solves symmetric linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Tile equivalent of `PLASMA_zcungesv()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *trans* Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A**H. Currently only PlasmaNoTrans is supported.
- ↔ *A*
 - If the iterative refinement converged, A is not modified;

- otherwise, it fell back to double precision solution, and on exit the M-by-N matrix A contains: if $M \geq N$, A is overwritten by details of its QR factorization as returned by `PLASMA_zgeqrf`; if $M < N$, A is overwritten by details of its LQ factorization as returned by `PLASMA_zgelqf`.

→ T On exit:

- if the iterative refinement converged, T is not modified;
- otherwise, it fell back to double precision solution, and then T is an auxiliary factorization data.

↔ B On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements $N+1$ to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zcungesv](#)
[PLASMA_zcungesv_Tile_Async](#)
[PLASMA_dsungesv_Tile](#)
[PLASMA_zgels_Tile](#)

3.5.2.5 int PLASMA_zgelqf_Tile (PLASMA_desc * A, PLASMA_desc * T)

`PLASMA_zgelqf_Tile` - Computes the tile LQ factorization of a matrix. Tile equivalent of [PLASMA_zgelqf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ A On entry, the M-by-N matrix A . On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if $M \leq N$); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

→ T On exit, auxiliary factorization data, required by `PLASMA_zgelqs` to solve the system of equations.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zgelqf](#)

[PLASMA_zgelqf_Tile_Async](#)
[PLASMA_cgelqf_Tile](#)
[PLASMA_dgelqf_Tile](#)
[PLASMA_sgelqf_Tile](#)
[PLASMA_zgelqs_Tile](#)

3.5.2.6 int PLASMA_zgelqs_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)

PLASMA_zgelqs_Tile - Computes a minimum-norm solution using previously computed LQ factorization. Tile equivalent of [PLASMA_zgelqs\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **A** Details of the LQ factorization of the original matrix A as returned by [PLASMA_zgelqf](#).
- ← **T** Auxiliary factorization data, computed by [PLASMA_zgelqf](#).
- ↔ **B** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zgelqs](#)
[PLASMA_zgelqs_Tile_Async](#)
[PLASMA_cgelqs_Tile](#)
[PLASMA_dgelqs_Tile](#)
[PLASMA_sgelqs_Tile](#)
[PLASMA_zgelqf_Tile](#)

3.5.2.7 int PLASMA_zgels_Tile (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)

PLASMA_zgels_Tile - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Tile equivalent of [PLASMA_zgels\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **trans** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A**H. Currently only PlasmaNoTrans is supported.
- ↔ **A** On entry, the M-by-N matrix A. On exit, if $M \geq N$, A is overwritten by details of its QR factorization as returned by [PLASMA_zgeqrf](#); if $M < N$, A is overwritten by details of its LQ factorization as returned by [PLASMA_zgelqf](#).
- **T** On exit, auxiliary factorization data.

↔ **B** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;

Returns

PLASMA_SUCCESS successful exit

See also

[PLASMA_zgels](#)
[PLASMA_zgels_Tile_Async](#)
[PLASMA_cgels_Tile](#)
[PLASMA_dgels_Tile](#)
[PLASMA_sgels_Tile](#)

3.5.2.8 int PLASMA_zgemm_Tile (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex64_t beta, PLASMA_desc * C)

PLASMA_zgemm_Tile - Performs matrix multiplication. Tile equivalent of [PLASMA_zgemm\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **transA** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← **transB** Specifies whether the matrix B is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaConjTrans: B is conjugate transposed.
- ← **alpha** alpha specifies the scalar alpha
- ← **A** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.
- ← **B** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.
- ← **beta** beta specifies the scalar beta
- ↔ **C** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha*op(A)*op(B) + beta*C)

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zgemm](#)
[PLASMA_zgemm_Tile_Async](#)
[PLASMA_cgemm_Tile](#)
[PLASMA_dgemm_Tile](#)
[PLASMA_sgemm_Tile](#)

3.5.2.9 `int PLASMA_zgeqrf_Tile (PLASMA_desc * A, PLASMA_desc * T)`

`PLASMA_zgeqrf_Tile` - Computes the tile QR factorization of a matrix. Tile equivalent of [PLASMA_zgeqrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal represent the unitary matrix Q as a product of elementary reflectors stored by tiles.
- *T* On exit, auxiliary factorization data, required by `PLASMA_zgeqrs` to solve the system of equations.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zgeqrf](#)
[PLASMA_zgeqrf_Tile_Async](#)
[PLASMA_cgeqrf_Tile](#)
[PLASMA_dgeqrf_Tile](#)
[PLASMA_sgeqrf_Tile](#)
[PLASMA_zgeqrs_Tile](#)

3.5.2.10 `int PLASMA_zgeqrs_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_zgeqrs_Tile` - Computes a minimum-norm solution using the tile QR factorization. Tile equivalent of [PLASMA_zgetrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ *A* Details of the QR factorization of the original matrix A as returned by `PLASMA_zgeqrf`.
- ← *T* Auxiliary factorization data, computed by `PLASMA_zgeqrf`.
- ↔ *B* On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zgeqrs](#)

[PLASMA_zgeqrs_Tile_Async](#)
[PLASMA_cgeqrs_Tile](#)
[PLASMA_dgeqrs_Tile](#)
[PLASMA_sgeqrs_Tile](#)
[PLASMA_zgeqrf_Tile](#)

3.5.2.11 int PLASMA_zgesv_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)

PLASMA_zgesv_Tile - Solves a system of linear equations using the tile LU factorization. Tile equivalent of [PLASMA_zgetrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).
- ↔ **L** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.
- **IPIV** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

- PLASMA_SUCCESS** successful exit
- >0 if $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_zgesv](#)
[PLASMA_zgesv_Tile_Async](#)
[PLASMA_cgesv_Tile](#)
[PLASMA_dgesv_Tile](#)
[PLASMA_sgesv_Tile](#)
[PLASMA_zcgesv_Tile](#)

3.5.2.12 int PLASMA_zgetrf_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV)

PLASMA_zgetrf_Tile - Computes the tile LU factorization of a matrix. Tile equivalent of [PLASMA_zgetrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

→ *L* On exit, auxiliary factorization data, related to the tile L factor, required by `PLASMA_zgetrs` to solve the system of equations.

→ *IPIV* The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if *i*, $U(i,i)$ is exactly zero. The factorization has been completed, but the factor *U* is exactly singular, and division by zero will occur if it is used to solve a system of equations.

See also

[PLASMA_zgetrf](#)
[PLASMA_zgetrf_Tile_Async](#)
[PLASMA_cgetrf_Tile](#)
[PLASMA_dgetrf_Tile](#)
[PLASMA_sgetrf_Tile](#)
[PLASMA_zgetrs_Tile](#)

3.5.2.13 int PLASMA_zgetrs_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)

`PLASMA_zgetrs_Tile` - Solves a system of linear equations using previously computed LU factorization. Tile equivalent of `PLASMA_zgetrs()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* The tile factors *L* and *U* from the factorization, computed by `PLASMA_zgetrf`.

← *L* Auxiliary factorization data, related to the tile *L* factor, computed by `PLASMA_zgetrf`.

← *IPIV* The pivot indices from `PLASMA_zgetrf` (not equivalent to LAPACK).

↔ *B* On entry, the *N*-by-*NRHS* matrix of right hand side matrix *B*. On exit, the solution matrix *X*.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zgetrs](#)
[PLASMA_zgetrs_Tile_Async](#)
[PLASMA_cgetrs_Tile](#)
[PLASMA_dgetrs_Tile](#)
[PLASMA_sgetrs_Tile](#)
[PLASMA_zgetrf_Tile](#)

3.5.2.14 `int PLASMA_zhemm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex64_t beta, PLASMA_desc * C)`

`PLASMA_zhemm_Tile` - Performs Hermitian matrix multiplication. Tile equivalent of `PLASMA_zhemm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *side* Specifies whether the hermitian matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← *uplo* Specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the hermitian matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the hermitian matrix A is to be referenced.

← *alpha* Specifies the scalar alpha.

← *A* A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← *B* B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← *beta* Specifies the scalar beta.

↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zhemm](#)
[PLASMA_zhemm_Tile_Async](#)
[PLASMA_chemm_Tile](#)
[PLASMA_dhemm_Tile](#)
[PLASMA_shemm_Tile](#)

3.5.2.15 `int PLASMA_zher2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, double beta, PLASMA_desc * C)`

`PLASMA_zher2k_Tile` - Performs hermitian rank k update. Tile equivalent of `PLASMA_zher2k()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zher2k_Tile](#)
[PLASMA_cher2k](#)
[PLASMA_dher2k](#)
[PLASMA_sher2k](#)

3.5.2.16 int PLASMA_zherk_Tile (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc * A, double beta, PLASMA_desc * C)

PLASMA_zherk_Tile - Performs hermitian rank k update. Tile equivalent of [PLASMA_zherk\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zherk_Tile](#)
[PLASMA_cherk](#)
[PLASMA_dherk](#)
[PLASMA_sherk](#)

3.5.2.17 double PLASMA_zlange_Tile (PLASMA_enum *norm*, PLASMA_desc * *A*, double * *work*)

PLASMA_zlange_Tile - Tile equivalent of [PLASMA_zlange\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zlange](#)
[PLASMA_zlange_Tile_Async](#)
[PLASMA_clange_Tile](#)
[PLASMA_dlange_Tile](#)
[PLASMA_slange_Tile](#)

3.5.2.18 double PLASMA_zlanhe_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc * *A*, double * *work*)

PLASMA_zlanhe_Tile - Tile equivalent of [PLASMA_zlanhe\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zlanhe](#)
[PLASMA_zlanhe_Tile_Async](#)
[PLASMA_clanhe_Tile](#)
[PLASMA_dlanhe_Tile](#)
[PLASMA_slanhe_Tile](#)

3.5.2.19 double PLASMA_zlansy_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc * *A*, double * *work*)

PLASMA_zlansy_Tile - Tile equivalent of [PLASMA_zlansy\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zlansy](#)
[PLASMA_zlansy_Tile_Async](#)
[PLASMA_clansy_Tile](#)
[PLASMA_dlansy_Tile](#)
[PLASMA_slansy_Tile](#)

3.5.2.20 int PLASMA_zlauum_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*)

PLASMA_zlauum_Tile - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Tile equivalent of [PLASMA_zlauum\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zlauum](#)
[PLASMA_zlauum_Tile_Async](#)
[PLASMA_clauum_Tile](#)
[PLASMA_dlauum_Tile](#)
[PLASMA_slauum_Tile](#)
[PLASMA_zpotri_Tile](#)

3.5.2.21 int PLASMA_zplghe_Tile (double *bump*, PLASMA_desc * *A*, unsigned long long int *seed*)

`PLASMA_zplghe_Tile` - Generate a random hermitian matrix by tiles. Tile equivalent of [PLASMA_zplghe\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* On exit, The random hermitian matrix *A* generated.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zplghe](#)
[PLASMA_zplghe_Tile_Async](#)
[PLASMA_cplghe_Tile](#)
[PLASMA_dplghe_Tile](#)
[PLASMA_splghe_Tile](#)
[PLASMA_zplrnt_Tile](#)
[PLASMA_zplgsy_Tile](#)

3.5.2.22 int PLASMA_zplgsy_Tile (PLASMA_Complex64_t *bump*, PLASMA_desc * *A*, unsigned long long int *seed*)

`PLASMA_zplgsy_Tile` - Generate a random hermitian matrix by tiles. Tile equivalent of [PLASMA_zplgsy\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* On exit, The random hermitian matrix *A* generated.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zplgsy](#)
[PLASMA_zplgsy_Tile_Async](#)
[PLASMA_cplgsy_Tile](#)
[PLASMA_dplgsy_Tile](#)
[PLASMA_splgsy_Tile](#)
[PLASMA_zplrnt_Tile](#)
[PLASMA_zplgsy_Tile](#)

3.5.2.23 int PLASMA_zplrnt_Tile (PLASMA_desc * A, unsigned long long int seed)

PLASMA_zplrnt_Tile - Generate a random matrix by tiles. Tile equivalent of [PLASMA_zplrnt\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* On exit, The random matrix A generated.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zplrnt](#)
[PLASMA_zplrnt_Tile_Async](#)
[PLASMA_cplrnt_Tile](#)
[PLASMA_dplrnt_Tile](#)
[PLASMA_splrnt_Tile](#)
[PLASMA_zplghe_Tile](#)
[PLASMA_zplgsy_Tile](#)

3.5.2.24 int PLASMA_zposv_Tile (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B)

PLASMA_zposv_Tile - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Tile equivalent of [PLASMA_zposv\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

- ↔ **A** On entry, the symmetric positive definite (or Hermitian) matrix A. If `uplo = PlasmaUpper`, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If `UPLO = 'L'`, the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U**H*U$ or $A = L*L**H$.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_zposv](#)
[PLASMA_zposv_Tile_Async](#)
[PLASMA_cposv_Tile](#)
[PLASMA_dposv_Tile](#)
[PLASMA_sposv_Tile](#)

3.5.2.25 int PLASMA_zpotrf_Tile (PLASMA_enum uplo, PLASMA_desc * A)

`PLASMA_zpotrf_Tile` - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Tile equivalent of `PLASMA_zpotrf()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **uplo** = `PlasmaUpper`: Upper triangle of A is stored; = `PlasmaLower`: Lower triangle of A is stored.
- ↔ **A** On entry, the symmetric positive definite (or Hermitian) matrix A. If `uplo = PlasmaUpper`, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If `UPLO = 'L'`, the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U**H*U$ or $A = L*L**H$.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_zpotrf](#)
[PLASMA_zpotrf_Tile_Async](#)
[PLASMA_cpotrf_Tile](#)
[PLASMA_dpotrf_Tile](#)
[PLASMA_spotrf_Tile](#)
[PLASMA_zpotrs_Tile](#)

3.5.2.26 int PLASMA_zpotri_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*)

PLASMA_zpotri_Tile - Computes the inverse of a complex Hermitian positive definite matrix *A* using the Cholesky factorization $A = U^{*}H^{*}U$ or $A = L^{*}L^{*}H^{*}$ computed by PLASMA_zpotrf. Tile equivalent of [PLASMA_zpotri\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.
- ← *A* On entry, the triangular factor *U* or *L* from the Cholesky factorization $A = U^{*}H^{*}U$ or $A = L^{*}L^{*}H^{*}$, as computed by PLASMA_zpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of *A*, overwriting the input factor *U* or *L*.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- >0 if *i*, the leading minor of order *i* of *A* is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_zpotri](#)
[PLASMA_zpotri_Tile_Async](#)
[PLASMA_cpotri_Tile](#)
[PLASMA_dpotri_Tile](#)
[PLASMA_spotri_Tile](#)
[PLASMA_zpotrf_Tile](#)

3.5.2.27 int PLASMA_zpotrs_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*, PLASMA_desc * *B*)

PLASMA_zpotrs_Tile - Solves a system of linear equations using previously computed Cholesky factorization. Tile equivalent of [PLASMA_zpotrs\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.

- ← **A** The triangular factor U or L from the Cholesky factorization $A = U^{**}H^{*}U$ or $A = L^{*}L^{**}H$, computed by `PLASMA_zpotrf`.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zpotrs](#)
[PLASMA_zpotrs_Tile_Async](#)
[PLASMA_cpotrs_Tile](#)
[PLASMA_dpotrs_Tile](#)
[PLASMA_spotrs_Tile](#)
[PLASMA_zpotrf_Tile](#)

3.5.2.28 `int PLASMA_zsymm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex64_t beta, PLASMA_desc * C)`

`PLASMA_zsymm_Tile` - Performs symmetric matrix multiplication. Tile equivalent of `PLASMA_zsymm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **side** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

- ← **uplo** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.
- ← **alpha** Specifies the scalar alpha.
- ← **A** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.
- ← **B** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.
- ← **beta** Specifies the scalar beta.
- ↔ **C** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zsymm](#)
[PLASMA_zsymm_Tile_Async](#)
[PLASMA_csymm_Tile](#)
[PLASMA_dsymm_Tile](#)
[PLASMA_ssymm_Tile](#)

3.5.2.29 `int PLASMA_zsyr2k_Tile (PLASMA_enum uplo, PLASMA_enum trans,
 PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B,
 PLASMA_Complex64_t beta, PLASMA_desc * C)`

PLASMA_zsyr2k_Tile - Performs symmetric rank k update. Tile equivalent of [PLASMA_zsyr2k\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is conjugate transposed.
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zsyr2k_Tile](#)
[PLASMA_csyr2k](#)
[PLASMA_dsyr2k](#)
[PLASMA_ssyr2k](#)

3.5.2.30 `int PLASMA_zsyrk_Tile (PLASMA_enum uplo, PLASMA_enum trans,
 PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_Complex64_t beta,
 PLASMA_desc * C)`

PLASMA_zsyrk_Tile - Performs rank k update. Tile equivalent of [PLASMA_zsyrk\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed.
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zsyrc_Tile](#)
[PLASMA_csyrc](#)
[PLASMA_dsyrc](#)
[PLASMA_ssyrc](#)

3.5.2.31 int PLASMA_ztrmm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B)

PLASMA_ztrmm_Tile - Computes triangular solve. Tile equivalent of [PLASMA_ztrmm\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *transA* Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← *diag* Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← *A* The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array a contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_ztrmm](#)
[PLASMA_ztrmm_Tile_Async](#)
[PLASMA_ctrmm_Tile](#)
[PLASMA_dtrmm_Tile](#)
[PLASMA_strmm_Tile](#)

3.5.2.32 int PLASMA_ztrsm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex64_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*)

PLASMA_ztrsm_Tile - Computes triangular solve. Tile equivalent of [PLASMA_ztrsm\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ↔ ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ↔ ***transA*** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ↔ ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ↔ ***A*** The triangular matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If *uplo* = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If *diag* = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_ztrsm](#)
[PLASMA_ztrsm_Tile_Async](#)
[PLASMA_ctrsm_Tile](#)
[PLASMA_dtrsm_Tile](#)
[PLASMA_strsm_Tile](#)

3.5.2.33 int PLASMA_ztrsmp1_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)

PLASMA_ztrsmp1_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **A** The tile factor L from the factorization, computed by PLASMA_zgetrf.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_zgetrf.
- ← **IPIV** The pivot indices from PLASMA_zgetrf (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_ztrsmp1](#)
[PLASMA_ztrsmp1_Tile_Async](#)
[PLASMA_ctrsmpl_Tile](#)
[PLASMA_dtrsmp1_Tile](#)
[PLASMA_strsmpl_Tile](#)
[PLASMA_zgetrf_Tile](#)

3.5.2.34 int PLASMA_ztrtri_Tile (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc * A)

PLASMA_ztrtri_Tile - Computes the inverse of a complex upper or lower triangular matrix A. Tile equivalent of [PLASMA_ztrtri\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **uplo** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **diag** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.

← *A* On entry, the triangular matrix *A*. If *UPLO* = 'U', the leading *N*-by-*N* upper triangular part of the array *A* contains the upper triangular matrix, and the strictly lower triangular part of *A* is not referenced. If *UPLO* = 'L', the leading *N*-by-*N* lower triangular part of the array *A* contains the lower triangular matrix, and the strictly upper triangular part of *A* is not referenced. If *DIAG* = 'U', the diagonal elements of *A* are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if *i*, *A*(*i*,*i*) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

[PLASMA_ztrtri](#)
[PLASMA_ztrtri_Tile_Async](#)
[PLASMA_ctrtri_Tile](#)
[PLASMA_dtrtri_Tile](#)
[PLASMA_strtri_Tile](#)
[PLASMA_zpotri_Tile](#)

3.5.2.35 int PLASMA_zunglq_Tile (PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_zunglq_Tile - Generates an *M*-by-*N* matrix *Q* with orthonormal rows, which is defined as the first *M* rows of a product of the elementary reflectors returned by *PLASMA_zgelqf*. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* Details of the LQ factorization of the original matrix *A* as returned by *PLASMA_zgelqf*.

← *T* Auxiliary factorization data, computed by *PLASMA_zgelqf*.

→ *B* On exit, the *M*-by-*N* matrix *Q*.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_zunglq](#)
[PLASMA_zunglq_Tile_Async](#)
[PLASMA_cunglq_Tile](#)
[PLASMA_dunglq_Tile](#)
[PLASMA_sunglq_Tile](#)
[PLASMA_zgelqf_Tile](#)

3.5.2.36 `int PLASMA_zungqr_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * Q)`

`PLASMA_zungqr_Tile` - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by `PLASMA_zgeqrf`. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* Details of the QR factorization of the original matrix A as returned by `PLASMA_zgeqrf`.
- ← *T* Auxiliary factorization data, computed by `PLASMA_zgeqrf`.
- *Q* On exit, the M-by-N matrix Q.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zungqr](#)
[PLASMA_zungqr_Tile_Async](#)
[PLASMA_cungqr_Tile](#)
[PLASMA_dungqr_Tile](#)
[PLASMA_sungqr_Tile](#)
[PLASMA_zgeqrf_Tile](#)

3.5.2.37 `int PLASMA_zunmlq_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_zunmlq_Tile` - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by `PLASMA_zgelqf_Tile` Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Intended usage: = `PlasmaLeft`: apply Q or Q**H from the left; = `PlasmaRight`: apply Q or Q**H from the right. Currently only `PlasmaLeft` is supported.
- ← *trans* Intended usage: = `PlasmaNoTrans`: no transpose, apply Q; = `PlasmaConjTrans`: conjugate transpose, apply Q**H. Currently only `PlasmaConjTrans` is supported.
- ← *A* Details of the LQ factorization of the original matrix A as returned by `PLASMA_zgelqf`.
- ← *T* Auxiliary factorization data, computed by `PLASMA_zgelqf`.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by Q*B or Q**H*B.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_zunmlq](#)
[PLASMA_zunmlq_Tile_Async](#)
[PLASMA_cunmlq_Tile](#)
[PLASMA_dunmlq_Tile](#)
[PLASMA_sunmlq_Tile](#)
[PLASMA_zgelqf_Tile](#)

3.5.2.38 `int PLASMA_zunmqr_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

PLASMA_zunmqr_Tile - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_zgeqrf_Tile. Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q**H from the left; = PlasmaRight: apply Q or Q**H from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjugate transpose, apply Q**H. Currently only PlasmaConjTrans is supported.
- ← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_zgeqrf.
- ← *T* Auxiliary factorization data, computed by PLASMA_zgeqrf.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by Q*B or Q**H*B.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_zunmqr](#)
[PLASMA_zunmqr_Tile_Async](#)
[PLASMA_cunmqr_Tile](#)
[PLASMA_dunmqr_Tile](#)
[PLASMA_sunmqr_Tile](#)
[PLASMA_zgeqrf_Tile](#)

3.6 Advanced Interface: Synchronous - Single Complex

Functions/Subroutines

- int [PLASMA_cgelqf_Tile](#) (PLASMA_desc *A, PLASMA_desc *T)
- int [PLASMA_cgelqs_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_cgels_Tile](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_cgemm_Tile](#) (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int [PLASMA_cgeqrf_Tile](#) (PLASMA_desc *A, PLASMA_desc *T)
- int [PLASMA_cgeqrs_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_cgesv_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int [PLASMA_cgetrf_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV)
- int [PLASMA_cgetrs_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int [PLASMA_chemm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int [PLASMA_cher2k_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C)
- int [PLASMA_cherk_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc *A, float beta, PLASMA_desc *C)
- float [PLASMA_clange_Tile](#) (PLASMA_enum norm, PLASMA_desc *A, float *work)
- float [PLASMA_clanhe_Tile](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work)
- float [PLASMA_clansy_Tile](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work)
- int [PLASMA_clauum_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_cplghe_Tile](#) (float bump, PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_cplgsy_Tile](#) (PLASMA_Complex32_t bump, PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_cplrnt_Tile](#) (PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_cposv_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_cpotrf_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_cpotri_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_cpotrs_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_csymm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int [PLASMA_csyr2k_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int [PLASMA_csyrk_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int [PLASMA_ctrmm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_ctrsm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_ctrsmpl_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int [PLASMA_ctrtri_Tile](#) (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A)
- int [PLASMA_cunglq_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_cungqr_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q)

- int [PLASMA_cunmlq_Tile](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_cunmqr_Tile](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)

3.6.1 Detailed Description

This is the group of single complex functions using the advanced synchronous interface.

3.6.2 Function/Subroutine Documentation

3.6.2.1 int PLASMA_cgelqf_Tile (PLASMA_desc * A, PLASMA_desc * T)

[PLASMA_cgelqf_Tile](#) - Computes the tile LQ factorization of a matrix. Tile equivalent of [PLASMA_cgelqf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if M ≤ N); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.
- *T* On exit, auxiliary factorization data, required by [PLASMA_cgelqs](#) to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cgelqf](#)
[PLASMA_cgelqf_Tile_Async](#)
[PLASMA_cgelqf_Tile](#)
[PLASMA_dgelqf_Tile](#)
[PLASMA_sgelqf_Tile](#)
[PLASMA_cgelqs_Tile](#)

3.6.2.2 int PLASMA_cgelqs_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)

[PLASMA_cgelqs_Tile](#) - Computes a minimum-norm solution using previously computed LQ factorization. Tile equivalent of [PLASMA_cgelqs\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* Details of the LQ factorization of the original matrix A as returned by [PLASMA_cgelqf](#).

- ← *T* Auxiliary factorization data, computed by PLASMA_cgelqf.
- ↔ *B* On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cgelqs](#)
[PLASMA_cgelqs_Tile_Async](#)
[PLASMA_cgelqs_Tile](#)
[PLASMA_dgelqs_Tile](#)
[PLASMA_sgelqs_Tile](#)
[PLASMA_cgelqf_Tile](#)

3.6.2.3 int PLASMA_cgels_Tile (PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_cgels_Tile - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Tile equivalent of [PLASMA_cgels\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *trans* Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A**H. Currently only PlasmaNoTrans is supported.
- ↔ *A* On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_cgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_cgelqf.
- *T* On exit, auxiliary factorization data.
- ↔ *B* On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns

PLASMA_SUCCESS successful exit

See also

[PLASMA_cgels](#)
[PLASMA_cgels_Tile_Async](#)
[PLASMA_cgels_Tile](#)
[PLASMA_dgels_Tile](#)
[PLASMA_sgels_Tile](#)

3.6.2.4 `int PLASMA_cgemm_Tile (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex32_t beta, PLASMA_desc * C)`

`PLASMA_cgemm_Tile` - Performs matrix multiplication. Tile equivalent of `PLASMA_cgemm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *transA* Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← *transB* Specifies whether the matrix B is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaConjTrans: B is conjugate transposed.
- ← *alpha* alpha specifies the scalar alpha
- ← *A* A is a LDA-by-ka matrix, where ka is K when *transA* = PlasmaNoTrans, and is M otherwise.
- ← *B* B is a LDB-by-kb matrix, where kb is N when *transB* = PlasmaNoTrans, and is K otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha*op(A)*op(B) + beta*C)

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_cgemm](#)
[PLASMA_cgemm_Tile_Async](#)
[PLASMA_cgemm_Tile](#)
[PLASMA_dgemm_Tile](#)
[PLASMA_sgemm_Tile](#)

3.6.2.5 `int PLASMA_cgeqrf_Tile (PLASMA_desc * A, PLASMA_desc * T)`

`PLASMA_cgeqrf_Tile` - Computes the tile QR factorization of a matrix. Tile equivalent of `PLASMA_cgeqrf()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal represent the unitary matrix Q as a product of elementary reflectors stored by tiles.
- *T* On exit, auxiliary factorization data, required by `PLASMA_cgeqrs` to solve the system of equations.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_cgeqrf](#)
[PLASMA_cgeqrf_Tile_Async](#)
[PLASMA_cgeqrf_Tile](#)
[PLASMA_dgeqrf_Tile](#)
[PLASMA_sgeqrf_Tile](#)
[PLASMA_cgeqrs_Tile](#)

3.6.2.6 int PLASMA_cgeqrs_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)

PLASMA_cgeqrs_Tile - Computes a minimum-norm solution using the tile QR factorization. Tile equivalent of [PLASMA_cgetrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ *A* Details of the QR factorization of the original matrix A as returned by *PLASMA_cgeqrf*.
- ← *T* Auxiliary factorization data, computed by *PLASMA_cgeqrf*.
- ↔ *B* On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_cgeqrs](#)
[PLASMA_cgeqrs_Tile_Async](#)
[PLASMA_cgeqrs_Tile](#)
[PLASMA_dgeqrs_Tile](#)
[PLASMA_sgeqrs_Tile](#)
[PLASMA_cgeqrf_Tile](#)

3.6.2.7 int PLASMA_cgesv_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)

PLASMA_cgesv_Tile - Solves a system of linear equations using the tile LU factorization. Tile equivalent of [PLASMA_cgetrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).
- ↔ **L** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.
- **IPIV** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

- PLASMA_SUCCESS** successful exit
- >0 if $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_cgesv](#)
[PLASMA_cgesv_Tile_Async](#)
[PLASMA_cgesv_Tile](#)
[PLASMA_dgesv_Tile](#)
[PLASMA_sgesv_Tile](#)
[PLASMA_ccgesv_Tile](#)

3.6.2.8 int PLASMA_cgetrf_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV)

PLASMA_cgetrf_Tile - Computes the tile LU factorization of a matrix. Tile equivalent of [PLASMA_cgetrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.
- **L** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_cgetrs to solve the system of equations.
- **IPIV** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns**Return values**

- PLASMA_SUCCESS** successful exit
- >0 if $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

See also

[PLASMA_cgetrf](#)
[PLASMA_cgetrf_Tile_Async](#)
[PLASMA_cgetrf_Tile](#)
[PLASMA_dgetrf_Tile](#)
[PLASMA_sgetrf_Tile](#)
[PLASMA_cgetrs_Tile](#)

3.6.2.9 int PLASMA_cgetrs_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)

PLASMA_cgetrs_Tile - Solves a system of linear equations using previously computed LU factorization. Tile equivalent of [PLASMA_cgetrs\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* The tile factors L and U from the factorization, computed by PLASMA_cgetrf.
- ← *L* Auxiliary factorization data, related to the tile L factor, computed by PLASMA_cgetrf.
- ← *IPIV* The pivot indices from PLASMA_cgetrf (not equivalent to LAPACK).
- ↔ *B* On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_cgetrs](#)
[PLASMA_cgetrs_Tile_Async](#)
[PLASMA_cgetrs_Tile](#)
[PLASMA_dgetrs_Tile](#)
[PLASMA_sgetrs_Tile](#)
[PLASMA_cgetrf_Tile](#)

3.6.2.10 int PLASMA_chemm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex32_t beta, PLASMA_desc * C)

PLASMA_chemm_Tile - Performs Hermitian matrix multiplication. Tile equivalent of [PLASMA_chemm\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Specifies whether the hermitian matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

- = PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

- ← ***uplo*** Specifies whether the upper or lower triangular part of the hermitian matrix *A* is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the hermitian matrix *A* is to be referenced. = PlasmaUpper: Only the upper triangular part of the hermitian matrix *A* is to be referenced.
- ← ***alpha*** Specifies the scalar alpha.
- ← ***A*** *A* is a LDA-by-ka matrix, where ka is *M* when side = PlasmaLeft, and is *N* otherwise. Only the uplo triangular part is referenced.
- ← ***B*** *B* is a LDB-by-*N* matrix, where the leading *M*-by-*N* part of the array *B* must contain the matrix *B*.
- ← ***beta*** Specifies the scalar beta.
- ↔ ***C*** *C* is a LDC-by-*N* matrix. On exit, the array is overwritten by the *M* by *N* updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_chemm](#)
[PLASMA_chemm_Tile_Async](#)
[PLASMA_chemm_Tile](#)
[PLASMA_dhemm_Tile](#)
[PLASMA_shemm_Tile](#)

3.6.2.11 int PLASMA_cher2k_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex32_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, float *beta*, PLASMA_desc * *C*)

PLASMA_cher2k_Tile - Performs hermitian rank *k* update. Tile equivalent of [PLASMA_cher2k\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← ***uplo*** = PlasmaUpper: Upper triangle of *C* is stored; = PlasmaLower: Lower triangle of *C* is stored.
- ← ***trans*** Specifies whether the matrix *A* is transposed or conjugate transposed: = PlasmaNoTrans: *A* is not transposed; = PlasmaConjTrans: *A* is conjugate transposed.
- ← ***alpha*** *alpha* specifies the scalar alpha.
- ← ***A*** *A* is a LDA-by-ka matrix, where ka is *K* when trans = PlasmaNoTrans, and is *N* otherwise.
- ← ***beta*** *beta* specifies the scalar beta
- ↔ ***C*** *C* is a LDC-by-*N* matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cher2k_Tile](#)
[PLASMA_cher2k](#)
[PLASMA_dher2k](#)
[PLASMA_sher2k](#)

3.6.2.12 int PLASMA_cherk_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, float *alpha*, PLASMA_desc * *A*, float *beta*, PLASMA_desc * *C*)

PLASMA_cherk_Tile - Performs hermitian rank k update. Tile equivalent of [PLASMA_cherk\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_cherk_Tile](#)
[PLASMA_cherk](#)
[PLASMA_dherk](#)
[PLASMA_sherk](#)

3.6.2.13 float PLASMA_clange_Tile (PLASMA_enum *norm*, PLASMA_desc * *A*, float * *work*)

PLASMA_clange_Tile - Tile equivalent of [PLASMA_clange\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of *A* is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of *A* is overwritten with the lower triangle of the product $L' * L$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_clange](#)
[PLASMA_clange_Tile_Async](#)
[PLASMA_clange_Tile](#)
[PLASMA_dlange_Tile](#)
[PLASMA_slange_Tile](#)

3.6.2.14 float PLASMA_clanhe_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc * *A*, float * *work*)

PLASMA_clanhe_Tile - Tile equivalent of [PLASMA_clanhe\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of *A* is stored; = PlasmaLower: Lower triangle of *A* is stored.
 ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of *A* is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of *A* is overwritten with the lower triangle of the product $L' * L$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_clanhe](#)
[PLASMA_clanhe_Tile_Async](#)
[PLASMA_clanhe_Tile](#)
[PLASMA_dlanhe_Tile](#)
[PLASMA_slanhe_Tile](#)

3.6.2.15 float PLASMA_clansy_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc * *A*, float * *work*)

PLASMA_clansy_Tile - Tile equivalent of [PLASMA_clansy\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_clansy](#)
[PLASMA_clansy_Tile_Async](#)
[PLASMA_clansy_Tile](#)
[PLASMA_dlansy_Tile](#)
[PLASMA_slansy_Tile](#)

3.6.2.16 int PLASMA_clauum_Tile (PLASMA_enum uplo, PLASMA_desc * A)

PLASMA_clauum_Tile - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Tile equivalent of [PLASMA_clauum\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_clauum](#)
[PLASMA_clauum_Tile_Async](#)
[PLASMA_clauum_Tile](#)
[PLASMA_dlauum_Tile](#)
[PLASMA_slauum_Tile](#)
[PLASMA_cpotri_Tile](#)

3.6.2.17 `int PLASMA_cplghe_Tile (float bump, PLASMA_desc * A, unsigned long long int seed)`

PLASMA_cplghe_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of [PLASMA_cplghe\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* On exit, The random hermitian matrix *A* generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cplghe](#)
[PLASMA_cplghe_Tile_Async](#)
[PLASMA_cplghe_Tile](#)
[PLASMA_dplghe_Tile](#)
[PLASMA_splghe_Tile](#)
[PLASMA_cplrint_Tile](#)
[PLASMA_cplgsy_Tile](#)

3.6.2.18 `int PLASMA_cplgsy_Tile (PLASMA_Complex32_t bump, PLASMA_desc * A, unsigned long long int seed)`

PLASMA_cplgsy_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of [PLASMA_cplgsy\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* On exit, The random hermitian matrix *A* generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cplgsy](#)
[PLASMA_cplgsy_Tile_Async](#)
[PLASMA_cplgsy_Tile](#)
[PLASMA_dplgsy_Tile](#)
[PLASMA_splgsy_Tile](#)
[PLASMA_cplrint_Tile](#)
[PLASMA_cplgsy_Tile](#)

3.6.2.19 `int PLASMA_cplrnt_Tile (PLASMA_desc * A, unsigned long long int seed)`

`PLASMA_cplrnt_Tile` - Generate a random matrix by tiles. Tile equivalent of `PLASMA_cplrnt()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← **A** On exit, The random matrix A generated.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_cplrnt](#)
[PLASMA_cplrnt_Tile_Async](#)
[PLASMA_cplrnt_Tile](#)
[PLASMA_dplrnt_Tile](#)
[PLASMA_splrnt_Tile](#)
[PLASMA_cplghe_Tile](#)
[PLASMA_cplgsy_Tile](#)

3.6.2.20 `int PLASMA_cposv_Tile (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B)`

`PLASMA_cposv_Tile` - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Tile equivalent of `PLASMA_cposv()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

↔ **A** On entry, the symmetric positive definite (or Hermitian) matrix A. If `uplo = PlasmaUpper`, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If `UPLO = 'L'`, the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U^* H U$ or $A = L L^* H$.

↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

>0 if i , the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_cposv](#)
[PLASMA_cposv_Tile_Async](#)
[PLASMA_cposv_Tile](#)
[PLASMA_dposv_Tile](#)
[PLASMA_sposv_Tile](#)

3.6.2.21 int PLASMA_cpotrf_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*)

PLASMA_cpotrf_Tile - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Tile equivalent of [PLASMA_cpotrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
 ← A On entry, the symmetric positive definite (or Hermitian) matrix A . If *uplo* = PlasmaUpper, the leading N -by- N upper triangular part of A contains the upper triangular part of the matrix A , and the strictly lower triangular part of A is not referenced. If *UPLO* = 'L', the leading N -by- N lower triangular part of A contains the lower triangular part of the matrix A , and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U \backslash * \backslash * H * U$ or $A = L * L \backslash * \backslash * H$.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i , the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_cpotrf](#)
[PLASMA_cpotrf_Tile_Async](#)
[PLASMA_cpotrf_Tile](#)
[PLASMA_dpotrf_Tile](#)
[PLASMA_spotrf_Tile](#)
[PLASMA_cpotrs_Tile](#)

3.6.2.22 int PLASMA_cpotri_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*)

PLASMA_cpotri_Tile - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U \backslash * \backslash * H * U$ or $A = L * L \backslash * \backslash * H$ computed by [PLASMA_cpotrf](#). Tile equivalent of [PLASMA_cpotri\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← *A* On entry, the triangular factor U or L from the Cholesky factorization $A = U^*H^*U$ or $A = L^*L^*H$, as computed by PLASMA_cpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

Returns**Return values**

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_cpotri](#)
[PLASMA_cpotri_Tile_Async](#)
[PLASMA_cpotri_Tile](#)
[PLASMA_dpotri_Tile](#)
[PLASMA_spotri_Tile](#)
[PLASMA_cpotrf_Tile](#)

3.6.2.23 int PLASMA_cpotsr_Tile (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B)

PLASMA_cpotsr_Tile - Solves a system of linear equations using previously computed Cholesky factorization. Tile equivalent of [PLASMA_cpotsr\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← *A* The triangular factor U or L from the Cholesky factorization $A = U^*H^*U$ or $A = L^*L^*H$, computed by PLASMA_cpotrf.

↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_cpotsr](#)
[PLASMA_cpotsr_Tile_Async](#)
[PLASMA_cpotsr_Tile](#)
[PLASMA_dpotsr_Tile](#)
[PLASMA_spotrs_Tile](#)
[PLASMA_cpotrf_Tile](#)

3.6.2.24 `int PLASMA_csymm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex32_t beta, PLASMA_desc * C)`

`PLASMA_csymm_Tile` - Performs symmetric matrix multiplication. Tile equivalent of `PLASMA_csymm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *side* Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← *uplo* Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← *alpha* Specifies the scalar alpha.

← *A* A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← *B* B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← *beta* Specifies the scalar beta.

↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_csymm](#)
[PLASMA_csymm_Tile_Async](#)
[PLASMA_csymm_Tile](#)
[PLASMA_dsymm_Tile](#)
[PLASMA_ssymm_Tile](#)

3.6.2.25 `int PLASMA_csyr2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex32_t beta, PLASMA_desc * C)`

`PLASMA_csyr2k_Tile` - Performs symmetric rank k update. Tile equivalent of `PLASMA_csyr2k()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is conjfugate transposed.
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_csyr2k_Tile](#)
[PLASMA_csyr2k](#)
[PLASMA_dsyr2k](#)
[PLASMA_ssyr2k](#)

3.6.2.26 `int PLASMA_csyrk_Tile (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_Complex32_t beta, PLASMA_desc * C)`

PLASMA_csyrk_Tile - Performs rank k update. Tile equivalent of [PLASMA_csyrk\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed.
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_csyk_Tile](#)
[PLASMA_csyk](#)
[PLASMA_dsyk](#)
[PLASMA_ssyk](#)

3.6.2.27 int PLASMA_ctrmm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex32_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*)

PLASMA_ctrmm_Tile - Computes triangular solve. Tile equivalent of [PLASMA_ctrmm\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *transA* Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← *diag* Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← *A* The triangular matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If *uplo* = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If *diag* = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_ctrmm](#)
[PLASMA_ctrmm_Tile_Async](#)
[PLASMA_ctrmm_Tile](#)
[PLASMA_dtrmm_Tile](#)
[PLASMA_strmm_Tile](#)

3.6.2.28 `int PLASMA_ctrsm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B)`

`PLASMA_ctrsm_Tile` - Computes triangular solve. Tile equivalent of `PLASMA_ctrsm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *transA* Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.
- ← *diag* Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A is unit.
- ← *A* The triangular matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If *uplo* = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If *diag* = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_ctrsm](#)
[PLASMA_ctrsm_Tile_Async](#)
[PLASMA_ctrsm_Tile](#)
[PLASMA_dtrsm_Tile](#)
[PLASMA_strsm_Tile](#)

3.6.2.29 `int PLASMA_ctrsmpl_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)`

`PLASMA_ctrsmpl_Tile` - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* The tile factor L from the factorization, computed by `PLASMA_cgetrf`.

- ← *L* Auxiliary factorization data, related to the tile L factor, computed by PLASMA_cgetrf.
- ← *IPIV* The pivot indices from PLASMA_cgetrf (not equivalent to LAPACK).
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_ctrsmpl](#)
[PLASMA_ctrsmpl_Tile_Async](#)
[PLASMA_ctrsmpl_Tile](#)
[PLASMA_dtrsmpl_Tile](#)
[PLASMA_strsmpl_Tile](#)
[PLASMA_cgetrf_Tile](#)

3.6.2.30 int PLASMA_ctrtri_Tile (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc * A)

PLASMA_ctrtri_Tile - Computes the inverse of a complex upper or lower triangular matrix A. Tile equivalent of [PLASMA_ctrtri\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *diag* = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.
- ← *A* On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

[PLASMA_ctrtri](#)
[PLASMA_ctrtri_Tile_Async](#)
[PLASMA_ctrtri_Tile](#)
[PLASMA_dtrtri_Tile](#)
[PLASMA_strtri_Tile](#)
[PLASMA_cpotri_Tile](#)

3.6.2.31 int PLASMA_cunglq_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)

PLASMA_cunglq_Tile - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_cgelqf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* Details of the LQ factorization of the original matrix A as returned by PLASMA_cgelqf.
- ← *T* Auxiliary factorization data, computed by PLASMA_cgelqf.
- *B* On exit, the M-by-N matrix Q.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_cunglq](#)
[PLASMA_cunglq_Tile_Async](#)
[PLASMA_cunglq_Tile](#)
[PLASMA_dunglq_Tile](#)
[PLASMA_sunglq_Tile](#)
[PLASMA_cgelqf_Tile](#)

3.6.2.32 int PLASMA_cungqr_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * Q)

PLASMA_cungqr_Tile - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_cgeqrf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_cgeqrf.
- ← *T* Auxiliary factorization data, computed by PLASMA_cgeqrf.
- *Q* On exit, the M-by-N matrix Q.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_cungqr](#)
[PLASMA_cungqr_Tile_Async](#)
[PLASMA_cungqr_Tile](#)
[PLASMA_dungqr_Tile](#)
[PLASMA_sungqr_Tile](#)
[PLASMA_cgeqrf_Tile](#)

3.6.2.33 `int PLASMA_cunmlq_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_cunmlq_Tile` - overwrites the general M-by-N matrix `C` with $Q * C$, where `Q` is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by `PLASMA_cgelqf_Tile`. `Q` is of order `M`. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Intended usage: = `PlasmaLeft`: apply Q or $Q^* H$ from the left; = `PlasmaRight`: apply Q or $Q^* H$ from the right. Currently only `PlasmaLeft` is supported.
- ← *trans* Intended usage: = `PlasmaNoTrans`: no transpose, apply Q ; = `PlasmaConjTrans`: conjugate transpose, apply $Q^* H$. Currently only `PlasmaConjTrans` is supported.
- ← *A* Details of the LQ factorization of the original matrix `A` as returned by `PLASMA_cgelqf`.
- ← *T* Auxiliary factorization data, computed by `PLASMA_cgelqf`.
- ↔ *B* On entry, the M-by-N matrix `B`. On exit, `B` is overwritten by $Q * B$ or $Q^* H * B$.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_cunmlq](#)
[PLASMA_cunmlq_Tile_Async](#)
[PLASMA_cunmlq_Tile](#)
[PLASMA_dunmlq_Tile](#)
[PLASMA_sunmlq_Tile](#)
[PLASMA_cgelqf_Tile](#)

3.6.2.34 `int PLASMA_cunmqr_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_cunmqr_Tile` - overwrites the general M-by-N matrix `C` with $Q * C$, where `Q` is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by `PLASMA_cgeqrf_Tile`. `Q` is of order `M`. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Intended usage: = `PlasmaLeft`: apply Q or $Q^* H$ from the left; = `PlasmaRight`: apply Q or $Q^* H$ from the right. Currently only `PlasmaLeft` is supported.
- ← *trans* Intended usage: = `PlasmaNoTrans`: no transpose, apply Q ; = `PlasmaConjTrans`: conjugate transpose, apply $Q^* H$. Currently only `PlasmaConjTrans` is supported.
- ← *A* Details of the QR factorization of the original matrix `A` as returned by `PLASMA_cgeqrf`.
- ← *T* Auxiliary factorization data, computed by `PLASMA_cgeqrf`.
- ↔ *B* On entry, the M-by-N matrix `B`. On exit, `B` is overwritten by $Q * B$ or $Q^* H * B$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_cumqr](#)
[PLASMA_cumqr_Tile_Async](#)
[PLASMA_cumqr_Tile](#)
[PLASMA_dunmqr_Tile](#)
[PLASMA_sunmqr_Tile](#)
[PLASMA_cgeqrf_Tile](#)

3.7 Advanced Interface: Synchronous - Double Real

Functions/Subroutines

- int [PLASMA_dgelqf_Tile](#) (PLASMA_desc *A, PLASMA_desc *T)
- int [PLASMA_dgelqs_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_dgels_Tile](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_dgemm_Tile](#) (PLASMA_enum transA, PLASMA_enum transB, double alpha, PLASMA_desc *A, PLASMA_desc *B, double beta, PLASMA_desc *C)
- int [PLASMA_dgeqrf_Tile](#) (PLASMA_desc *A, PLASMA_desc *T)
- int [PLASMA_dgeqrs_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_dgesv_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int [PLASMA_dgetrf_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV)
- int [PLASMA_dgetrs_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- double [PLASMA_dlange_Tile](#) (PLASMA_enum norm, PLASMA_desc *A, double *work)
- double [PLASMA_dlansy_Tile](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, double *work)
- int [PLASMA_dlauum_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_dorglq_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_dorgqr_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q)
- int [PLASMA_dormlq_Tile](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_dormqr_Tile](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_dpLgsy_Tile](#) (double bump, PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_dprnt_Tile](#) (PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_dposv_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_dpotrj_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_dpotri_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_dpotrs_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_dsgevs_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_desc *X, int *ITER)
- int [PLASMA_dsposv_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_desc *X, int *ITER)
- int [PLASMA_dsungesv_Tile](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_desc *X, int *ITER)
- int [PLASMA_dsymm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, double alpha, PLASMA_desc *A, PLASMA_desc *B, double beta, PLASMA_desc *C)
- int [PLASMA_dsyrr2k_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc *A, PLASMA_desc *B, double beta, PLASMA_desc *C)
- int [PLASMA_dsyrrk_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc *A, double beta, PLASMA_desc *C)
- int [PLASMA_dtrmm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_dtrsm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_dtrsmpi_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int [PLASMA_dtrtri_Tile](#) (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A)

3.7.1 Detailed Description

This is the group of double real functions using the advanced synchronous interface.

3.7.2 Function/Subroutine Documentation

3.7.2.1 `int PLASMA_dgelqf_Tile (PLASMA_desc * A, PLASMA_desc * T)`

`PLASMA_dgelqf_Tile` - Computes the tile LQ factorization of a matrix. Tile equivalent of `PLASMA_dgelqf()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if $M \leq N$); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.
- **T** On exit, auxiliary factorization data, required by `PLASMA_dgelqs` to solve the system of equations.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_dgelqf](#)
[PLASMA_dgelqf_Tile_Async](#)
[PLASMA_cgelqf_Tile](#)
[PLASMA_dgelqf_Tile](#)
[PLASMA_sgelqf_Tile](#)
[PLASMA_dgelqs_Tile](#)

3.7.2.2 `int PLASMA_dgelqs_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_dgelqs_Tile` - Computes a minimum-norm solution using previously computed LQ factorization. Tile equivalent of `PLASMA_dgelqs()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **A** Details of the LQ factorization of the original matrix A as returned by `PLASMA_dgelqf`.
- ← **T** Auxiliary factorization data, computed by `PLASMA_dgelqf`.
- ↔ **B** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_dgelqs](#)
[PLASMA_dgelqs_Tile_Async](#)
[PLASMA_cgelqs_Tile](#)
[PLASMA_dgelqs_Tile](#)
[PLASMA_sgelqs_Tile](#)
[PLASMA_dgelqf_Tile](#)

3.7.2.3 `int PLASMA_dgels_Tile (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_dgels_Tile` - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Tile equivalent of `PLASMA_dgels()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *trans* Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves $A \setminus * \setminus * T$. Currently only PlasmaNoTrans is supported.
- ↔ *A* On entry, the M-by-N matrix A. On exit, if $M \geq N$, A is overwritten by details of its QR factorization as returned by `PLASMA_dgeqrf`; if $M < N$, A is overwritten by details of its LQ factorization as returned by `PLASMA_dgelqf`.
- *T* On exit, auxiliary factorization data.
- ↔ *B* On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if $M < N$, rows 1 to N of B contain the minimum norm solution vectors;

Returns

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_dgels](#)
[PLASMA_dgels_Tile_Async](#)
[PLASMA_cgels_Tile](#)
[PLASMA_dgels_Tile](#)
[PLASMA_sgels_Tile](#)

3.7.2.4 `int PLASMA_dgemm_Tile (PLASMA_enum transA, PLASMA_enum transB, double alpha, PLASMA_desc * A, PLASMA_desc * B, double beta, PLASMA_desc * C)`

`PLASMA_dgemm_Tile` - Performs matrix multiplication. Tile equivalent of `PLASMA_dgemm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *transA* Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaTrans: A is ugate transposed.
- ← *transB* Specifies whether the matrix B is transposed, not transposed or ugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaTrans: B is ugate transposed.
- ← *alpha* alpha specifies the scalar alpha
- ← *A* A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.
- ← *B* B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha*op(A)*op(B) + beta*C)

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dgemm](#)
[PLASMA_dgemm_Tile_Async](#)
[PLASMA_cgemm_Tile](#)
[PLASMA_dgemm_Tile](#)
[PLASMA_sgemm_Tile](#)

3.7.2.5 int PLASMA_dgeqrf_Tile (PLASMA_desc * A, PLASMA_desc * T)

PLASMA_dgeqrf_Tile - Computes the tile QR factorization of a matrix. Tile equivalent of [PLASMA_dgeqrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ *A* On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal represent the unitary matrix Q as a product of elementary reflectors stored by tiles.
- *T* On exit, auxiliary factorization data, required by *PLASMA_dgeqrs* to solve the system of equations.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dgeqrf](#)
[PLASMA_dgeqrf_Tile_Async](#)
[PLASMA_cgeqrf_Tile](#)
[PLASMA_dgeqrf_Tile](#)
[PLASMA_sgeqrf_Tile](#)
[PLASMA_dgeqrs_Tile](#)

3.7.2.6 int PLASMA_dgeqrs_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)

PLASMA_dgeqrs_Tile - Computes a minimum-norm solution using the tile QR factorization. Tile equivalent of [PLASMA_dgetrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.
- ← **T** Auxiliary factorization data, computed by PLASMA_dgeqrf.
- ↔ **B** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dgeqrs](#)
[PLASMA_dgeqrs_Tile_Async](#)
[PLASMA_cgeqrs_Tile](#)
[PLASMA_dgeqrs_Tile](#)
[PLASMA_sgeqrs_Tile](#)
[PLASMA_dgeqrf_Tile](#)

3.7.2.7 int PLASMA_dgesv_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)

PLASMA_dgesv_Tile - Solves a system of linear equations using the tile LU factorization. Tile equivalent of [PLASMA_dgetrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).
- ↔ **L** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.
- **IPIV** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_dgesv](#)
[PLASMA_dgesv_Tile_Async](#)
[PLASMA_cgesv_Tile](#)
[PLASMA_dgesv_Tile](#)
[PLASMA_sgesv_Tile](#)
[PLASMA_dcgsv_Tile](#)

3.7.2.8 int PLASMA_dgetrf_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV)

PLASMA_dgetrf_Tile - Computes the tile LU factorization of a matrix. Tile equivalent of [PLASMA_dgetrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ **A** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

→ **L** On exit, auxiliary factorization data, related to the tile L factor, required by **PLASMA_dgetrs** to solve the system of equations.

→ **IPIV** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

See also

[PLASMA_dgetrf](#)
[PLASMA_dgetrf_Tile_Async](#)
[PLASMA_cgetrf_Tile](#)
[PLASMA_dgetrf_Tile](#)
[PLASMA_sgetrf_Tile](#)
[PLASMA_dgetrs_Tile](#)

3.7.2.9 `int PLASMA_dgetrs_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)`

`PLASMA_dgetrs_Tile` - Solves a system of linear equations using previously computed LU factorization. Tile equivalent of `PLASMA_dgetrs()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* The tile factors L and U from the factorization, computed by `PLASMA_dgetrf`.
- ← *L* Auxiliary factorization data, related to the tile L factor, computed by `PLASMA_dgetrf`.
- ← *IPIV* The pivot indices from `PLASMA_dgetrf` (not equivalent to LAPACK).
- ↔ *B* On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_dgetrs](#)
[PLASMA_dgetrs_Tile_Async](#)
[PLASMA_cgetrs_Tile](#)
[PLASMA_dgetrs_Tile](#)
[PLASMA_sgetrs_Tile](#)
[PLASMA_dgetrf_Tile](#)

3.7.2.10 `double PLASMA_dlange_Tile (PLASMA_enum norm, PLASMA_desc * A, double * work)`

`PLASMA_dlange_Tile` - Tile equivalent of `PLASMA_dlange()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if `UPLO = 'U'`, the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if `UPLO = 'L'`, the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_dlange](#)

[PLASMA_dlange_Tile_Async](#)
[PLASMA_clange_Tile](#)
[PLASMA_dlange_Tile](#)
[PLASMA_slange_Tile](#)

3.7.2.11 `double PLASMA_dlansy_Tile (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc * A, double * work)`

PLASMA_dlansy_Tile - Tile equivalent of [PLASMA_dlansy\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_dlansy](#)
[PLASMA_dlansy_Tile_Async](#)
[PLASMA_clansy_Tile](#)
[PLASMA_dlansy_Tile](#)
[PLASMA_slansy_Tile](#)

3.7.2.12 `int PLASMA_dlauum_Tile (PLASMA_enum uplo, PLASMA_desc * A)`

PLASMA_dlauum_Tile - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Tile equivalent of [PLASMA_dlauum\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_dlauum](#)
[PLASMA_dlauum_Tile_Async](#)
[PLASMA_clauum_Tile](#)
[PLASMA_dlauum_Tile](#)
[PLASMA_slauum_Tile](#)
[PLASMA_dpotri_Tile](#)

3.7.2.13 int PLASMA_dorglq_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)

PLASMA_dorglq_Tile - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_dgelqf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* Details of the LQ factorization of the original matrix A as returned by PLASMA_dgelqf.
 ← *T* Auxiliary factorization data, computed by PLASMA_dgelqf.
 → *B* On exit, the M-by-N matrix Q.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dorglq](#)
[PLASMA_dorglq_Tile_Async](#)
[PLASMA_cunglq_Tile](#)
[PLASMA_dunglq_Tile](#)
[PLASMA_sunglq_Tile](#)
[PLASMA_dgelqf_Tile](#)

3.7.2.14 int PLASMA_dorgqr_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * Q)

PLASMA_dorgqr_Tile - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_dgeqrf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.
 ← *T* Auxiliary factorization data, computed by PLASMA_dgeqrf.
 → *Q* On exit, the M-by-N matrix Q.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_dorgqr](#)
[PLASMA_dorgqr_Tile_Async](#)
[PLASMA_cungqr_Tile](#)
[PLASMA_dungqr_Tile](#)
[PLASMA_sungqr_Tile](#)
[PLASMA_dgeqrf_Tile](#)

3.7.2.15 `int PLASMA_dormlq_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_dormlq_Tile` - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by `PLASMA_dgelqf_Tile` Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q^T from the left; = PlasmaRight: apply Q or Q^T from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: apply transpose, apply Q^T . Currently only PlasmaTrans is supported.
- ← *A* Details of the LQ factorization of the original matrix A as returned by `PLASMA_dgelqf`.
- ← *T* Auxiliary factorization data, computed by `PLASMA_dgelqf`.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by $Q^T B$ or $Q B^T$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dormlq](#)
[PLASMA_dormlq_Tile_Async](#)
[PLASMA_cunmlq_Tile](#)
[PLASMA_dunmlq_Tile](#)
[PLASMA_sunmlq_Tile](#)
[PLASMA_dgelqf_Tile](#)

3.7.2.16 `int PLASMA_dormqr_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_dormqr_Tile` - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by `PLASMA_dgeqrf_Tile` Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q^*T from the left; = PlasmaRight: apply Q or Q^*T from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q ; = PlasmaTrans: ugate transpose, apply Q^*T . Currently only PlasmaTrans is supported.
- ← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.
- ← *T* Auxiliary factorization data, computed by PLASMA_dgeqrf.
- ↔ *B* On entry, the M -by- N matrix B . On exit, B is overwritten by Q^*B or Q^*T^*B .

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dormqr](#)
[PLASMA_dormqr_Tile_Async](#)
[PLASMA_cunmqr_Tile](#)
[PLASMA_dunmqr_Tile](#)
[PLASMA_sunmqr_Tile](#)
[PLASMA_dgeqrf_Tile](#)

3.7.2.17 `int PLASMA_dpigsy_Tile (double bump, PLASMA_desc * A, unsigned long long int seed)`

PLASMA_dpigsy_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of [PLASMA_dpigsy\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* On exit, The random hermitian matrix A generated.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dpigsy](#)
[PLASMA_dpigsy_Tile_Async](#)
[PLASMA_cpigsy_Tile](#)
[PLASMA_dpigsy_Tile](#)
[PLASMA_spigsy_Tile](#)
[PLASMA_dplrnt_Tile](#)
[PLASMA_dpigsy_Tile](#)

3.7.2.18 `int PLASMA_dplrnt_Tile (PLASMA_desc * A, unsigned long long int seed)`

`PLASMA_dplrnt_Tile` - Generate a random matrix by tiles. Tile equivalent of `PLASMA_dplrnt()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* On exit, The random matrix *A* generated.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_dplrnt](#)
[PLASMA_dplrnt_Tile_Async](#)
[PLASMA_cplrnt_Tile](#)
[PLASMA_dplrnt_Tile](#)
[PLASMA_splrnt_Tile](#)
[PLASMA_dplgsy_Tile](#)
[PLASMA_dplgsy_Tile](#)

3.7.2.19 `int PLASMA_dposv_Tile (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B)`

`PLASMA_dposv_Tile` - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Tile equivalent of `PLASMA_dposv()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* Specifies whether the matrix *A* is upper triangular or lower triangular: = `PlasmaUpper`: Upper triangle of *A* is stored; = `PlasmaLower`: Lower triangle of *A* is stored.

↔ *A* On entry, the symmetric positive definite (or Hermitian) matrix *A*. If *uplo* = `PlasmaUpper`, the leading *N*-by-*N* upper triangular part of *A* contains the upper triangular part of the matrix *A*, and the strictly lower triangular part of *A* is not referenced. If *UPLO* = 'L', the leading *N*-by-*N* lower triangular part of *A* contains the lower triangular part of the matrix *A*, and the strictly upper triangular part of *A* is not referenced. On exit, if return value = 0, the factor *U* or *L* from the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$.

↔ *B* On entry, the *N*-by-*NRHS* right hand side matrix *B*. On exit, if return value = 0, the *N*-by-*NRHS* solution matrix *X*.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

>0 if i , the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_dposv](#)
[PLASMA_dposv_Tile_Async](#)
[PLASMA_cposv_Tile](#)
[PLASMA_dposv_Tile](#)
[PLASMA_sposv_Tile](#)

3.7.2.20 int PLASMA_dpotrf_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*)

PLASMA_dpotrf_Tile - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Tile equivalent of [PLASMA_dpotrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
 ← A On entry, the symmetric positive definite (or Hermitian) matrix A . If *uplo* = PlasmaUpper, the leading N -by- N upper triangular part of A contains the upper triangular part of the matrix A , and the strictly lower triangular part of A is not referenced. If *UPLO* = 'L', the leading N -by- N lower triangular part of A contains the lower triangular part of the matrix A , and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i , the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_dpotrf](#)
[PLASMA_dpotrf_Tile_Async](#)
[PLASMA_cpotrf_Tile](#)
[PLASMA_dpotrf_Tile](#)
[PLASMA_spotrf_Tile](#)
[PLASMA_dpotrs_Tile](#)

3.7.2.21 int PLASMA_dpotri_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*)

PLASMA_dpotri_Tile - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$ computed by [PLASMA_dpotrf](#). Tile equivalent of [PLASMA_dpotri\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L from the Cholesky factorization $A = U^*U$ or $A = LL^*$, as computed by PLASMA_dpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- >0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_dpotri](#)
[PLASMA_dpotri_Tile_Async](#)
[PLASMA_cpotri_Tile](#)
[PLASMA_dpotri_Tile](#)
[PLASMA_spotri_Tile](#)
[PLASMA_dpotrf_Tile](#)

3.7.2.22 int PLASMA_dpotrs_Tile (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B)

PLASMA_dpotrs_Tile - Solves a system of linear equations using previously computed Cholesky factorization. Tile equivalent of [PLASMA_dpotrs\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* The triangular factor U or L from the Cholesky factorization $A = U^*U$ or $A = LL^*$, computed by PLASMA_dpotrf.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

- PLASMA_SUCCESS* successful exit

See also

[PLASMA_dpotrs](#)
[PLASMA_dpotrs_Tile_Async](#)
[PLASMA_cpotrs_Tile](#)
[PLASMA_dpotrs_Tile](#)
[PLASMA_spotrs_Tile](#)
[PLASMA_dpotrf_Tile](#)

3.7.2.23 `int PLASMA_dsgesv_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_desc * X, int * ITER)`

`PLASMA_dsgesv_Tile` - Solves a system of linear equations using the tile LU factorization and mixed-precision iterative refinement. Tile equivalent of `PLASMA_dsgesv()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the N-by-N coefficient matrix A.
 - If the iterative refinement converged, A is not modified;
 - otherwise, it fell back to double precision solution, and then A contains the tile L and U factors from the factorization (not equivalent to LAPACK).
- **L** On exit:
 - if the iterative refinement converged, L is not modified;
 - otherwise, it fell back to double precision solution, and then L is an auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations (not equivalent to LAPACK).
- **IPIV** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

- `PLASMA_SUCCESS` successful exit
- >0 if i, `U(i,i)` is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_dsgesv](#)
[PLASMA_dsgesv_Tile_Async](#)
[PLASMA_dsgesv_Tile](#)
[PLASMA_dgesv_Tile](#)

3.7.2.24 `int PLASMA_dsposv_Tile (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_desc * X, int * ITER)`

`PLASMA_dsposv_Tile` - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization and mixed-precision iterative refinement. Tile equivalent of `PLASMA_dsposv()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = `PlasmaUpper`: Upper triangle of A is stored; = `PlasmaLower`: Lower triangle of A is stored.

- ↔ **A** On entry, the N-by-N symmetric positive definite (or Hermitian) coefficient matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.
- If the iterative refinement converged, A is not modified;
 - otherwise, it failed backed to double precision solution,
- ↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_dsposv](#)

[PLASMA_dsposv_Tile_Async](#)

[PLASMA_dsposv_Tile](#)

[PLASMA_zposv_Tile](#)

3.7.2.25 int PLASMA_dsungesv_Tile (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_desc * X, int * ITER)

PLASMA_dsungesv_Tile - Solves symmetric linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Tile equivalent of [PLASMA_dsungesv\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **trans** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves A**H. Currently only PlasmaNoTrans is supported.
- ↔ **A**
- If the iterative refinement converged, A is not modified;
 - otherwise, it fell back to double precision solution, and on exit the M-by-N matrix A contains: if M >= N, A is overwritten by details of its QR factorization as returned by [PLASMA_zgeqrf](#); if M < N, A is overwritten by details of its LQ factorization as returned by [PLASMA_zgelqf](#).
- **T** On exit:
- if the iterative refinement converged, T is not modified;
 - otherwise, it fell back to double precision solution, and then T is an auxiliary factorization data.
- ↔ **B** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dsungesv](#)
[PLASMA_dsungesv_Tile_Async](#)
[PLASMA_dsungesv_Tile](#)
[PLASMA_zgels_Tile](#)

3.7.2.26 int PLASMA_dsymm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, double *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, double *beta*, PLASMA_desc * *C*)

PLASMA_dsymm_Tile - Performs symmetric matrix multiplication. Tile equivalent of [PLASMA_dsymm\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *side* Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← *uplo* Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← *alpha* Specifies the scalar alpha.

← *A* A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← *B* B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← *beta* Specifies the scalar beta.

↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dsymm](#)
[PLASMA_dsymm_Tile_Async](#)
[PLASMA_csymm_Tile](#)
[PLASMA_dsymm_Tile](#)
[PLASMA_ssymm_Tile](#)

3.7.2.27 `int PLASMA_dsyr2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc * A, PLASMA_desc * B, double beta, PLASMA_desc * C)`

`PLASMA_dsyr2k_Tile` - Performs symmetric rank k update. Tile equivalent of `PLASMA_dsyr2k()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when *trans* = PlasmaNoTrans, and is N otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array *uplo* part of the matrix is overwritten by the *uplo* part of the updated matrix.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_dsyr2k_Tile](#)
[PLASMA_csyr2k](#)
[PLASMA_dsyr2k](#)
[PLASMA_ssyr2k](#)

3.7.2.28 `int PLASMA_dsyrk_Tile (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc * A, double beta, PLASMA_desc * C)`

`PLASMA_dsyrk_Tile` - Performs rank k update. Tile equivalent of `PLASMA_dsyrk()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← *trans* Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed.
- ← *alpha* alpha specifies the scalar alpha.
- ← *A* A is a LDA-by-ka matrix, where ka is K when *trans* = PlasmaNoTrans, and is N otherwise.
- ← *beta* beta specifies the scalar beta
- ↔ *C* C is a LDC-by-N matrix. On exit, the array *uplo* part of the matrix is overwritten by the *uplo* part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dsyrc_Tile](#)

[PLASMA_csyrc](#)

[PLASMA_dsyrc](#)

[PLASMA_ssyrc](#)

3.7.2.29 `int PLASMA_dtrmm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc * A, PLASMA_desc * B)`

PLASMA_dtrmm_Tile - Computes triangular solve. Tile equivalent of [PLASMA_dtrmm\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *transA* Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.
- ← *diag* Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.
- ← *A* The triangular matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If *uplo* = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If *diag* = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dtrmm](#)

[PLASMA_dtrmm_Tile_Async](#)
[PLASMA_ctrmm_Tile](#)
[PLASMA_dtrmm_Tile](#)
[PLASMA_strmm_Tile](#)

3.7.2.30 `int PLASMA_dtrsm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc * A, PLASMA_desc * B)`

`PLASMA_dtrsm_Tile` - Computes triangular solve. Tile equivalent of `PLASMA_dtrsm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Specifies whether A appears on the left or on the right of X: = PlasmaLeft: $A * X = B$ = PlasmaRight: $X * A = B$
- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *transA* Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.
- ← *diag* Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.
- ← *A* The triangular matrix A. If *uplo* = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If *uplo* = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If *diag* = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_dtrsm](#)
[PLASMA_dtrsm_Tile_Async](#)
[PLASMA_ctrsm_Tile](#)
[PLASMA_dtrsm_Tile](#)
[PLASMA_strsm_Tile](#)

3.7.2.31 `int PLASMA_dtrsmpL_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)`

`PLASMA_dtrsmpL_Tile` - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **A** The tile factor L from the factorization, computed by PLASMA_dgetrf.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_dgetrf.
- ← **IPIV** The pivot indices from PLASMA_dgetrf (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_dtrsml](#)
[PLASMA_dtrsml_Tile_Async](#)
[PLASMA_ctrsmpl_Tile](#)
[PLASMA_dtrsml_Tile](#)
[PLASMA_strsmpl_Tile](#)
[PLASMA_dgetrf_Tile](#)

3.7.2.32 int PLASMA_dtrtri_Tile (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc * *A*)

PLASMA_dtrtri_Tile - Computes the inverse of a complex upper or lower triangular matrix A. Tile equivalent of [PLASMA_dtrtri\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **uplo** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **diag** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.
- ← **A** On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

Returns**Return values**

PLASMA_SUCCESS successful exit

>0 if $A(i,i)$ is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

[PLASMA_dtrtri](#)

PLASMA_dtrtri_Tile_Async
PLASMA_ctrtri_Tile
PLASMA_dtrtri_Tile
PLASMA_strtri_Tile
PLASMA_dpotri_Tile

3.8 Advanced Interface: Synchronous - Single Real

Functions/Subroutines

- int [PLASMA_sgelqf_Tile](#) (PLASMA_desc *A, PLASMA_desc *T)
- int [PLASMA_sgelqs_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_sgels_Tile](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_sgemm_Tile](#) (PLASMA_enum transA, PLASMA_enum transB, float alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C)
- int [PLASMA_sgeqrf_Tile](#) (PLASMA_desc *A, PLASMA_desc *T)
- int [PLASMA_sgeqrs_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_sgesv_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int [PLASMA_sgetrf_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV)
- int [PLASMA_sgetrs_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- float [PLASMA_slange_Tile](#) (PLASMA_enum norm, PLASMA_desc *A, float *work)
- float [PLASMA_slansy_Tile](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work)
- int [PLASMA_slauum_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_sorglq_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_sorgqr_Tile](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q)
- int [PLASMA_sormlq_Tile](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_sormqr_Tile](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int [PLASMA_splgsy_Tile](#) (float bump, PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_splrnt_Tile](#) (PLASMA_desc *A, unsigned long long int seed)
- int [PLASMA_sposv_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_spotrf_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_spotri_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A)
- int [PLASMA_spotrs_Tile](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_ssymm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, float alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C)
- int [PLASMA_ssy2k_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C)
- int [PLASMA_ssy2k_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C)
- int [PLASMA_ssy2k_Tile](#) (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C)
- int [PLASMA_strmm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_strsm_Tile](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc *A, PLASMA_desc *B)
- int [PLASMA_strsmpl_Tile](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int [PLASMA_strtri_Tile](#) (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A)

3.8.1 Detailed Description

This is the group of single real functions using the advanced synchronous interface.

3.8.2 Function/Subroutine Documentation

3.8.2.1 `int PLASMA_sgelqf_Tile (PLASMA_desc * A, PLASMA_desc * T)`

`PLASMA_sgelqf_Tile` - Computes the tile LQ factorization of a matrix. Tile equivalent of `PLASMA_sgelqf()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if $M \leq N$); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.
- **T** On exit, auxiliary factorization data, required by `PLASMA_sgelqs` to solve the system of equations.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_sgelqf](#)
[PLASMA_sgelqf_Tile_Async](#)
[PLASMA_cgelqf_Tile](#)
[PLASMA_dgelqf_Tile](#)
[PLASMA_sgelqf_Tile](#)
[PLASMA_sgelqs_Tile](#)

3.8.2.2 `int PLASMA_sgelqs_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_sgelqs_Tile` - Computes a minimum-norm solution using previously computed LQ factorization. Tile equivalent of `PLASMA_sgelqs()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **A** Details of the LQ factorization of the original matrix A as returned by `PLASMA_sgelqf`.
- ← **T** Auxiliary factorization data, computed by `PLASMA_sgelqf`.
- ↔ **B** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_sgelqs](#)
[PLASMA_sgelqs_Tile_Async](#)
[PLASMA_cgelqs_Tile](#)
[PLASMA_dgelqs_Tile](#)
[PLASMA_sgelqs_Tile](#)
[PLASMA_sgelqf_Tile](#)

3.8.2.3 int PLASMA_sgels_Tile (PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_sgels_Tile - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Tile equivalent of [PLASMA_sgels\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *trans* Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves A**T. Currently only PlasmaNoTrans is supported.
- ↔ *A* On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by [PLASMA_sgeqrf](#); if M < N, A is overwritten by details of its LQ factorization as returned by [PLASMA_sgelqf](#).
- *T* On exit, auxiliary factorization data.
- ↔ *B* On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns

PLASMA_SUCCESS successful exit

See also

[PLASMA_sgels](#)
[PLASMA_sgels_Tile_Async](#)
[PLASMA_cgels_Tile](#)
[PLASMA_dgels_Tile](#)
[PLASMA_sgels_Tile](#)

3.8.2.4 int PLASMA_sgemm_Tile (PLASMA_enum *transA*, PLASMA_enum *transB*, float *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, float *beta*, PLASMA_desc * *C*)

PLASMA_sgemm_Tile - Performs matrix multiplication. Tile equivalent of [PLASMA_sgemm\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *transA* Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaTrans: A is ugate transposed.

- ← **transB** Specifies whether the matrix B is transposed, not transposed or ugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaTrans: B is ugate transposed.
- ← **alpha** alpha specifies the scalar alpha
- ← **A** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.
- ← **B** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.
- ← **beta** beta specifies the scalar beta
- ↔ **C** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha*op(A)*op(B) + beta*C)

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_sgemm](#)
[PLASMA_sgemm_Tile_Async](#)
[PLASMA_cgemm_Tile](#)
[PLASMA_dgemm_Tile](#)
[PLASMA_sgemm_Tile](#)

3.8.2.5 int PLASMA_sgeqrf_Tile (PLASMA_desc * A, PLASMA_desc * T)

PLASMA_sgeqrf_Tile - Computes the tile QR factorization of a matrix. Tile equivalent of [PLASMA_sgeqrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal represent the unitary matrix Q as a product of elementary reflectors stored by tiles.
- **T** On exit, auxiliary factorization data, required by PLASMA_sgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_sgeqrf](#)
[PLASMA_sgeqrf_Tile_Async](#)
[PLASMA_cgeqrf_Tile](#)
[PLASMA_dgeqrf_Tile](#)
[PLASMA_sgeqrf_Tile](#)
[PLASMA_sgeqrs_Tile](#)

3.8.2.6 `int PLASMA_sgeqrs_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_sgeqrs_Tile` - Computes a minimum-norm solution using the tile QR factorization. Tile equivalent of `PLASMA_sgetrf()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** Details of the QR factorization of the original matrix A as returned by `PLASMA_sgeqrf`.
- ← **T** Auxiliary factorization data, computed by `PLASMA_sgeqrf`.
- ↔ **B** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_sgeqrs](#)
[PLASMA_sgeqrs_Tile_Async](#)
[PLASMA_cgeqrs_Tile](#)
[PLASMA_dgeqrs_Tile](#)
[PLASMA_sgeqrs_Tile](#)
[PLASMA_sgeqrf_Tile](#)

3.8.2.7 `int PLASMA_sgesv_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)`

`PLASMA_sgesv_Tile` - Solves a system of linear equations using the tile LU factorization. Tile equivalent of `PLASMA_sgetrf()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ **A** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).
- ↔ **L** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.
- **IPIV** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

>0 if i , $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

[PLASMA_sgesv](#)
[PLASMA_sgesv_Tile_Async](#)
[PLASMA_cgesv_Tile](#)
[PLASMA_dgesv_Tile](#)
[PLASMA_sgesv_Tile](#)
[PLASMA_scgesv_Tile](#)

3.8.2.8 int PLASMA_sgetrf_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV)

`PLASMA_sgetrf_Tile` - Computes the tile LU factorization of a matrix. Tile equivalent of `PLASMA_sgetrf()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ A On entry, the M -by- N matrix to be factored. On exit, the tile factors L and U from the factorization.
- L On exit, auxiliary factorization data, related to the tile L factor, required by `PLASMA_sgetrs` to solve the system of equations.
- $IPIV$ The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

`PLASMA_SUCCESS` successful exit

>0 if i , $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

See also

[PLASMA_sgetrf](#)
[PLASMA_sgetrf_Tile_Async](#)
[PLASMA_cgetrf_Tile](#)
[PLASMA_dgetrf_Tile](#)
[PLASMA_sgetrf_Tile](#)
[PLASMA_sgetrs_Tile](#)

3.8.2.9 int PLASMA_sgetrs_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)

`PLASMA_sgetrs_Tile` - Solves a system of linear equations using previously computed LU factorization. Tile equivalent of `PLASMA_sgetrs()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **A** The tile factors L and U from the factorization, computed by PLASMA_sgetrf.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_sgetrf.
- ← **IPIV** The pivot indices from PLASMA_sgetrf (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_sgetrs](#)
[PLASMA_sgetrs_Tile_Async](#)
[PLASMA_cgetrs_Tile](#)
[PLASMA_dgetrs_Tile](#)
[PLASMA_sgetrs_Tile](#)
[PLASMA_sgetrf_Tile](#)

3.8.2.10 float PLASMA_slange_Tile (PLASMA_enum norm, PLASMA_desc * A, float * work)

PLASMA_slange_Tile - Tile equivalent of [PLASMA_slange\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **uplo** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **A** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_slange](#)
[PLASMA_slange_Tile_Async](#)
[PLASMA_clange_Tile](#)
[PLASMA_dlange_Tile](#)
[PLASMA_slange_Tile](#)

3.8.2.11 float PLASMA_slansy_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc * *A*, float * *work*)

PLASMA_slansy_Tile - Tile equivalent of [PLASMA_slansy\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_slansy](#)
[PLASMA_slansy_Tile_Async](#)
[PLASMA_clansy_Tile](#)
[PLASMA_dlansy_Tile](#)
[PLASMA_slansy_Tile](#)

3.8.2.12 int PLASMA_slauum_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*)

PLASMA_slauum_Tile - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Tile equivalent of [PLASMA_slauum\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *A* On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product $U * U'$; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product $L' * L$.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_slauum](#)
[PLASMA_slauum_Tile_Async](#)

[PLASMA_clauum_Tile](#)
[PLASMA_dlauum_Tile](#)
[PLASMA_slauum_Tile](#)
[PLASMA_spotri_Tile](#)

3.8.2.13 `int PLASMA_sorglq_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B)`

`PLASMA_sorglq_Tile` - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by `PLASMA_sgelqf`. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* Details of the LQ factorization of the original matrix A as returned by `PLASMA_sgelqf`.
- ← *T* Auxiliary factorization data, computed by `PLASMA_sgelqf`.
- *B* On exit, the M-by-N matrix Q.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_sorglq](#)
[PLASMA_sorglq_Tile_Async](#)
[PLASMA_cunglq_Tile](#)
[PLASMA_dunglq_Tile](#)
[PLASMA_sunglq_Tile](#)
[PLASMA_sgelqf_Tile](#)

3.8.2.14 `int PLASMA_sorgqr_Tile (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * Q)`

`PLASMA_sorgqr_Tile` - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by `PLASMA_sgeqrf`. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* Details of the QR factorization of the original matrix A as returned by `PLASMA_sgeqrf`.
- ← *T* Auxiliary factorization data, computed by `PLASMA_sgeqrf`.
- *Q* On exit, the M-by-N matrix Q.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_sorgqr](#)
[PLASMA_sorgqr_Tile_Async](#)
[PLASMA_cungqr_Tile](#)
[PLASMA_dungqr_Tile](#)
[PLASMA_sungqr_Tile](#)
[PLASMA_sgeqrf_Tile](#)

3.8.2.15 int PLASMA_sormlq_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_sormlq_Tile - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_sgelqf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q^*T from the left; = PlasmaRight: apply Q or Q^*T from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q^*T . Currently only PlasmaTrans is supported.
- ← *A* Details of the LQ factorization of the original matrix A as returned by PLASMA_sgelqf.
- ← *T* Auxiliary factorization data, computed by PLASMA_sgelqf.
- ↔ *B* On entry, the M-by-N matrix B. On exit, B is overwritten by Q^*B or Q^*T^*B .

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_sormlq](#)
[PLASMA_sormlq_Tile_Async](#)
[PLASMA_cunmlq_Tile](#)
[PLASMA_dunmlq_Tile](#)
[PLASMA_sunmlq_Tile](#)
[PLASMA_sgelqf_Tile](#)

3.8.2.16 int PLASMA_sormqr_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_sormqr_Tile - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_sgeqrf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Intended usage: = PlasmaLeft: apply Q or Q^*T from the left; = PlasmaRight: apply Q or Q^*T from the right. Currently only PlasmaLeft is supported.
- ← *trans* Intended usage: = PlasmaNoTrans: no transpose, apply Q ; = PlasmaTrans: ugate transpose, apply Q^*T . Currently only PlasmaTrans is supported.
- ← *A* Details of the QR factorization of the original matrix A as returned by PLASMA_sgeqrf.
- ← *T* Auxiliary factorization data, computed by PLASMA_sgeqrf.
- ↔ *B* On entry, the M -by- N matrix B . On exit, B is overwritten by Q^*B or Q^*T^*B .

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_sormqr](#)
[PLASMA_sormqr_Tile_Async](#)
[PLASMA_cunmqr_Tile](#)
[PLASMA_dunmqr_Tile](#)
[PLASMA_sunmqr_Tile](#)
[PLASMA_sgeqrf_Tile](#)

3.8.2.17 int PLASMA_splgsy_Tile (float bump, PLASMA_desc * A, unsigned long long int seed)

PLASMA_splgsy_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of [PLASMA_splgsy\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *A* On exit, The random hermitian matrix A generated.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_splgsy](#)
[PLASMA_splgsy_Tile_Async](#)
[PLASMA_cpplgsy_Tile](#)
[PLASMA_dpplgsy_Tile](#)
[PLASMA_splgsy_Tile](#)
[PLASMA_splrnt_Tile](#)
[PLASMA_splgsy_Tile](#)

3.8.2.18 `int PLASMA_splrnt_Tile (PLASMA_desc * A, unsigned long long int seed)`

`PLASMA_splrnt_Tile` - Generate a random matrix by tiles. Tile equivalent of `PLASMA_splrnt()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *A* On exit, The random matrix *A* generated.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_splrnt](#)
[PLASMA_splrnt_Tile_Async](#)
[PLASMA_cplrnt_Tile](#)
[PLASMA_dplrnt_Tile](#)
[PLASMA_splrnt_Tile](#)
[PLASMA_splgsy_Tile](#)
[PLASMA_splgsy_Tile](#)

3.8.2.19 `int PLASMA_sposv_Tile (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B)`

`PLASMA_sposv_Tile` - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Tile equivalent of `PLASMA_sposv()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* Specifies whether the matrix *A* is upper triangular or lower triangular: = `PlasmaUpper`: Upper triangle of *A* is stored; = `PlasmaLower`: Lower triangle of *A* is stored.

↔ *A* On entry, the symmetric positive definite (or Hermitian) matrix *A*. If *uplo* = `PlasmaUpper`, the leading *N*-by-*N* upper triangular part of *A* contains the upper triangular part of the matrix *A*, and the strictly lower triangular part of *A* is not referenced. If *UPLO* = 'L', the leading *N*-by-*N* lower triangular part of *A* contains the lower triangular part of the matrix *A*, and the strictly upper triangular part of *A* is not referenced. On exit, if return value = 0, the factor *U* or *L* from the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$.

↔ *B* On entry, the *N*-by-*NRHS* right hand side matrix *B*. On exit, if return value = 0, the *N*-by-*NRHS* solution matrix *X*.

Returns**Return values**

`PLASMA_SUCCESS` successful exit

>0 if i , the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_sposv](#)
[PLASMA_sposv_Tile_Async](#)
[PLASMA_cposv_Tile](#)
[PLASMA_dposv_Tile](#)
[PLASMA_sposv_Tile](#)

3.8.2.20 int PLASMA_spotrf_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*)

PLASMA_spotrf_Tile - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Tile equivalent of [PLASMA_spotrf\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
 ← A On entry, the symmetric positive definite (or Hermitian) matrix A . If *uplo* = PlasmaUpper, the leading N -by- N upper triangular part of A contains the upper triangular part of the matrix A , and the strictly lower triangular part of A is not referenced. If *UPLO* = 'L', the leading N -by- N lower triangular part of A contains the lower triangular part of the matrix A , and the strictly upper triangular part of A is not referenced. On exit, if return value = 0, the factor U or L from the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i , the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_spotrf](#)
[PLASMA_spotrf_Tile_Async](#)
[PLASMA_cpotrf_Tile](#)
[PLASMA_dpotrf_Tile](#)
[PLASMA_spotrf_Tile](#)
[PLASMA_spotrs_Tile](#)

3.8.2.21 int PLASMA_spotri_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*)

PLASMA_spotri_Tile - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$ computed by [PLASMA_spotrf](#). Tile equivalent of [PLASMA_spotri\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← *A* On entry, the triangular factor U or L from the Cholesky factorization $A = U^*U$ or $A = LL^*$, as computed by PLASMA_spotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

Returns**Return values**

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

[PLASMA_spotri](#)
[PLASMA_spotri_Tile_Async](#)
[PLASMA_cpotri_Tile](#)
[PLASMA_dpotri_Tile](#)
[PLASMA_spotri_Tile](#)
[PLASMA_spotrf_Tile](#)

3.8.2.22 int PLASMA_spotrs_Tile (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B)

PLASMA_spotrs_Tile - Solves a system of linear equations using previously computed Cholesky factorization. Tile equivalent of [PLASMA_spotrs\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← *A* The triangular factor U or L from the Cholesky factorization $A = U^*U$ or $A = LL^*$, computed by PLASMA_spotrf.

↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_spotrs](#)
[PLASMA_spotrs_Tile_Async](#)
[PLASMA_cpotrs_Tile](#)
[PLASMA_dpotrs_Tile](#)
[PLASMA_spotrs_Tile](#)
[PLASMA_spotrf_Tile](#)

3.8.2.23 `int PLASMA_ssymm_Tile (PLASMA_enum side, PLASMA_enum uplo, float alpha, PLASMA_desc * A, PLASMA_desc * B, float beta, PLASMA_desc * C)`

`PLASMA_ssymm_Tile` - Performs symmetric matrix multiplication. Tile equivalent of `PLASMA_ssymm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *side* Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← *uplo* Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← *alpha* Specifies the scalar alpha.

← *A* A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← *B* B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← *beta* Specifies the scalar beta.

↔ *C* C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_ssymm](#)
[PLASMA_ssymm_Tile_Async](#)
[PLASMA_csymm_Tile](#)
[PLASMA_dsymm_Tile](#)
[PLASMA_ssymm_Tile](#)

3.8.2.24 `int PLASMA_ssy2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc * A, PLASMA_desc * B, float beta, PLASMA_desc * C)`

`PLASMA_ssy2k_Tile` - Performs symmetric rank k update. Tile equivalent of `PLASMA_ssy2k()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← *uplo* = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

- ← **trans** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.
- ← **alpha** alpha specifies the scalar alpha.
- ← **A** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← **beta** beta specifies the scalar beta
- ↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_ssy2k_Tile](#)
[PLASMA_csy2k](#)
[PLASMA_dsy2k](#)
[PLASMA_ssy2k](#)

3.8.2.25 int PLASMA_ssyk_Tile (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc * A, float beta, PLASMA_desc * C)

PLASMA_ssyk_Tile - Performs rank k update. Tile equivalent of [PLASMA_ssyk\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **uplo** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← **trans** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed.
- ← **alpha** alpha specifies the scalar alpha.
- ← **A** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← **beta** beta specifies the scalar beta
- ↔ **C** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns**Return values**

PLASMA_SUCCESS successful exit

See also

[PLASMA_ssyk_Tile](#)
[PLASMA_csyk](#)
[PLASMA_dsyk](#)
[PLASMA_ssyk](#)

3.8.2.26 `int PLASMA_strmm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc * A, PLASMA_desc * B)`

`PLASMA_strmm_Tile` - Computes triangular solve. Tile equivalent of `PLASMA_strmm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Specifies whether A appears on the left or on the right of $X: = \text{PlasmaLeft}: A * X = B = \text{PlasmaRight}: X * A = B$
- ← *uplo* Specifies whether the matrix A is upper triangular or lower triangular: = `PlasmaUpper`: Upper triangle of A is stored; = `PlasmaLower`: Lower triangle of A is stored.
- ← *transA* Specifies whether the matrix A is transposed, not transposed or ugate transposed: = `PlasmaNoTrans`: A is transposed; = `PlasmaTrans`: A is not transposed; = `PlasmaTrans`: A is ugate transposed.
- ← *diag* Specifies whether or not A is unit triangular: = `PlasmaNonUnit`: A is non unit; = `PlasmaUnit`: A us unit.
- ← *A* The triangular matrix A. If `uplo = PlasmaUpper`, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If `uplo = PlasmaLower`, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If `diag = PlasmaUnit`, the diagonal elements of A are also not referenced and are assumed to be 1.
- ↔ *B* On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

`PLASMA_SUCCESS` successful exit

See also

[PLASMA_strmm](#)
[PLASMA_strmm_Tile_Async](#)
[PLASMA_ctrmm_Tile](#)
[PLASMA_dtrmm_Tile](#)
[PLASMA_strmm_Tile](#)

3.8.2.27 `int PLASMA_strsm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc * A, PLASMA_desc * B)`

`PLASMA_strsm_Tile` - Computes triangular solve. Tile equivalent of `PLASMA_strsm()`. Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *side* Specifies whether A appears on the left or on the right of $X: = \text{PlasmaLeft}: A * X = B = \text{PlasmaRight}: X * A = B$

- ← **uplo** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← **transA** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.
- ← **diag** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.
- ← **A** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_strsm](#)
[PLASMA_strsm_Tile_Async](#)
[PLASMA_ctrsm_Tile](#)
[PLASMA_dtrsm_Tile](#)
[PLASMA_strsm_Tile](#)

3.8.2.28 int PLASMA_strsmpl_Tile (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B)

PLASMA_strsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← **A** The tile factor L from the factorization, computed by PLASMA_sgetrf.
- ← **L** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_sgetrf.
- ← **IPIV** The pivot indices from PLASMA_sgetrf (not equivalent to LAPACK).
- ↔ **B** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

[PLASMA_strsmpl](#)
[PLASMA_strsmpl_Tile_Async](#)
[PLASMA_ctrsmpl_Tile](#)
[PLASMA_dtrsmpl_Tile](#)
[PLASMA_strsmpl_Tile](#)
[PLASMA_sgetrf_Tile](#)

3.8.2.29 `int PLASMA_strtri_Tile (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc * A)`

PLASMA_strtri_Tile - Computes the inverse of a complex upper or lower triangular matrix A. Tile equivalent of [PLASMA_strtri\(\)](#). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← *uplo* = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← *diag* = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.
- ← *A* On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

Returns**Return values**

- PLASMA_SUCCESS* successful exit
- >0 if $A(i,i)$ is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

[PLASMA_strtri](#)
[PLASMA_strtri_Tile_Async](#)
[PLASMA_ctrtri_Tile](#)
[PLASMA_dtrtri_Tile](#)
[PLASMA_strtri_Tile](#)
[PLASMA_spotri_Tile](#)

3.9 Advanced Interface: Asynchronous - Double Complex

Functions/Subroutines

- int [PLASMA_zcgels_Tile_Async](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_desc *X, int *ITER, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zcgesv_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_desc *X, int *ITER, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zcposv_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_desc *X, int *ITER, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zcungesv_Tile_Async](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_desc *X, int *ITER, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zgelqf_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zgelqs_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zgels_Tile_Async](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zgemm_Tile_Async](#) (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex64_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zgeqrf_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zgeqrs_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zgesv_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zgetrf_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zgetrs_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zhemm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex64_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zher2k_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, double beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zherk_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc *A, double beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zlange_Tile_Async](#) (PLASMA_enum norm, PLASMA_desc *A, double *work, double *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zlanhe_Tile_Async](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, double *work, double *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zlansy_Tile_Async](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, double *work, double *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zlauum_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)

- int [PLASMA_zplghe_Tile_Async](#) (double bump, PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zplgsy_Tile_Async](#) (PLASMA_Complex64_t bump, PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zplrnt_Tile_Async](#) (PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zposv_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zpotrf_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zpotri_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zpotrs_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zsymm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex64_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zsyr2k_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex64_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zsyrk_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_Complex64_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ztrmm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ztrsm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ztrsmpl_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ztrtri_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zunglq_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zungqr_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zunmlq_Tile_Async](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zunmqr_Tile_Async](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zLapack_to_Tile_Async](#) (PLASMA_Complex64_t *Af77, int LDA, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_zTile_to_Lapack_Async](#) (PLASMA_desc *A, PLASMA_Complex64_t *Af77, int LDA, PLASMA_sequence *sequence, PLASMA_request *request)

3.9.1 Detailed Description

This is the group of double complex functions using the advanced asynchronous interface.

3.9.2 Function/Subroutine Documentation

3.9.2.1 `int PLASMA_zcgels_Tile_Async (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_desc * X, int * ITER, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_zcgels_Tile_Async - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Non-blocking equivalent of [PLASMA_zcgels_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zcgels](#)
[PLASMA_zcgels_Tile](#)
[PLASMA_dsgels_Tile_Async](#)
[PLASMA_zcgels_Tile_Async](#)

3.9.2.2 `int PLASMA_zcgesv_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_desc * X, int * ITER, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_zcgesv_Tile_Async - Solves a system of linear equations using the tile LU factorization and mixed-precision iterative refinement. Non-blocking equivalent of [PLASMA_zcgesv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zcgesv](#)
[PLASMA_zcgesv_Tile](#)
[PLASMA_dsgesv_Tile_Async](#)
[PLASMA_zcgesv_Tile_Async](#)

3.9.2.3 `int PLASMA_zcposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_desc * X, int * ITER, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_zcposv_Tile_Async - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization and mixed-precision iterative refinement. Non-blocking equivalent of [PLASMA_zcposv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zcposv](#)
[PLASMA_zcposv_Tile](#)
[PLASMA_dsposv_Tile_Async](#)
[PLASMA_zposv_Tile_Async](#)

3.9.2.4 int PLASMA_zcungesv_Tile_Async (PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*, PLASMA_desc * *X*, int * *ITER*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zcungesv_Tile_Async - Solves symmetric linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Non-blocking equivalent of [PLASMA_zcungesv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zcungesv](#)
[PLASMA_zcungesv_Tile](#)
[PLASMA_dsungesv_Tile_Async](#)
[PLASMA_zgels_Tile_Async](#)

3.9.2.5 int PLASMA_zgelqf_Tile_Async (PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zgelqf_Tile_Async - Computes the tile LQ factorization of a matrix. Non-blocking equivalent of [PLASMA_zgelqf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zgelqf](#)
[PLASMA_zgelqf_Tile](#)
[PLASMA_cgelqf_Tile_Async](#)

[PLASMA_dgelqf_Tile_Async](#)
[PLASMA_sgelqf_Tile_Async](#)
[PLASMA_zgelqs_Tile_Async](#)

3.9.2.6 `int PLASMA_zgelqs_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zgelqs_Tile_Async` - Computes a minimum-norm solution using previously computed LQ factorization. Non-blocking equivalent of [PLASMA_zgelqs_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zgelqs](#)
[PLASMA_zgelqs_Tile](#)
[PLASMA_cgelqs_Tile_Async](#)
[PLASMA_dgelqs_Tile_Async](#)
[PLASMA_sgelqs_Tile_Async](#)
[PLASMA_zgelqf_Tile_Async](#)

3.9.2.7 `int PLASMA_zgels_Tile_Async (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zgels_Tile_Async` - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Non-blocking equivalent of [PLASMA_zgels_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zgels](#)
[PLASMA_zgels_Tile](#)
[PLASMA_cgels_Tile_Async](#)
[PLASMA_dgels_Tile_Async](#)
[PLASMA_sgels_Tile_Async](#)

3.9.2.8 `int PLASMA_zgemm_Tile_Async (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex64_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_zgemm_Tile_Async - Performs matrix multiplication. Non-blocking equivalent of [PLASMA_zgemm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zgemm](#)
[PLASMA_zgemm_Tile](#)
[PLASMA_cgemm_Tile_Async](#)
[PLASMA_dgemm_Tile_Async](#)
[PLASMA_sgemm_Tile_Async](#)

3.9.2.9 `int PLASMA_zgeqrf_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_zgeqrf_Tile_Async - Computes the tile QR factorization of a matrix. Non-blocking equivalent of [PLASMA_zgeqrf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zgeqrf](#)
[PLASMA_zgeqrf_Tile](#)
[PLASMA_cgeqrf_Tile_Async](#)
[PLASMA_dgeqrf_Tile_Async](#)
[PLASMA_sgeqrf_Tile_Async](#)
[PLASMA_zgeqrs_Tile_Async](#)

3.9.2.10 `int PLASMA_zgeqrs_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_zgeqrs_Tile_Async - Computes a minimum-norm solution using the tile QR factorization. Non-blocking equivalent of [PLASMA_zgeqrs_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zgeqrs](#)
[PLASMA_zgeqrs_Tile](#)
[PLASMA_cgeqrs_Tile_Async](#)
[PLASMA_dgeqrs_Tile_Async](#)
[PLASMA_sgeqrs_Tile_Async](#)
[PLASMA_zgeqrf_Tile_Async](#)

3.9.2.11 `int PLASMA_zgesv_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zgesv_Tile_Async` - Solves a system of linear equations using the tile LU factorization. Non-blocking equivalent of `PLASMA_zgesv_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zgesv](#)
[PLASMA_zgesv_Tile](#)
[PLASMA_cgesv_Tile_Async](#)
[PLASMA_dgesv_Tile_Async](#)
[PLASMA_sgesv_Tile_Async](#)
[PLASMA_zcgesv_Tile_Async](#)

3.9.2.12 `int PLASMA_zgetrf_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zgetrf_Tile_Async` - Computes the tile LU factorization of a matrix. Non-blocking equivalent of `PLASMA_zgetrf_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zgetrf](#)

[PLASMA_zgetrf_Tile](#)
[PLASMA_cgetrf_Tile_Async](#)
[PLASMA_dgetrf_Tile_Async](#)
[PLASMA_sgetrf_Tile_Async](#)
[PLASMA_zgetrs_Tile_Async](#)

3.9.2.13 `int PLASMA_zgetrs_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zgetrs_Tile_Async` - Solves a system of linear equations using previously computed LU factorization. Non-blocking equivalent of [PLASMA_zgetrs_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zgetrs](#)
[PLASMA_zgetrs_Tile](#)
[PLASMA_cgetrs_Tile_Async](#)
[PLASMA_dgetrs_Tile_Async](#)
[PLASMA_sgetrs_Tile_Async](#)
[PLASMA_zgetrf_Tile_Async](#)

3.9.2.14 `int PLASMA_zhemm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex64_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zhemm_Tile_Async` - Performs Hermitian matrix multiplication. Non-blocking equivalent of [PLASMA_zhemm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zhemm](#)
[PLASMA_zhemm_Tile](#)
[PLASMA_chemm_Tile_Async](#)
[PLASMA_dhemm_Tile_Async](#)
[PLASMA_shemm_Tile_Async](#)

3.9.2.15 `int PLASMA_zher2k_Tile_Async` (`PLASMA_enum uplo`, `PLASMA_enum trans`, `PLASMA_Complex64_t alpha`, `PLASMA_desc * A`, `PLASMA_desc * B`, `double beta`, `PLASMA_desc * C`, `PLASMA_sequence * sequence`, `PLASMA_request * request`)

`PLASMA_zher2k_Tile_Async` - Performs Hermitian rank-k update. Non-blocking equivalent of `PLASMA_zher2k_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zher2k](#)

[PLASMA_zher2k_Tile](#)

[PLASMA_cher2k_Tile_Async](#)

[PLASMA_dher2k_Tile_Async](#)

[PLASMA_sher2k_Tile_Async](#)

3.9.2.16 `int PLASMA_zherk_Tile_Async` (`PLASMA_enum uplo`, `PLASMA_enum trans`, `double alpha`, `PLASMA_desc * A`, `double beta`, `PLASMA_desc * C`, `PLASMA_sequence * sequence`, `PLASMA_request * request`)

`PLASMA_zherk_Tile_Async` - Performs Hermitian rank-k update. Non-blocking equivalent of `PLASMA_zherk_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zherk](#)

[PLASMA_zherk_Tile](#)

[PLASMA_cherk_Tile_Async](#)

[PLASMA_dherk_Tile_Async](#)

[PLASMA_sherk_Tile_Async](#)

3.9.2.17 `int PLASMA_zlange_Tile_Async` (`PLASMA_enum norm`, `PLASMA_desc * A`, `double * work`, `double * value`, `PLASMA_sequence * sequence`, `PLASMA_request * request`)

`PLASMA_zlange_Tile_Async` - Non-blocking equivalent of `PLASMA_zlange_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zlange](#)
[PLASMA_zlange_Tile](#)
[PLASMA_clange_Tile_Async](#)
[PLASMA_dlange_Tile_Async](#)
[PLASMA_slange_Tile_Async](#)

3.9.2.18 `int PLASMA_zlanhe_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc * A, double * work, double * value, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zlanhe_Tile_Async` - Non-blocking equivalent of [PLASMA_zlanhe_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zlanhe](#)
[PLASMA_zlanhe_Tile](#)
[PLASMA_clanhe_Tile_Async](#)
[PLASMA_dlanhe_Tile_Async](#)
[PLASMA_slanhe_Tile_Async](#)

3.9.2.19 `int PLASMA_zlansy_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc * A, double * work, double * value, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zlansy_Tile_Async` - Non-blocking equivalent of [PLASMA_zlansy_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zlansy](#)
[PLASMA_zlansy_Tile](#)
[PLASMA_clansy_Tile_Async](#)
[PLASMA_dlansy_Tile_Async](#)
[PLASMA_slansy_Tile_Async](#)

3.9.2.20 `int PLASMA_zLapack_to_Tile_Async (PLASMA_Complex64_t * Af77, int LDA, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zLapack_to_Tile_Async` - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of `PLASMA_zLapack_to_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zTile_to_Lapack_Async](#)
[PLASMA_zLapack_to_Tile](#)
[PLASMA_cLapack_to_Tile_Async](#)
[PLASMA_dLapack_to_Tile_Async](#)
[PLASMA_sLapack_to_Tile_Async](#)

3.9.2.21 `int PLASMA_zlauum_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zlauum_Tile_Async` - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A . Non-blocking equivalent of `PLASMA_zlauum_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zlauum](#)
[PLASMA_zlauum_Tile](#)
[PLASMA_clauum_Tile_Async](#)
[PLASMA_dlauum_Tile_Async](#)
[PLASMA_slauum_Tile_Async](#)
[PLASMA_zpotri_Tile_Async](#)

3.9.2.22 `int PLASMA_zplghe_Tile_Async (double bump, PLASMA_desc * A, unsigned long long int seed, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zplghe_Tile_Async` - Generate a random hermitian matrix by tiles. Non-blocking equivalent of `PLASMA_zplghe_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zplghe](#)
[PLASMA_zplghe_Tile](#)
[PLASMA_cplghe_Tile_Async](#)
[PLASMA_dplghe_Tile_Async](#)
[PLASMA_splghe_Tile_Async](#)
[PLASMA_zplghe_Tile_Async](#)
[PLASMA_zplgsy_Tile_Async](#)

3.9.2.23 int PLASMA_zplgsy_Tile_Async (PLASMA_Complex64_t *bump*, PLASMA_desc * *A*, unsigned long long int *seed*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zplgsy_Tile_Async - Generate a random hermitian matrix by tiles. Non-blocking equivalent of [PLASMA_zplgsy_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zplgsy](#)
[PLASMA_zplgsy_Tile](#)
[PLASMA_cplgsy_Tile_Async](#)
[PLASMA_dplgsy_Tile_Async](#)
[PLASMA_splgsy_Tile_Async](#)
[PLASMA_zplgsy_Tile_Async](#)
[PLASMA_zplgsy_Tile_Async](#)

3.9.2.24 int PLASMA_zplrnt_Tile_Async (PLASMA_desc * *A*, unsigned long long int *seed*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zplrnt_Tile_Async - Generate a random matrix by tiles. Non-blocking equivalent of [PLASMA_zplrnt_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zplrnt](#)
[PLASMA_zplrnt_Tile](#)
[PLASMA_cplrnt_Tile_Async](#)
[PLASMA_dplrnt_Tile_Async](#)
[PLASMA_splrnt_Tile_Async](#)
[PLASMA_zplghe_Tile_Async](#)
[PLASMA_zplgsy_Tile_Async](#)

3.9.2.25 `int PLASMA_zposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zposv_Tile_Async` - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Non-blocking equivalent of `PLASMA_zposv_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zposv](#)
[PLASMA_zposv_Tile](#)
[PLASMA_cposv_Tile_Async](#)
[PLASMA_dposv_Tile_Async](#)
[PLASMA_sposv_Tile_Async](#)

3.9.2.26 `int PLASMA_zpotrf_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zpotrf_Tile_Async` - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Non-blocking equivalent of `PLASMA_zpotrf_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zpotrf](#)
[PLASMA_zpotrf_Tile](#)
[PLASMA_cpotrf_Tile_Async](#)
[PLASMA_dpotrf_Tile_Async](#)
[PLASMA_spotrf_Tile_Async](#)
[PLASMA_zpotrs_Tile_Async](#)

3.9.2.27 `int PLASMA_zpotri_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zpotri_Tile_Async` - Computes the inverse of a complex Hermitian positive definite matrix *A* using the Cholesky factorization $A = U^{*}H^{*}U$ or $A = L^{*}L^{*}H^{*}$ computed by `PLASMA_zpotrf`. Non-blocking equivalent of `PLASMA_zpotri_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zpotri](#)
[PLASMA_zpotri_Tile](#)
[PLASMA_cpotri_Tile_Async](#)
[PLASMA_dpotri_Tile_Async](#)
[PLASMA_spotri_Tile_Async](#)
[PLASMA_zpotrf_Tile_Async](#)

3.9.2.28 `int PLASMA_zpotrs_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zpotrs_Tile_Async` - Solves a system of linear equations using previously computed Cholesky factorization. Non-blocking equivalent of `PLASMA_zpotrs_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zpotrs](#)
[PLASMA_zpotrs_Tile](#)
[PLASMA_cpotrs_Tile_Async](#)
[PLASMA_dpotrs_Tile_Async](#)
[PLASMA_spotrs_Tile_Async](#)
[PLASMA_zpotrf_Tile_Async](#)

3.9.2.29 `int PLASMA_zsymm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex64_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zsymm_Tile_Async` - Performs symmetric matrix multiplication. Non-blocking equivalent of `PLASMA_zsymm_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zsymm](#)
[PLASMA_zsymm_Tile](#)
[PLASMA_csymm_Tile_Async](#)
[PLASMA_dsymm_Tile_Async](#)
[PLASMA_ssymm_Tile_Async](#)

3.9.2.30 `int PLASMA_zsyr2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex64_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_zsyr2k_Tile_Async - Performs symmetric rank-k update. Non-blocking equivalent of [PLASMA_zsyr2k_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zsyr2k](#)
[PLASMA_zsyr2k_Tile](#)
[PLASMA_csyr2k_Tile_Async](#)
[PLASMA_dsyr2k_Tile_Async](#)
[PLASMA_ssyr2k_Tile_Async](#)

3.9.2.31 `int PLASMA_zsyrk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_Complex64_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_zsyrk_Tile_Async - Performs rank-k update. Non-blocking equivalent of [PLASMA_zsyrk_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zsyrk](#)

[PLASMA_zsyrk_Tile](#)
[PLASMA_csyrk_Tile_Async](#)
[PLASMA_dsyrk_Tile_Async](#)
[PLASMA_ssyrk_Tile_Async](#)

3.9.2.32 `int PLASMA_zTile_to_Lapack_Async (PLASMA_desc * A, PLASMA_Complex64_t * Af77, int LDA, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_zTile_to_Lapack_Async` - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of [PLASMA_zTile_to_Lapack\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zLapack_to_Tile_Async](#)
[PLASMA_zTile_to_Lapack](#)
[PLASMA_cTile_to_Lapack_Async](#)
[PLASMA_dTile_to_Lapack_Async](#)
[PLASMA_sTile_to_Lapack_Async](#)

3.9.2.33 `int PLASMA_ztrmm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_ztrmm_Tile_Async` - Performs triangular matrix multiplication. Non-blocking equivalent of [PLASMA_ztrmm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ztrmm](#)
[PLASMA_ztrmm_Tile](#)
[PLASMA_ctrmm_Tile_Async](#)
[PLASMA_dtrmm_Tile_Async](#)
[PLASMA_strmm_Tile_Async](#)

3.9.2.34 `int PLASMA_ztrsm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ztrsm_Tile_Async - Computes triangular solve. Non-blocking equivalent of [PLASMA_ztrsm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ztrsm](#)
[PLASMA_ztrsm_Tile](#)
[PLASMA_ctrsm_Tile_Async](#)
[PLASMA_dtrsm_Tile_Async](#)
[PLASMA_strsm_Tile_Async](#)

3.9.2.35 `int PLASMA_ztrsmpi_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ztrsmpi_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. Non-blocking equivalent of [PLASMA_ztrsmpi_Tile\(\)](#). Returns control to the user thread before worker threads finish the computation to allow for pipelined execution of different routines.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ztrsmpi](#)
[PLASMA_ztrsmpi_Tile](#)
[PLASMA_ctrsmpl_Tile_Async](#)
[PLASMA_dtrsmpl_Tile_Async](#)
[PLASMA_strsmpl_Tile_Async](#)
[PLASMA_zgetrf_Tile_Async](#)

3.9.2.36 `int PLASMA_ztrtri_Tile_Async (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ztrtri_Tile_Async - Computes the inverse of a complex upper or lower triangular matrix A. Non-blocking equivalent of [PLASMA_ztrtri_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ztrtri](#)
[PLASMA_ztrtri_Tile](#)
[PLASMA_ctrtri_Tile_Async](#)
[PLASMA_dtrtri_Tile_Async](#)
[PLASMA_strtri_Tile_Async](#)
[PLASMA_zpotri_Tile_Async](#)

3.9.2.37 int PLASMA_zunglq_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)

Non-blocking equivalent of [PLASMA_zunglq_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zunglq](#)
[PLASMA_zunglq_Tile](#)
[PLASMA_cunglq_Tile_Async](#)
[PLASMA_dunglq_Tile_Async](#)
[PLASMA_sunglq_Tile_Async](#)
[PLASMA_zgelqf_Tile_Async](#)

3.9.2.38 int PLASMA_zungqr_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * Q, PLASMA_sequence * sequence, PLASMA_request * request)

Non-blocking equivalent of [PLASMA_zungqr_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zungqr](#)
[PLASMA_zungqr_Tile](#)

[PLASMA_cungqr_Tile_Async](#)
[PLASMA_dungqr_Tile_Async](#)
[PLASMA_sungqr_Tile_Async](#)
[PLASMA_zgeqrf_Tile_Async](#)

3.9.2.39 `int PLASMA_zunmlq_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of [PLASMA_zunmlq_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zunmlq](#)
[PLASMA_zunmlq_Tile](#)
[PLASMA_cunmlq_Tile_Async](#)
[PLASMA_dunmlq_Tile_Async](#)
[PLASMA_sunmlq_Tile_Async](#)
[PLASMA_zgelqf_Tile_Async](#)

3.9.2.40 `int PLASMA_zunmqr_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of [PLASMA_zunmqr_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_zunmqr](#)
[PLASMA_zunmqr_Tile](#)
[PLASMA_cunmqr_Tile_Async](#)
[PLASMA_dunmqr_Tile_Async](#)
[PLASMA_sunmqr_Tile_Async](#)
[PLASMA_zgeqrf_Tile_Async](#)

3.10 Advanced Interface: Asynchronous - Single Complex

Functions/Subroutines

- int [PLASMA_cgelqf_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cgelqs_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cgels_Tile_Async](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cgemm_Tile_Async](#) (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cgeqrf_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cgeqrs_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cgesv_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cgetrf_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cgetrs_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_chemm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cher2k_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cherk_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc *A, float beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_clange_Tile_Async](#) (PLASMA_enum norm, PLASMA_desc *A, float *work, float *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_clanhe_Tile_Async](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work, float *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_clansy_Tile_Async](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work, float *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_clauum_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cplghe_Tile_Async](#) (float bump, PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cplgsy_Tile_Async](#) (PLASMA_Complex32_t bump, PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cplrnt_Tile_Async](#) (PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cposv_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cpotrf_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cpotri_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)

- int [PLASMA_cpotrs_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_csymm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_csytr2k_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_csyrk_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_Complex32_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ctrmm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ctrsm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ctrsmpl_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ctrtri_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cunglq_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cungqr_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cunmlq_Tile_Async](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cunmqr_Tile_Async](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cLapack_to_Tile_Async](#) (PLASMA_Complex32_t *Af77, int LDA, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_cTile_to_Lapack_Async](#) (PLASMA_desc *A, PLASMA_Complex32_t *Af77, int LDA, PLASMA_sequence *sequence, PLASMA_request *request)

3.10.1 Detailed Description

This is the group of single complex functions using the advanced asynchronous interface.

3.10.2 Function/Subroutine Documentation

3.10.2.1 int [PLASMA_cgelqf_Tile_Async](#) (PLASMA_desc * A, PLASMA_desc * T, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

[PLASMA_cgelqf_Tile_Async](#) - Computes the tile LQ factorization of a matrix. Non-blocking equivalent of [PLASMA_cgelqf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cgelqf](#)
[PLASMA_cgelqf_Tile](#)
[PLASMA_cgelqf_Tile_Async](#)
[PLASMA_dgelqf_Tile_Async](#)
[PLASMA_sgelqf_Tile_Async](#)
[PLASMA_cgelqs_Tile_Async](#)

3.10.2.2 int PLASMA_cgelqs_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)

PLASMA_cgelqs_Tile_Async - Computes a minimum-norm solution using previously computed LQ factorization. Non-blocking equivalent of [PLASMA_cgelqs_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cgelqs](#)
[PLASMA_cgelqs_Tile](#)
[PLASMA_cgelqs_Tile_Async](#)
[PLASMA_dgelqs_Tile_Async](#)
[PLASMA_sgelqs_Tile_Async](#)
[PLASMA_cgelqf_Tile_Async](#)

3.10.2.3 int PLASMA_cgels_Tile_Async (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)

PLASMA_cgels_Tile_Async - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Non-blocking equivalent of [PLASMA_cgels_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cgels](#)
[PLASMA_cgels_Tile](#)
[PLASMA_cgels_Tile_Async](#)
[PLASMA_dgels_Tile_Async](#)
[PLASMA_sgels_Tile_Async](#)

3.10.2.4 `int PLASMA_cgemm_Tile_Async (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex32_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_cgemm_Tile_Async - Performs matrix multiplication. Non-blocking equivalent of [PLASMA_cgemm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cgemm](#)
[PLASMA_cgemm_Tile](#)
[PLASMA_cgemm_Tile_Async](#)
[PLASMA_dgemm_Tile_Async](#)
[PLASMA_sgemm_Tile_Async](#)

3.10.2.5 `int PLASMA_cgeqrf_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_cgeqrf_Tile_Async - Computes the tile QR factorization of a matrix. Non-blocking equivalent of [PLASMA_cgeqrf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cgeqrf](#)
[PLASMA_cgeqrf_Tile](#)
[PLASMA_cgeqrf_Tile_Async](#)
[PLASMA_dgeqrf_Tile_Async](#)
[PLASMA_sgeqrf_Tile_Async](#)
[PLASMA_cgeqrs_Tile_Async](#)

3.10.2.6 `int PLASMA_cgeqrs_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_cgeqrs_Tile_Async - Computes a minimum-norm solution using the tile QR factorization. Non-blocking equivalent of [PLASMA_cgeqrs_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cgeqrs](#)
[PLASMA_cgeqrs_Tile](#)
[PLASMA_cgeqrs_Tile_Async](#)
[PLASMA_dgeqrs_Tile_Async](#)
[PLASMA_sgeqrs_Tile_Async](#)
[PLASMA_cgeqrf_Tile_Async](#)

3.10.2.7 int PLASMA_cgesv_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)

PLASMA_cgesv_Tile_Async - Solves a system of linear equations using the tile LU factorization. Non-blocking equivalent of [PLASMA_cgesv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cgesv](#)
[PLASMA_cgesv_Tile](#)
[PLASMA_cgesv_Tile_Async](#)
[PLASMA_dgesv_Tile_Async](#)
[PLASMA_sgesv_Tile_Async](#)
[PLASMA_ccgesv_Tile_Async](#)

3.10.2.8 int PLASMA_cgetrf_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_sequence * sequence, PLASMA_request * request)

PLASMA_cgetrf_Tile_Async - Computes the tile LU factorization of a matrix. Non-blocking equivalent of [PLASMA_cgetrf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cgetrf](#)

[PLASMA_cgetrf_Tile](#)
[PLASMA_cgetrf_Tile_Async](#)
[PLASMA_dgetrf_Tile_Async](#)
[PLASMA_sgetrf_Tile_Async](#)
[PLASMA_cgetrs_Tile_Async](#)

3.10.2.9 `int PLASMA_cgetrs_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_cgetrs_Tile_Async` - Solves a system of linear equations using previously computed LU factorization. Non-blocking equivalent of [PLASMA_cgetrs_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cgetrs](#)
[PLASMA_cgetrs_Tile](#)
[PLASMA_cgetrs_Tile_Async](#)
[PLASMA_dgetrs_Tile_Async](#)
[PLASMA_sgetrs_Tile_Async](#)
[PLASMA_cgetrf_Tile_Async](#)

3.10.2.10 `int PLASMA_chemm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex32_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_chemm_Tile_Async` - Performs Hermitian matrix multiplication. Non-blocking equivalent of [PLASMA_chemm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_chemm](#)
[PLASMA_chemm_Tile](#)
[PLASMA_chemm_Tile_Async](#)
[PLASMA_dhemm_Tile_Async](#)
[PLASMA_shemm_Tile_Async](#)

3.10.2.11 `int PLASMA_cher2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, float beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_cher2k_Tile_Async - Performs Hermitian rank-k update. Non-blocking equivalent of [PLASMA_cher2k_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cher2k](#)

[PLASMA_cher2k_Tile](#)

[PLASMA_cher2k_Tile_Async](#)

[PLASMA_dher2k_Tile_Async](#)

[PLASMA_sher2k_Tile_Async](#)

3.10.2.12 `int PLASMA_cherk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc * A, float beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_cherk_Tile_Async - Performs Hermitian rank-k update. Non-blocking equivalent of [PLASMA_cherk_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cherk](#)

[PLASMA_cherk_Tile](#)

[PLASMA_cherk_Tile_Async](#)

[PLASMA_dherk_Tile_Async](#)

[PLASMA_sherk_Tile_Async](#)

3.10.2.13 `int PLASMA_clange_Tile_Async (PLASMA_enum norm, PLASMA_desc * A, float * work, float * value, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_clange_Tile_Async - Non-blocking equivalent of [PLASMA_clange_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_clange](#)
[PLASMA_clange_Tile](#)
[PLASMA_clange_Tile_Async](#)
[PLASMA_dlange_Tile_Async](#)
[PLASMA_slange_Tile_Async](#)

3.10.2.14 int PLASMA_clanhe_Tile_Async (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc * *A*, float * *work*, float * *value*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_clanhe_Tile_Async - Non-blocking equivalent of [PLASMA_clanhe_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_clanhe](#)
[PLASMA_clanhe_Tile](#)
[PLASMA_clanhe_Tile_Async](#)
[PLASMA_dlanhe_Tile_Async](#)
[PLASMA_slanhe_Tile_Async](#)

3.10.2.15 int PLASMA_clansy_Tile_Async (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc * *A*, float * *work*, float * *value*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_clansy_Tile_Async - Non-blocking equivalent of [PLASMA_clansy_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_clansy](#)
[PLASMA_clansy_Tile](#)
[PLASMA_clansy_Tile_Async](#)
[PLASMA_dlansy_Tile_Async](#)
[PLASMA_slansy_Tile_Async](#)

3.10.2.16 `int PLASMA_cLapack_to_Tile_Async (PLASMA_Complex32_t * A[77], int LDA, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_cLapack_to_Tile_Async` - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of `PLASMA_cLapack_to_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cTile_to_Lapack_Async](#)
[PLASMA_cLapack_to_Tile](#)
[PLASMA_cLapack_to_Tile_Async](#)
[PLASMA_dLapack_to_Tile_Async](#)
[PLASMA_sLapack_to_Tile_Async](#)

3.10.2.17 `int PLASMA_clauum_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_clauum_Tile_Async` - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A . Non-blocking equivalent of `PLASMA_clauum_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_clauum](#)
[PLASMA_clauum_Tile](#)
[PLASMA_clauum_Tile_Async](#)
[PLASMA_dlauum_Tile_Async](#)
[PLASMA_slauum_Tile_Async](#)
[PLASMA_cpotri_Tile_Async](#)

3.10.2.18 `int PLASMA_cplghe_Tile_Async (float bump, PLASMA_desc * A, unsigned long long int seed, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_cplghe_Tile_Async` - Generate a random hermitian matrix by tiles. Non-blocking equivalent of `PLASMA_cplghe_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cplghe](#)
[PLASMA_cplghe_Tile](#)
[PLASMA_cplghe_Tile_Async](#)
[PLASMA_dp1ghe_Tile_Async](#)
[PLASMA_splghe_Tile_Async](#)
[PLASMA_cplghe_Tile_Async](#)
[PLASMA_cplgsy_Tile_Async](#)

3.10.2.19 int PLASMA_cplgsy_Tile_Async (PLASMA_Complex32_t *bump*, PLASMA_desc * *A*, unsigned long long int *seed*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_cplgsy_Tile_Async - Generate a random hermitian matrix by tiles. Non-blocking equivalent of [PLASMA_cplgsy_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cplgsy](#)
[PLASMA_cplgsy_Tile](#)
[PLASMA_cplgsy_Tile_Async](#)
[PLASMA_dp1gsy_Tile_Async](#)
[PLASMA_splgsy_Tile_Async](#)
[PLASMA_cplgsy_Tile_Async](#)
[PLASMA_cplgsy_Tile_Async](#)

3.10.2.20 int PLASMA_cplrnt_Tile_Async (PLASMA_desc * *A*, unsigned long long int *seed*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_cplrnt_Tile_Async - Generate a random matrix by tiles. Non-blocking equivalent of [PLASMA_cplrnt_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cplrint](#)
[PLASMA_cplrint_Tile](#)
[PLASMA_cplrint_Tile_Async](#)
[PLASMA_dpplrint_Tile_Async](#)
[PLASMA_splrint_Tile_Async](#)
[PLASMA_cplghe_Tile_Async](#)
[PLASMA_cplgsy_Tile_Async](#)

3.10.2.21 `int PLASMA_cposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_cposv_Tile_Async - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Non-blocking equivalent of [PLASMA_cposv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
 → *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cposv](#)
[PLASMA_cposv_Tile](#)
[PLASMA_cposv_Tile_Async](#)
[PLASMA_dposv_Tile_Async](#)
[PLASMA_sposv_Tile_Async](#)

3.10.2.22 `int PLASMA_cpotrf_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_cpotrf_Tile_Async - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Non-blocking equivalent of [PLASMA_cpotrf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
 → *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cpotrf](#)
[PLASMA_cpotrf_Tile](#)
[PLASMA_cpotrf_Tile_Async](#)
[PLASMA_dpotrf_Tile_Async](#)
[PLASMA_spotrf_Tile_Async](#)
[PLASMA_cpotrs_Tile_Async](#)

3.10.2.23 `int PLASMA_cpotri_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_cpotri_Tile_Async` - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U \backslash * \backslash * H * U$ or $A = L * L \backslash * \backslash * H$ computed by `PLASMA_cpotrf`. Non-blocking equivalent of `PLASMA_cpotri_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cpotri](#)
[PLASMA_cpotri_Tile](#)
[PLASMA_cpotri_Tile_Async](#)
[PLASMA_dpotri_Tile_Async](#)
[PLASMA_spotri_Tile_Async](#)
[PLASMA_cpotrf_Tile_Async](#)

3.10.2.24 `int PLASMA_cpotrs_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_cpotrs_Tile_Async` - Solves a system of linear equations using previously computed Cholesky factorization. Non-blocking equivalent of `PLASMA_cpotrs_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cpotrs](#)
[PLASMA_cpotrs_Tile](#)
[PLASMA_cpotrs_Tile_Async](#)
[PLASMA_dpotrs_Tile_Async](#)
[PLASMA_spotrs_Tile_Async](#)
[PLASMA_cpotrf_Tile_Async](#)

3.10.2.25 `int PLASMA_csymm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex32_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_csymm_Tile_Async` - Performs symmetric matrix multiplication. Non-blocking equivalent of `PLASMA_csymm_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_csymm](#)
[PLASMA_csymm_Tile](#)
[PLASMA_csymm_Tile_Async](#)
[PLASMA_dsymm_Tile_Async](#)
[PLASMA_ssymm_Tile_Async](#)

3.10.2.26 `int PLASMA_csyr2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_Complex32_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_csyr2k_Tile_Async - Performs symmetric rank-k update. Non-blocking equivalent of [PLASMA_csyr2k_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_csyr2k](#)
[PLASMA_csyr2k_Tile](#)
[PLASMA_csyr2k_Tile_Async](#)
[PLASMA_dsyr2k_Tile_Async](#)
[PLASMA_ssyr2k_Tile_Async](#)

3.10.2.27 `int PLASMA_csyrrk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_Complex32_t beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_csyrrk_Tile_Async - Performs rank-k update. Non-blocking equivalent of [PLASMA_csyrrk_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_csyrrk](#)

[PLASMA_csyk_Tile](#)
[PLASMA_csyk_Tile_Async](#)
[PLASMA_dsyk_Tile_Async](#)
[PLASMA_ssyk_Tile_Async](#)

3.10.2.28 `int PLASMA_cTile_to_Lapack_Async (PLASMA_desc * A, PLASMA_Complex32_t * Af77, int LDA, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_cTile_to_Lapack_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of [PLASMA_cTile_to_Lapack\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cLapack_to_Tile_Async](#)
[PLASMA_cTile_to_Lapack](#)
[PLASMA_cTile_to_Lapack_Async](#)
[PLASMA_dTile_to_Lapack_Async](#)
[PLASMA_sTile_to_Lapack_Async](#)

3.10.2.29 `int PLASMA_ctrmm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ctrmm_Tile_Async - Performs triangular matrix multiplication. Non-blocking equivalent of [PLASMA_ctrmm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ctrmm](#)
[PLASMA_ctrmm_Tile](#)
[PLASMA_ctrmm_Tile_Async](#)
[PLASMA_dtrmm_Tile_Async](#)
[PLASMA_strmm_Tile_Async](#)

3.10.2.30 `int PLASMA_ctrsm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ctrsm_Tile_Async - Computes triangular solve. Non-blocking equivalent of [PLASMA_ctrsm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ctrsm](#)
[PLASMA_ctrsm_Tile](#)
[PLASMA_ctrsm_Tile_Async](#)
[PLASMA_dtrsm_Tile_Async](#)
[PLASMA_strsm_Tile_Async](#)

3.10.2.31 `int PLASMA_ctrsmpl_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ctrsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. Non-blocking equivalent of [PLASMA_ctrsmpl_Tile\(\)](#). Returns control to the user thread before worker threads finish the computation to allow for pipelined execution of different routines.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ctrsmpl](#)
[PLASMA_ctrsmpl_Tile](#)
[PLASMA_ctrsmpl_Tile_Async](#)
[PLASMA_dtrsmpl_Tile_Async](#)
[PLASMA_strsmpl_Tile_Async](#)
[PLASMA_cgetrf_Tile_Async](#)

3.10.2.32 `int PLASMA_ctrtri_Tile_Async (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ctrtri_Tile_Async - Computes the inverse of a complex upper or lower triangular matrix A. Non-blocking equivalent of [PLASMA_ctrtri_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ctrtri](#)
[PLASMA_ctrtri_Tile](#)
[PLASMA_ctrtri_Tile_Async](#)
[PLASMA_dtrtri_Tile_Async](#)
[PLASMA_strtri_Tile_Async](#)
[PLASMA_cpotri_Tile_Async](#)

3.10.233 int PLASMA_cunglq_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)

Non-blocking equivalent of [PLASMA_cunglq_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cunglq](#)
[PLASMA_cunglq_Tile](#)
[PLASMA_cunglq_Tile_Async](#)
[PLASMA_dunglq_Tile_Async](#)
[PLASMA_sunglq_Tile_Async](#)
[PLASMA_cgellqf_Tile_Async](#)

3.10.234 int PLASMA_cungqr_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * Q, PLASMA_sequence * sequence, PLASMA_request * request)

Non-blocking equivalent of [PLASMA_cungqr_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cungqr](#)
[PLASMA_cungqr_Tile](#)

[PLASMA_cungqr_Tile_Async](#)
[PLASMA_dungqr_Tile_Async](#)
[PLASMA_sungqr_Tile_Async](#)
[PLASMA_cgeqrf_Tile_Async](#)

3.10.2.35 `int PLASMA_cunmlq_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of [PLASMA_cunmlq_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cunmlq](#)
[PLASMA_cunmlq_Tile](#)
[PLASMA_cunmlq_Tile_Async](#)
[PLASMA_dunmlq_Tile_Async](#)
[PLASMA_sunmlq_Tile_Async](#)
[PLASMA_cgelqf_Tile_Async](#)

3.10.2.36 `int PLASMA_cunmqr_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of [PLASMA_cunmqr_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_cunmqr](#)
[PLASMA_cunmqr_Tile](#)
[PLASMA_cunmqr_Tile_Async](#)
[PLASMA_dunmqr_Tile_Async](#)
[PLASMA_sunmqr_Tile_Async](#)
[PLASMA_cgeqrf_Tile_Async](#)

3.11 Advanced Interface: Asynchronous - Double Real

Functions/Subroutines

- int [PLASMA_dgelqf_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dgelqs_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dgels_Tile_Async](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dgemm_Tile_Async](#) (PLASMA_enum transA, PLASMA_enum transB, double alpha, PLASMA_desc *A, PLASMA_desc *B, double beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dgeqrf_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dgeqrs_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dgesv_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dgetrf_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dgetrs_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dlange_Tile_Async](#) (PLASMA_enum norm, PLASMA_desc *A, double *work, double *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dlansy_Tile_Async](#) (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, double *work, double *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dlauum_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dorglq_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dorgqr_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dormlq_Tile_Async](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dormqr_Tile_Async](#) (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dplgsy_Tile_Async](#) (double bump, PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dplmt_Tile_Async](#) (PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dposv_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dpotrff_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dpotri_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dpotrs_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)

- int [PLASMA_dsgesv_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_desc *X, int *ITER, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dsposv_Tile_Async](#) (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_desc *X, int *ITER, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dsungesv_Tile_Async](#) (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_desc *X, int *ITER, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dsymm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, double alpha, PLASMA_desc *A, PLASMA_desc *B, double beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dsyr2k_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc *A, PLASMA_desc *B, double beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dsyrk_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc *A, double beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dtrmm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dtrsm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dtrsmpi_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dtrtri_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dLapack_to_Tile_Async](#) (double *Af77, int LDA, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_dTile_to_Lapack_Async](#) (PLASMA_desc *A, double *Af77, int LDA, PLASMA_sequence *sequence, PLASMA_request *request)

3.11.1 Detailed Description

This is the group of double real functions using the advanced asynchronous interface.

3.11.2 Function/Subroutine Documentation

3.11.2.1 int PLASMA_dgelqf_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_sequence * sequence, PLASMA_request * request)

PLASMA_dgelqf_Tile_Async - Computes the tile LQ factorization of a matrix. Non-blocking equivalent of [PLASMA_dgelqf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dgelqf](#)
[PLASMA_dgelqf_Tile](#)
[PLASMA_cgelqf_Tile_Async](#)
[PLASMA_dgelqf_Tile_Async](#)
[PLASMA_sgelqf_Tile_Async](#)
[PLASMA_dgelqs_Tile_Async](#)

3.11.2.2 `int PLASMA_dgelqs_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dgelqs_Tile_Async` - Computes a minimum-norm solution using previously computed LQ factorization. Non-blocking equivalent of `PLASMA_dgelqs_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

\leftarrow *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
 \rightarrow *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dgelqs](#)
[PLASMA_dgelqs_Tile](#)
[PLASMA_cgelqs_Tile_Async](#)
[PLASMA_dgelqs_Tile_Async](#)
[PLASMA_sgelqs_Tile_Async](#)
[PLASMA_dgelqf_Tile_Async](#)

3.11.2.3 `int PLASMA_dgels_Tile_Async (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dgels_Tile_Async` - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Non-blocking equivalent of `PLASMA_dgels_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

\leftarrow *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
 \rightarrow *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dgels](#)
[PLASMA_dgels_Tile](#)
[PLASMA_cgels_Tile_Async](#)
[PLASMA_dgels_Tile_Async](#)
[PLASMA_sgels_Tile_Async](#)

3.11.2.4 `int PLASMA_dgemm_Tile_Async (PLASMA_enum transA, PLASMA_enum transB, double alpha, PLASMA_desc * A, PLASMA_desc * B, double beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dgemm_Tile_Async - Performs matrix multiplication. Non-blocking equivalent of [PLASMA_dgemm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dgemm](#)
[PLASMA_dgemm_Tile](#)
[PLASMA_cgemm_Tile_Async](#)
[PLASMA_dgemm_Tile_Async](#)
[PLASMA_sgemm_Tile_Async](#)

3.11.2.5 `int PLASMA_dgeqrf_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dgeqrf_Tile_Async - Computes the tile QR factorization of a matrix. Non-blocking equivalent of [PLASMA_dgeqrf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dgeqrf](#)
[PLASMA_dgeqrf_Tile](#)
[PLASMA_cgeqrf_Tile_Async](#)
[PLASMA_dgeqrf_Tile_Async](#)
[PLASMA_sgeqrf_Tile_Async](#)
[PLASMA_dgeqrs_Tile_Async](#)

3.11.2.6 `int PLASMA_dgeqrs_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dgeqrs_Tile_Async - Computes a minimum-norm solution using the tile QR factorization. Non-blocking equivalent of [PLASMA_dgeqrs_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dgeqrs](#)
[PLASMA_dgeqrs_Tile](#)
[PLASMA_cgeqrs_Tile_Async](#)
[PLASMA_dgeqrs_Tile_Async](#)
[PLASMA_sgeqrs_Tile_Async](#)
[PLASMA_dgeqrf_Tile_Async](#)

3.11.2.7 `int PLASMA_dgesv_Tile_Async(PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dgesv_Tile_Async` - Solves a system of linear equations using the tile LU factorization. Non-blocking equivalent of [PLASMA_dgesv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dgesv](#)
[PLASMA_dgesv_Tile](#)
[PLASMA_cgesv_Tile_Async](#)
[PLASMA_dgesv_Tile_Async](#)
[PLASMA_sgesv_Tile_Async](#)
[PLASMA_dcgsv_Tile_Async](#)

3.11.2.8 `int PLASMA_dgetrf_Tile_Async(PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dgetrf_Tile_Async` - Computes the tile LU factorization of a matrix. Non-blocking equivalent of [PLASMA_dgetrf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dgetrf](#)

[PLASMA_dgetrf_Tile](#)
[PLASMA_cgetrf_Tile_Async](#)
[PLASMA_dgetrf_Tile_Async](#)
[PLASMA_sgetrf_Tile_Async](#)
[PLASMA_dgetrs_Tile_Async](#)

3.11.2.9 `int PLASMA_dgetrs_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dgetrs_Tile_Async` - Solves a system of linear equations using previously computed LU factorization. Non-blocking equivalent of [PLASMA_dgetrs_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dgetrs](#)
[PLASMA_dgetrs_Tile](#)
[PLASMA_cgetrs_Tile_Async](#)
[PLASMA_dgetrs_Tile_Async](#)
[PLASMA_sgetrs_Tile_Async](#)
[PLASMA_dgetrf_Tile_Async](#)

3.11.2.10 `int PLASMA_dlange_Tile_Async (PLASMA_enum norm, PLASMA_desc * A, double * work, double * value, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dlange_Tile_Async` - Non-blocking equivalent of [PLASMA_dlange_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dlange](#)
[PLASMA_dlange_Tile](#)
[PLASMA_clange_Tile_Async](#)
[PLASMA_dlange_Tile_Async](#)
[PLASMA_slange_Tile_Async](#)

3.11.2.11 `int PLASMA_dlansy_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc * A, double * work, double * value, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dlansy_Tile_Async - Non-blocking equivalent of [PLASMA_dlansy_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dlansy](#)
[PLASMA_dlansy_Tile](#)
[PLASMA_clansy_Tile_Async](#)
[PLASMA_dlansy_Tile_Async](#)
[PLASMA_slansy_Tile_Async](#)

3.11.2.12 `int PLASMA_dLapack_to_Tile_Async (double * Af77, int LDA, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dLapack_to_Tile_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of [PLASMA_dLapack_to_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dTile_to_Lapack_Async](#)
[PLASMA_dLapack_to_Tile](#)
[PLASMA_cLapack_to_Tile_Async](#)
[PLASMA_dLapack_to_Tile_Async](#)
[PLASMA_sLapack_to_Tile_Async](#)

3.11.2.13 `int PLASMA_dlauum_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dlauum_Tile_Async - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A . Non-blocking equivalent of [PLASMA_dlauum_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dlauum](#)
[PLASMA_dlauum_Tile](#)
[PLASMA_clauum_Tile_Async](#)
[PLASMA_dlauum_Tile_Async](#)
[PLASMA_slauum_Tile_Async](#)
[PLASMA_dpotri_Tile_Async](#)

3.11.2.14 int PLASMA_dorglq_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

Non-blocking equivalent of [PLASMA_dorglq_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dorglq](#)
[PLASMA_dorglq_Tile](#)
[PLASMA_cunglq_Tile_Async](#)
[PLASMA_dunglq_Tile_Async](#)
[PLASMA_sunglq_Tile_Async](#)
[PLASMA_dgelqf_Tile_Async](#)

3.11.2.15 int PLASMA_dorgqr_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * Q, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

Non-blocking equivalent of [PLASMA_dorgqr_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dorgqr](#)
[PLASMA_dorgqr_Tile](#)
[PLASMA_cungqr_Tile_Async](#)
[PLASMA_dungqr_Tile_Async](#)
[PLASMA_sungqr_Tile_Async](#)
[PLASMA_dgeqrf_Tile_Async](#)

3.11.2.16 `int PLASMA_dormlq_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of `PLASMA_dormlq_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dormlq](#)
[PLASMA_dormlq_Tile](#)
[PLASMA_cunmlq_Tile_Async](#)
[PLASMA_dunmlq_Tile_Async](#)
[PLASMA_sunmlq_Tile_Async](#)
[PLASMA_dgelqf_Tile_Async](#)

3.11.2.17 `int PLASMA_dormqr_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of `PLASMA_dormqr_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dormqr](#)
[PLASMA_dormqr_Tile](#)
[PLASMA_cunmqr_Tile_Async](#)
[PLASMA_dunmqr_Tile_Async](#)
[PLASMA_sunmqr_Tile_Async](#)
[PLASMA_dgeqrf_Tile_Async](#)

3.11.2.18 `int PLASMA_dplgsy_Tile_Async (double bump, PLASMA_desc * A, unsigned long long int seed, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dplgsy_Tile_Async` - Generate a random hermitian matrix by tiles. Non-blocking equivalent of `PLASMA_dplgsy_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dplgsy](#)
[PLASMA_dplgsy_Tile](#)
[PLASMA_cplgsy_Tile_Async](#)
[PLASMA_dplgsy_Tile_Async](#)
[PLASMA_splgsy_Tile_Async](#)
[PLASMA_dplgsy_Tile_Async](#)
[PLASMA_dplgsy_Tile_Async](#)

3.11.2.19 `int PLASMA_dplrnt_Tile_Async (PLASMA_desc * A, unsigned long long int seed, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dplrnt_Tile_Async` - Generate a random matrix by tiles. Non-blocking equivalent of [PLASMA_dplrnt_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dplrnt](#)
[PLASMA_dplrnt_Tile](#)
[PLASMA_cplrnt_Tile_Async](#)
[PLASMA_dplrnt_Tile_Async](#)
[PLASMA_splrnt_Tile_Async](#)
[PLASMA_dplgsy_Tile_Async](#)
[PLASMA_dplgsy_Tile_Async](#)

3.11.2.20 `int PLASMA_dposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dposv_Tile_Async` - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Non-blocking equivalent of [PLASMA_dposv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dposv](#)
[PLASMA_dposv_Tile](#)
[PLASMA_cposv_Tile_Async](#)
[PLASMA_dposv_Tile_Async](#)
[PLASMA_sposv_Tile_Async](#)

3.11.2.21 `int PLASMA_dpotrf_Tile_Async(PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dpotrf_Tile_Async - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Non-blocking equivalent of [PLASMA_dpotrf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

\leftarrow *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
 \rightarrow *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dpotrf](#)
[PLASMA_dpotrf_Tile](#)
[PLASMA_cpotrf_Tile_Async](#)
[PLASMA_dpotrf_Tile_Async](#)
[PLASMA_spotrf_Tile_Async](#)
[PLASMA_dpotsr_Tile_Async](#)

3.11.2.22 `int PLASMA_dpotri_Tile_Async(PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dpotri_Tile_Async - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U \backslash * \backslash * T * U$ or $A = L * L \backslash * \backslash * T$ computed by [PLASMA_dpotrf](#). Non-blocking equivalent of [PLASMA_dpotri_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

\leftarrow *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
 \rightarrow *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dpotri](#)
[PLASMA_dpotri_Tile](#)
[PLASMA_cpotri_Tile_Async](#)
[PLASMA_dpotri_Tile_Async](#)
[PLASMA_spotri_Tile_Async](#)
[PLASMA_dpotrf_Tile_Async](#)

3.11.2.23 `int PLASMA_dpotrs_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dpotrs_Tile_Async` - Solves a system of linear equations using previously computed Cholesky factorization. Non-blocking equivalent of `PLASMA_dpotrs_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dpotrs](#)
[PLASMA_dpotrs_Tile](#)
[PLASMA_cpotrs_Tile_Async](#)
[PLASMA_dpotrs_Tile_Async](#)
[PLASMA_spotrs_Tile_Async](#)
[PLASMA_dpotrf_Tile_Async](#)

3.11.2.24 `int PLASMA_dsgesv_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_desc * X, int * ITER, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dsgesv_Tile_Async` - Solves a system of linear equations using the tile LU factorization and mixed-precision iterative refinement. Non-blocking equivalent of `PLASMA_dsgesv_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dsgesv](#)
[PLASMA_dsgesv_Tile](#)
[PLASMA_dsgesv_Tile_Async](#)
[PLASMA_dgesv_Tile_Async](#)

3.11.2.25 `int PLASMA_dsposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_desc * X, int * ITER, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dsposv_Tile_Async` - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization and mixed-precision iterative refinement. Non-blocking equivalent of `PLASMA_dsposv_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dsposv](#)
[PLASMA_dsposv_Tile](#)
[PLASMA_dsposv_Tile_Async](#)
[PLASMA_zposv_Tile_Async](#)

3.11.2.26 `int PLASMA_dsungesv_Tile_Async (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_desc * X, int * ITER, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dsungesv_Tile_Async - Solves symmetric linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Non-blocking equivalent of [PLASMA_dsungesv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dsungesv](#)
[PLASMA_dsungesv_Tile](#)
[PLASMA_dsungesv_Tile_Async](#)
[PLASMA_zgels_Tile_Async](#)

3.11.2.27 `int PLASMA_dsymm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, double alpha, PLASMA_desc * A, PLASMA_desc * B, double beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dsymm_Tile_Async - Performs symmetric matrix multiplication. Non-blocking equivalent of [PLASMA_dsymm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dsymm](#)
[PLASMA_dsymm_Tile](#)

[PLASMA_csymm_Tile_Async](#)
[PLASMA_dsymm_Tile_Async](#)
[PLASMA_ssymm_Tile_Async](#)

3.11.2.28 `int PLASMA_dsyr2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc * A, PLASMA_desc * B, double beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dsyr2k_Tile_Async - Performs symmetric rank-k update. Non-blocking equivalent of [PLASMA_dsyr2k_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dsyr2k](#)
[PLASMA_dsyr2k_Tile](#)
[PLASMA_csyr2k_Tile_Async](#)
[PLASMA_dsyr2k_Tile_Async](#)
[PLASMA_ssyr2k_Tile_Async](#)

3.11.2.29 `int PLASMA_dsyrk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc * A, double beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dsyrk_Tile_Async - Performs rank-k update. Non-blocking equivalent of [PLASMA_dsyrk_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dsyrk](#)
[PLASMA_dsyrk_Tile](#)
[PLASMA_csyrk_Tile_Async](#)
[PLASMA_dsyrk_Tile_Async](#)
[PLASMA_ssyrk_Tile_Async](#)

3.11.2.30 `int PLASMA_dTile_to_Lapack_Async (PLASMA_desc * A, double * Af77, int LDA, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_dTile_to_Lapack_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of [PLASMA_dTile_to_Lapack\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dLapack_to_Tile_Async](#)
[PLASMA_dTile_to_Lapack](#)
[PLASMA_cTile_to_Lapack_Async](#)
[PLASMA_dTile_to_Lapack_Async](#)
[PLASMA_sTile_to_Lapack_Async](#)

3.11.2.31 `int PLASMA_dtrmm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dtrmm_Tile_Async` - Performs triangular matrix multiplication. Non-blocking equivalent of `PLASMA_dtrmm_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dtrmm](#)
[PLASMA_dtrmm_Tile](#)
[PLASMA_ctrmm_Tile_Async](#)
[PLASMA_dtrmm_Tile_Async](#)
[PLASMA_strmm_Tile_Async](#)

3.11.2.32 `int PLASMA_dtrsm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dtrsm_Tile_Async` - Computes triangular solve. Non-blocking equivalent of `PLASMA_dtrsm_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dtrsm](#)
[PLASMA_dtrsm_Tile](#)

[PLASMA_ctrsm_Tile_Async](#)
[PLASMA_dtrsm_Tile_Async](#)
[PLASMA_strsm_Tile_Async](#)

3.11.2.33 `int PLASMA_dtrsmpl_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dtrsmpl_Tile` - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. Non-blocking equivalent of `PLASMA_dtrsmpl_Tile()`. Returns control to the user thread before worker threads finish the computation to allow for pipelined execution of different routines.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dtrsmpl](#)
[PLASMA_dtrsmpl_Tile](#)
[PLASMA_ctrsmpl_Tile_Async](#)
[PLASMA_dtrsmpl_Tile_Async](#)
[PLASMA_strsmpl_Tile_Async](#)
[PLASMA_dgetrf_Tile_Async](#)

3.11.2.34 `int PLASMA_dtrtri_Tile_Async (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_dtrtri_Tile_Async` - Computes the inverse of a complex upper or lower triangular matrix A. Non-blocking equivalent of `PLASMA_dtrtri_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_dtrtri](#)
[PLASMA_dtrtri_Tile](#)
[PLASMA_ctrtri_Tile_Async](#)
[PLASMA_dtrtri_Tile_Async](#)
[PLASMA_strtri_Tile_Async](#)
[PLASMA_dpotri_Tile_Async](#)

3.12 Advanced Interface: Asynchronous - Single Real

Functions/Subroutines

- int `PLASMA_sgelqf_Tile_Async` (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sgelqs_Tile_Async` (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sgels_Tile_Async` (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sgemm_Tile_Async` (PLASMA_enum transA, PLASMA_enum transB, float alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sgeqrf_Tile_Async` (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sgeqrs_Tile_Async` (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sgesv_Tile_Async` (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sgetrf_Tile_Async` (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sgetrs_Tile_Async` (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_slange_Tile_Async` (PLASMA_enum norm, PLASMA_desc *A, float *work, float *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_slansy_Tile_Async` (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work, float *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_slauum_Tile_Async` (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sorglq_Tile_Async` (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sorgqr_Tile_Async` (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sormlq_Tile_Async` (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sormqr_Tile_Async` (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_splgsy_Tile_Async` (float bump, PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_splrnt_Tile_Async` (PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_sposv_Tile_Async` (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_spotrf_Tile_Async` (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_spotri_Tile_Async` (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int `PLASMA_spotrs_Tile_Async` (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)

- int [PLASMA_ssyrmm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, float alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ssyr2k_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_ssyrk_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc *A, float beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_strmm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_strsm_Tile_Async](#) (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_strsmpl_Tile_Async](#) (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_strtri_Tile_Async](#) (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_sLapack_to_Tile_Async](#) (float *Af77, int LDA, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int [PLASMA_sTile_to_Lapack_Async](#) (PLASMA_desc *A, float *Af77, int LDA, PLASMA_sequence *sequence, PLASMA_request *request)

3.12.1 Detailed Description

This is the group of single real functions using the advanced asynchronous interface.

3.12.2 Function/Subroutine Documentation

3.12.2.1 int [PLASMA_sgelqf_Tile_Async](#) (PLASMA_desc * A, PLASMA_desc * T, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

[PLASMA_sgelqf_Tile_Async](#) - Computes the tile LQ factorization of a matrix. Non-blocking equivalent of [PLASMA_sgelqf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sgelqf](#)
[PLASMA_sgelqf_Tile](#)
[PLASMA_cgelqf_Tile_Async](#)
[PLASMA_dgelqf_Tile_Async](#)
[PLASMA_sgelqf_Tile_Async](#)
[PLASMA_sgelqs_Tile_Async](#)

3.12.2.2 `int PLASMA_sgelqs_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_sgelqs_Tile_Async` - Computes a minimum-norm solution using previously computed LQ factorization. Non-blocking equivalent of `PLASMA_sgelqs_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sgelqs](#)
[PLASMA_sgelqs_Tile](#)
[PLASMA_cgelqs_Tile_Async](#)
[PLASMA_dgelqs_Tile_Async](#)
[PLASMA_sgelqs_Tile_Async](#)
[PLASMA_sgelqf_Tile_Async](#)

3.12.2.3 `int PLASMA_sgels_Tile_Async (PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_sgels_Tile_Async` - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Non-blocking equivalent of `PLASMA_sgels_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sgels](#)
[PLASMA_sgels_Tile](#)
[PLASMA_cgels_Tile_Async](#)
[PLASMA_dgels_Tile_Async](#)
[PLASMA_sgels_Tile_Async](#)

3.12.2.4 `int PLASMA_sgemm_Tile_Async (PLASMA_enum transA, PLASMA_enum transB, float alpha, PLASMA_desc * A, PLASMA_desc * B, float beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_sgemm_Tile_Async` - Performs matrix multiplication. Non-blocking equivalent of `PLASMA_sgemm_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sgemm](#)
[PLASMA_sgemm_Tile](#)
[PLASMA_cgemm_Tile_Async](#)
[PLASMA_dgemm_Tile_Async](#)
[PLASMA_sgemm_Tile_Async](#)

3.12.2.5 int PLASMA_sgeqrf_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_sequence * sequence, PLASMA_request * request)

PLASMA_sgeqrf_Tile_Async - Computes the tile QR factorization of a matrix. Non-blocking equivalent of [PLASMA_sgeqrf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sgeqrf](#)
[PLASMA_sgeqrf_Tile](#)
[PLASMA_cgeqrf_Tile_Async](#)
[PLASMA_dgeqrf_Tile_Async](#)
[PLASMA_sgeqrf_Tile_Async](#)
[PLASMA_sgeqrs_Tile_Async](#)

3.12.2.6 int PLASMA_sgeqrs_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)

PLASMA_sgeqrs_Tile_Async - Computes a minimum-norm solution using the tile QR factorization. Non-blocking equivalent of [PLASMA_sgeqrs_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sgeqrs](#)

[PLASMA_sgeqrs_Tile](#)
[PLASMA_cgeqrs_Tile_Async](#)
[PLASMA_dgeqrs_Tile_Async](#)
[PLASMA_sgeqrs_Tile_Async](#)
[PLASMA_sgeqrf_Tile_Async](#)

3.12.2.7 `int PLASMA_sgesv_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_sgesv_Tile_Async` - Solves a system of linear equations using the tile LU factorization. Non-blocking equivalent of [PLASMA_sgesv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sgesv](#)
[PLASMA_sgesv_Tile](#)
[PLASMA_cgesv_Tile_Async](#)
[PLASMA_dgesv_Tile_Async](#)
[PLASMA_sgesv_Tile_Async](#)
[PLASMA_scgesv_Tile_Async](#)

3.12.2.8 `int PLASMA_sgetrf_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_sgetrf_Tile_Async` - Computes the tile LU factorization of a matrix. Non-blocking equivalent of [PLASMA_sgetrf_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sgetrf](#)
[PLASMA_sgetrf_Tile](#)
[PLASMA_cgetrf_Tile_Async](#)
[PLASMA_dgetrf_Tile_Async](#)
[PLASMA_sgetrf_Tile_Async](#)
[PLASMA_sgetrs_Tile_Async](#)

3.12.2.9 `int PLASMA_sgetrs_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_sgetrs_Tile_Async` - Solves a system of linear equations using previously computed LU factorization. Non-blocking equivalent of `PLASMA_sgetrs_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sgetrs](#)
[PLASMA_sgetrs_Tile](#)
[PLASMA_cgetrs_Tile_Async](#)
[PLASMA_dgetrs_Tile_Async](#)
[PLASMA_sgetrs_Tile_Async](#)
[PLASMA_sgetrf_Tile_Async](#)

3.12.2.10 `int PLASMA_slange_Tile_Async (PLASMA_enum norm, PLASMA_desc * A, float * work, float * value, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_slange_Tile_Async` - Non-blocking equivalent of `PLASMA_slange_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_slange](#)
[PLASMA_slange_Tile](#)
[PLASMA_clange_Tile_Async](#)
[PLASMA_dlange_Tile_Async](#)
[PLASMA_slange_Tile_Async](#)

3.12.2.11 `int PLASMA_slansy_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc * A, float * work, float * value, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_slansy_Tile_Async` - Non-blocking equivalent of `PLASMA_slansy_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_slansy](#)
[PLASMA_slansy_Tile](#)
[PLASMA_clansy_Tile_Async](#)
[PLASMA_dlansy_Tile_Async](#)
[PLASMA_slansy_Tile_Async](#)

3.12.2.12 int PLASMA_sLapack_to_Tile_Async (float * *Af77*, int *LDA*, PLASMA_desc * *A*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_sLapack_to_Tile_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of [PLASMA_sLapack_to_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sTile_to_Lapack_Async](#)
[PLASMA_sLapack_to_Tile](#)
[PLASMA_cLapack_to_Tile_Async](#)
[PLASMA_dLapack_to_Tile_Async](#)
[PLASMA_sLapack_to_Tile_Async](#)

3.12.2.13 int PLASMA_slauum_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc * *A*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_slauum_Tile_Async - Computes the product $U * U'$ or $L' * L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A . Non-blocking equivalent of [PLASMA_slauum_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_slauum](#)
[PLASMA_slauum_Tile](#)
[PLASMA_clauum_Tile_Async](#)
[PLASMA_dlauum_Tile_Async](#)
[PLASMA_slauum_Tile_Async](#)
[PLASMA_spotri_Tile_Async](#)

3.12.2.14 `int PLASMA_sorglq_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of `PLASMA_sorglq_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sorglq](#)
[PLASMA_sorglq_Tile](#)
[PLASMA_cunglq_Tile_Async](#)
[PLASMA_dunglq_Tile_Async](#)
[PLASMA_sunglq_Tile_Async](#)
[PLASMA_sgelqf_Tile_Async](#)

3.12.2.15 `int PLASMA_sorgqr_Tile_Async (PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * Q, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of `PLASMA_sorgqr_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sorgqr](#)
[PLASMA_sorgqr_Tile](#)
[PLASMA_cungqr_Tile_Async](#)
[PLASMA_dungqr_Tile_Async](#)
[PLASMA_sungqr_Tile_Async](#)
[PLASMA_sgeqrf_Tile_Async](#)

3.12.2.16 `int PLASMA_sormlq_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of `PLASMA_sormlq_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sormlq](#)
[PLASMA_sormlq_Tile](#)
[PLASMA_cunmlq_Tile_Async](#)
[PLASMA_dunmlq_Tile_Async](#)
[PLASMA_sunmlq_Tile_Async](#)
[PLASMA_sgelqf_Tile_Async](#)

3.12.2.17 `int PLASMA_sormqr_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc * A, PLASMA_desc * T, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

Non-blocking equivalent of [PLASMA_sormqr_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sormqr](#)
[PLASMA_sormqr_Tile](#)
[PLASMA_cunmqr_Tile_Async](#)
[PLASMA_dunmqr_Tile_Async](#)
[PLASMA_sunmqr_Tile_Async](#)
[PLASMA_sgeqrf_Tile_Async](#)

3.12.2.18 `int PLASMA_splgsy_Tile_Async (float bump, PLASMA_desc * A, unsigned long long int seed, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_splgsy_Tile_Async` - Generate a random hermitian matrix by tiles. Non-blocking equivalent of [PLASMA_splgsy_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_splgsy](#)
[PLASMA_splgsy_Tile](#)
[PLASMA_cplgsy_Tile_Async](#)
[PLASMA_dplgsy_Tile_Async](#)

[PLASMA_splgsy_Tile_Async](#)
[PLASMA_splgsy_Tile_Async](#)
[PLASMA_splgsy_Tile_Async](#)

3.12.2.19 `int PLASMA_splrnt_Tile_Async (PLASMA_desc * A, unsigned long long int seed, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_splrnt_Tile_Async` - Generate a random matrix by tiles. Non-blocking equivalent of [PLASMA_splrnt_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_splrnt](#)
[PLASMA_splrnt_Tile](#)
[PLASMA_cplrnt_Tile_Async](#)
[PLASMA_dplrnt_Tile_Async](#)
[PLASMA_splrnt_Tile_Async](#)
[PLASMA_splgsy_Tile_Async](#)
[PLASMA_splgsy_Tile_Async](#)

3.12.2.20 `int PLASMA_sposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_sposv_Tile_Async` - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Non-blocking equivalent of [PLASMA_sposv_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sposv](#)
[PLASMA_sposv_Tile](#)
[PLASMA_cposv_Tile_Async](#)
[PLASMA_dposv_Tile_Async](#)
[PLASMA_sposv_Tile_Async](#)

3.12.2.21 `int PLASMA_spotrf_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_spotrf_Tile_Async` - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Non-blocking equivalent of `PLASMA_spotrf_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_spotrf](#)
[PLASMA_spotrf_Tile](#)
[PLASMA_cpotrf_Tile_Async](#)
[PLASMA_dpotrf_Tile_Async](#)
[PLASMA_spotrf_Tile_Async](#)
[PLASMA_spotrs_Tile_Async](#)

3.12.2.22 `int PLASMA_spotri_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_spotri_Tile_Async` - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U^*U$ or $A = LL^*$ computed by `PLASMA_spotrf`. Non-blocking equivalent of `PLASMA_spotri_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_spotri](#)
[PLASMA_spotri_Tile](#)
[PLASMA_cpotri_Tile_Async](#)
[PLASMA_dpotri_Tile_Async](#)
[PLASMA_spotri_Tile_Async](#)
[PLASMA_spotrf_Tile_Async](#)

3.12.2.23 `int PLASMA_spotrs_Tile_Async (PLASMA_enum uplo, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_spotrs_Tile_Async` - Solves a system of linear equations using previously computed Cholesky factorization. Non-blocking equivalent of `PLASMA_spotrs_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_spotrs](#)
[PLASMA_spotrs_Tile](#)
[PLASMA_cpotrs_Tile_Async](#)
[PLASMA_dpotrs_Tile_Async](#)
[PLASMA_spotrs_Tile_Async](#)
[PLASMA_spotrf_Tile_Async](#)

3.12.2.24 `int PLASMA_ssymm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, float alpha, PLASMA_desc * A, PLASMA_desc * B, float beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ssymm_Tile_Async - Performs symmetric matrix multiplication. Non-blocking equivalent of [PLASMA_ssymm_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ssymm](#)
[PLASMA_ssymm_Tile](#)
[PLASMA_csymm_Tile_Async](#)
[PLASMA_dsymm_Tile_Async](#)
[PLASMA_ssymm_Tile_Async](#)

3.12.2.25 `int PLASMA_ssyr2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc * A, PLASMA_desc * B, float beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ssyr2k_Tile_Async - Performs symmetric rank-k update. Non-blocking equivalent of [PLASMA_ssyr2k_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ssyk2k](#)
[PLASMA_ssyk2k_Tile](#)
[PLASMA_csyr2k_Tile_Async](#)
[PLASMA_dsyr2k_Tile_Async](#)
[PLASMA_ssyk2k_Tile_Async](#)

3.12.2.26 `int PLASMA_ssyk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc * A, float beta, PLASMA_desc * C, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_ssyk_Tile_Async - Performs rank-k update. Non-blocking equivalent of [PLASMA_ssyk_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_ssyk](#)
[PLASMA_ssyk_Tile](#)
[PLASMA_csyrk_Tile_Async](#)
[PLASMA_dsyrk_Tile_Async](#)
[PLASMA_ssyk_Tile_Async](#)

3.12.2.27 `int PLASMA_sTile_to_Lapack_Async (PLASMA_desc * A, float * Af77, int LDA, PLASMA_sequence * sequence, PLASMA_request * request)`

PLASMA_sTile_to_Lapack_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of [PLASMA_sTile_to_Lapack\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_sLapack_to_Tile_Async](#)
[PLASMA_sTile_to_Lapack](#)
[PLASMA_cTile_to_Lapack_Async](#)
[PLASMA_dTile_to_Lapack_Async](#)
[PLASMA_sTile_to_Lapack_Async](#)

3.12.2.28 `int PLASMA_strmm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_strmm_Tile_Async` - Performs triangular matrix multiplication. Non-blocking equivalent of `PLASMA_strmm_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_strmm](#)
[PLASMA_strmm_Tile](#)
[PLASMA_ctrmm_Tile_Async](#)
[PLASMA_dtrmm_Tile_Async](#)
[PLASMA_strmm_Tile_Async](#)

3.12.2.29 `int PLASMA_strsm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc * A, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_strsm_Tile_Async` - Computes triangular solve. Non-blocking equivalent of `PLASMA_strsm_Tile()`. May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_strsm](#)
[PLASMA_strsm_Tile](#)
[PLASMA_ctrsm_Tile_Async](#)
[PLASMA_dtrsm_Tile_Async](#)
[PLASMA_strsm_Tile_Async](#)

3.12.2.30 `int PLASMA_strsmpl_Tile_Async (PLASMA_desc * A, PLASMA_desc * L, int * IPIV, PLASMA_desc * B, PLASMA_sequence * sequence, PLASMA_request * request)`

`PLASMA_strsmpl_Tile` - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. Non-blocking equivalent of `PLASMA_strsmpl_Tile()`. Returns control to the user thread before worker threads finish the computation to allow for pipelined execution of different routines.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_strsmpl](#)
[PLASMA_strsmpl_Tile](#)
[PLASMA_ctrsmpl_Tile_Async](#)
[PLASMA_dtrsmpl_Tile_Async](#)
[PLASMA_strsmpl_Tile_Async](#)
[PLASMA_sgetrf_Tile_Async](#)

3.12.2.31 int PLASMA_strtri_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc * *A*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_strtri_Tile_Async - Computes the inverse of a complex upper or lower triangular matrix *A*. Non-blocking equivalent of [PLASMA_strtri_Tile\(\)](#). May return before the computation is finished. Allows for pipelining of operations at runtime.

Parameters

- ← *sequence* Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).
- *request* Identifies this function call (for exception handling purposes).

See also

[PLASMA_strtri](#)
[PLASMA_strtri_Tile](#)
[PLASMA_ctrtri_Tile_Async](#)
[PLASMA_dtrtri_Tile_Async](#)
[PLASMA_strtri_Tile_Async](#)
[PLASMA_spotri_Tile_Async](#)

Chapter 4

Data Type Documentation

4.1 plasma_context_map_struct Type Reference

```
#include <context.h>
```

Data Fields

- [pthread_t](#) `thread_id`
- [plasma_context_t](#)* `context`

4.1.1 Detailed Description

Threads contexts map

4.2 plasma_context_struct Type Reference

```
#include <context.h>
```

Data Fields

- PLASMA_bool **initialized**
- int **world_size**
- int **group_size**
- int **thread_bind** [CONTEXT_THREADS_MAX]
- int **thread_rank** [CONTEXT_THREADS_MAX]
- pthread_attr_t **thread_attr**
- pthread_t **thread_id** [CONTEXT_THREADS_MAX]
- pthread_mutex_t **action_mutex**
- pthread_cond_t **action_condt**
- volatile PLASMA_enum **action**
- void(* **parallel_func_ptr**)(struct [plasma_context_struct](#) *)
- unsigned char **args_buff** [ARGS_BUFF_SIZE]
- PLASMA_bool **errors_enabled**
- PLASMA_bool **warnings_enabled**
- PLASMA_bool **autotuning_enabled**
- PLASMA_bool **dynamic_section**
- PLASMA_enum **scheduling**
- PLASMA_enum **householder**
- PLASMA_enum **translation**
- int **nb**
- int **ib**
- int **nbnbsize**
- int **ibnbsize**
- int **info**
- int **iinfo**
- int **rhblock**
- int volatile **barrier_id**
- int volatile **barrier_nblocked_thrds**
- pthread_mutex_t **barrier_synclock**
- pthread_cond_t **barrier_syncond**
- int **ss_id**
- volatile int **ss_abort**
- volatile int * **ss_progress**
- Quark * **quark**

4.2.1 Detailed Description

PLASMA context

4.3 primedec Type Reference

Data Fields

- int **p**
- int **e**
- int **pe**

4.4 pthread_cond_s Type Reference

Data Fields

- HANDLE **hSem**
- HANDLE **hEvt**
- CRITICAL_SECTION **cs**
- int **waitCount**

4.5 pthread_s Type Reference

Data Fields

- HANDLE **hThread**
- unsigned int **uThId**

Index

- Advanced Interface: Asynchronous - Double Complex, [215](#)
- Advanced Interface: Asynchronous - Double Real, [251](#)
- Advanced Interface: Asynchronous - Single Complex, [234](#)
- Advanced Interface: Asynchronous - Single Real, [267](#)
- Advanced Interface: Synchronous - Double Complex, [125](#)
- Advanced Interface: Synchronous - Double Real, [174](#)
- Advanced Interface: Synchronous - Single Complex, [151](#)
- Advanced Interface: Synchronous - Single Real, [196](#)
- double
 - PLASMA_dgelqf, [72](#)
 - PLASMA_dgelqs, [72](#)
 - PLASMA_dgels, [73](#)
 - PLASMA_dgemm, [74](#)
 - PLASMA_dgeqrf, [75](#)
 - PLASMA_dgeqrs, [76](#)
 - PLASMA_dgesv, [76](#)
 - PLASMA_dgetrf, [77](#)
 - PLASMA_dgetrs, [78](#)
 - PLASMA_dlange, [79](#)
 - PLASMA_dlansy, [79](#)
 - PLASMA_dLapack_to_Tile, [80](#)
 - PLASMA_dlauum, [81](#)
 - PLASMA_dorglq, [81](#)
 - PLASMA_dorgqr, [82](#)
 - PLASMA_dormlq, [83](#)
 - PLASMA_dormqr, [84](#)
 - PLASMA_dplgsy, [84](#)
 - PLASMA_dplrnt, [85](#)
 - PLASMA_dposv, [86](#)
 - PLASMA_dpotrf, [87](#)
 - PLASMA_dpotri, [87](#)
 - PLASMA_dpotrs, [88](#)
 - PLASMA_dsgesv, [89](#)
 - PLASMA_dsposv, [90](#)
 - PLASMA_dsungesv, [91](#)
 - PLASMA_dsymm, [93](#)
 - PLASMA_dsyr2k, [94](#)
 - PLASMA_dsyrk, [95](#)
 - PLASMA_dTile_to_Lapack, [96](#)
 - PLASMA_dtrmm, [96](#)
 - PLASMA_dtrsm, [97](#)
 - PLASMA_dtrsmpl, [98](#)
 - PLASMA_dtrtri, [99](#)
- double_Tile
 - PLASMA_dgelqf_Tile, [175](#)
 - PLASMA_dgelqs_Tile, [175](#)
 - PLASMA_dgels_Tile, [176](#)
 - PLASMA_dgemm_Tile, [176](#)
 - PLASMA_dgeqrf_Tile, [177](#)
 - PLASMA_dgeqrs_Tile, [178](#)
 - PLASMA_dgesv_Tile, [178](#)
 - PLASMA_dgetrf_Tile, [179](#)
 - PLASMA_dgetrs_Tile, [179](#)
 - PLASMA_dlange_Tile, [180](#)
 - PLASMA_dlansy_Tile, [181](#)
 - PLASMA_dlauum_Tile, [181](#)
 - PLASMA_dorglq_Tile, [182](#)
 - PLASMA_dorgqr_Tile, [182](#)
 - PLASMA_dormlq_Tile, [183](#)
 - PLASMA_dormqr_Tile, [183](#)
 - PLASMA_dplgsy_Tile, [184](#)
 - PLASMA_dplrnt_Tile, [184](#)
 - PLASMA_dposv_Tile, [185](#)
 - PLASMA_dpotrf_Tile, [186](#)
 - PLASMA_dpotri_Tile, [186](#)
 - PLASMA_dpotrs_Tile, [187](#)
 - PLASMA_dsgesv_Tile, [187](#)
 - PLASMA_dsposv_Tile, [188](#)
 - PLASMA_dsungesv_Tile, [189](#)
 - PLASMA_dsymm_Tile, [190](#)
 - PLASMA_dsyr2k_Tile, [190](#)
 - PLASMA_dsyrk_Tile, [191](#)
 - PLASMA_dtrmm_Tile, [192](#)
 - PLASMA_dtrsm_Tile, [193](#)
 - PLASMA_dtrsmpl_Tile, [193](#)
 - PLASMA_dtrtri_Tile, [194](#)
- double_Tile_Async
 - PLASMA_dgelqf_Tile_Async, [252](#)
 - PLASMA_dgelqs_Tile_Async, [253](#)
 - PLASMA_dgels_Tile_Async, [253](#)
 - PLASMA_dgemm_Tile_Async, [253](#)

- PLASMA_dgeqrf_Tile_Async, 254
 - PLASMA_dgeqrs_Tile_Async, 254
 - PLASMA_dgesv_Tile_Async, 255
 - PLASMA_dgetrf_Tile_Async, 255
 - PLASMA_dgetrs_Tile_Async, 256
 - PLASMA_dlange_Tile_Async, 256
 - PLASMA_dlansy_Tile_Async, 256
 - PLASMA_dLapack_to_Tile_Async, 257
 - PLASMA_dlauum_Tile_Async, 257
 - PLASMA_dorglq_Tile_Async, 258
 - PLASMA_dorgqr_Tile_Async, 258
 - PLASMA_dormlq_Tile_Async, 258
 - PLASMA_dormqr_Tile_Async, 259
 - PLASMA_dplgsy_Tile_Async, 259
 - PLASMA_dplrnt_Tile_Async, 260
 - PLASMA_dposv_Tile_Async, 260
 - PLASMA_dpotrf_Tile_Async, 261
 - PLASMA_dpotri_Tile_Async, 261
 - PLASMA_dpotrs_Tile_Async, 261
 - PLASMA_dsgesv_Tile_Async, 262
 - PLASMA_dsposv_Tile_Async, 262
 - PLASMA_dsungesv_Tile_Async, 263
 - PLASMA_dsymm_Tile_Async, 263
 - PLASMA_dsyr2k_Tile_Async, 264
 - PLASMA_dsyrk_Tile_Async, 264
 - PLASMA_dTile_to_Lapack_Async, 264
 - PLASMA_dtrmm_Tile_Async, 265
 - PLASMA_dtrsm_Tile_Async, 265
 - PLASMA_dtrsmpl_Tile_Async, 266
 - PLASMA_dtrtri_Tile_Async, 266
- float
- PLASMA_sgelqf, 101
 - PLASMA_sgelqs, 101
 - PLASMA_sgels, 102
 - PLASMA_sgemm, 103
 - PLASMA_sgeqrf, 104
 - PLASMA_sgeqrs, 104
 - PLASMA_sgesv, 105
 - PLASMA_sgetrf, 106
 - PLASMA_sgetrs, 107
 - PLASMA_slange, 107
 - PLASMA_slansy, 108
 - PLASMA_sLapack_to_Tile, 109
 - PLASMA_slauum, 109
 - PLASMA_sorglq, 110
 - PLASMA_sorgqr, 111
 - PLASMA_sormlq, 111
 - PLASMA_sormqr, 112
 - PLASMA_splgsy, 113
 - PLASMA_splrnt, 114
 - PLASMA_sposv, 114
 - PLASMA_spotrf, 115
 - PLASMA_spotri, 116
 - PLASMA_spotrs, 117
 - PLASMA_ssymm, 117
 - PLASMA_ssyr2k, 118
 - PLASMA_ssyrik, 119
 - PLASMA_sTile_to_Lapack, 120
 - PLASMA_strmm, 121
 - PLASMA_strsm, 122
 - PLASMA_strsmpl, 123
 - PLASMA_strtri, 123
- float_Tile
- PLASMA_sgelqf_Tile, 197
 - PLASMA_sgelqs_Tile, 197
 - PLASMA_sgels_Tile, 198
 - PLASMA_sgemm_Tile, 198
 - PLASMA_sgeqrf_Tile, 199
 - PLASMA_sgeqrs_Tile, 199
 - PLASMA_sgesv_Tile, 200
 - PLASMA_sgetrf_Tile, 201
 - PLASMA_sgetrs_Tile, 201
 - PLASMA_slange_Tile, 202
 - PLASMA_slansy_Tile, 202
 - PLASMA_slauum_Tile, 203
 - PLASMA_sorglq_Tile, 204
 - PLASMA_sorgqr_Tile, 204
 - PLASMA_sormlq_Tile, 205
 - PLASMA_sormqr_Tile, 205
 - PLASMA_splgsy_Tile, 206
 - PLASMA_splrnt_Tile, 206
 - PLASMA_sposv_Tile, 207
 - PLASMA_spotrf_Tile, 208
 - PLASMA_spotri_Tile, 208
 - PLASMA_spotrs_Tile, 209
 - PLASMA_ssymm_Tile, 209
 - PLASMA_ssyr2k_Tile, 210
 - PLASMA_ssyrik_Tile, 211
 - PLASMA_strmm_Tile, 211
 - PLASMA_strsm_Tile, 212
 - PLASMA_strsmpl_Tile, 213
 - PLASMA_strtri_Tile, 214
- float_Tile_Async
- PLASMA_sgelqf_Tile_Async, 268
 - PLASMA_sgelqs_Tile_Async, 268
 - PLASMA_sgels_Tile_Async, 269
 - PLASMA_sgemm_Tile_Async, 269
 - PLASMA_sgeqrf_Tile_Async, 270
 - PLASMA_sgeqrs_Tile_Async, 270
 - PLASMA_sgesv_Tile_Async, 271
 - PLASMA_sgetrf_Tile_Async, 271
 - PLASMA_sgetrs_Tile_Async, 271
 - PLASMA_slange_Tile_Async, 272
 - PLASMA_slansy_Tile_Async, 272
 - PLASMA_sLapack_to_Tile_Async, 273
 - PLASMA_slauum_Tile_Async, 273
 - PLASMA_sorglq_Tile_Async, 273

- PLASMA_sorgqr_Tile_Async, 274
- PLASMA_sormlq_Tile_Async, 274
- PLASMA_sormqr_Tile_Async, 275
- PLASMA_splgsy_Tile_Async, 275
- PLASMA_splrnt_Tile_Async, 276
- PLASMA_sposv_Tile_Async, 276
- PLASMA_spotrf_Tile_Async, 276
- PLASMA_spotri_Tile_Async, 277
- PLASMA_spotrs_Tile_Async, 277
- PLASMA_ssymm_Tile_Async, 278
- PLASMA_ssyr2k_Tile_Async, 278
- PLASMA_ssyrk_Tile_Async, 279
- PLASMA_sTile_to_Lapack_Async, 279
- PLASMA_strmm_Tile_Async, 279
- PLASMA_strsm_Tile_Async, 280
- PLASMA_strsmpl_Tile_Async, 280
- PLASMA_strtri_Tile_Async, 281

- PLASMA_cgelqf
 - PLASMA_Complex32_t, 42
- PLASMA_cgelqf_Tile
 - PLASMA_Complex32_t_Tile, 152
- PLASMA_cgelqf_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 235
- PLASMA_cgelqs
 - PLASMA_Complex32_t, 43
- PLASMA_cgelqs_Tile
 - PLASMA_Complex32_t_Tile, 152
- PLASMA_cgelqs_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 236
- PLASMA_cgels
 - PLASMA_Complex32_t, 44
- PLASMA_cgels_Tile
 - PLASMA_Complex32_t_Tile, 153
- PLASMA_cgels_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 236
- PLASMA_cgemm
 - PLASMA_Complex32_t, 45
- PLASMA_cgemm_Tile
 - PLASMA_Complex32_t_Tile, 153
- PLASMA_cgemm_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 236
- PLASMA_cgeqrf
 - PLASMA_Complex32_t, 46
- PLASMA_cgeqrf_Tile
 - PLASMA_Complex32_t_Tile, 154
- PLASMA_cgeqrf_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 237
- PLASMA_cgeqrs
 - PLASMA_Complex32_t, 46
- PLASMA_cgeqrs_Tile
 - PLASMA_Complex32_t_Tile, 155
- PLASMA_cgeqrs_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 237

- PLASMA_cgesv
 - PLASMA_Complex32_t, 47
- PLASMA_cgesv_Tile
 - PLASMA_Complex32_t_Tile, 155
- PLASMA_cgesv_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 238
- PLASMA_cgetrf
 - PLASMA_Complex32_t, 48
- PLASMA_cgetrf_Tile
 - PLASMA_Complex32_t_Tile, 156
- PLASMA_cgetrf_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 238
- PLASMA_cgetrs
 - PLASMA_Complex32_t, 48
- PLASMA_cgetrs_Tile
 - PLASMA_Complex32_t_Tile, 157
- PLASMA_cgetrs_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 239
- PLASMA_chemm
 - PLASMA_Complex32_t, 49
- PLASMA_chemm_Tile
 - PLASMA_Complex32_t_Tile, 157
- PLASMA_chemm_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 239
- PLASMA_cher2k
 - PLASMA_Complex32_t, 50
- PLASMA_cher2k_Tile
 - PLASMA_Complex32_t_Tile, 158
- PLASMA_cher2k_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 239
- PLASMA_cherk
 - PLASMA_Complex32_t, 51
- PLASMA_cherk_Tile
 - PLASMA_Complex32_t_Tile, 159
- PLASMA_cherk_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 240
- PLASMA_clange
 - PLASMA_Complex32_t, 52
- PLASMA_clange_Tile
 - PLASMA_Complex32_t_Tile, 159
- PLASMA_clange_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 240
- PLASMA_clanhe
 - PLASMA_Complex32_t, 53
- PLASMA_clanhe_Tile
 - PLASMA_Complex32_t_Tile, 160
- PLASMA_clanhe_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 241
- PLASMA_clansy
 - PLASMA_Complex32_t, 54
- PLASMA_clansy_Tile
 - PLASMA_Complex32_t_Tile, 160
- PLASMA_clansy_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 241

- PLASMA_cLapack_to_Tile
 - PLASMA_Complex32_t, 55
- PLASMA_cLapack_to_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 241
- PLASMA_clauum
 - PLASMA_Complex32_t, 55
- PLASMA_clauum_Tile
 - PLASMA_Complex32_t_Tile, 161
- PLASMA_clauum_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 242
- PLASMA_Complex32_t
 - PLASMA_cgelsqf, 42
 - PLASMA_cgelsqs, 43
 - PLASMA_cgels, 44
 - PLASMA_cgemm, 45
 - PLASMA_cgeqrf, 46
 - PLASMA_cgeqrs, 46
 - PLASMA_cgesv, 47
 - PLASMA_cgetrf, 48
 - PLASMA_cgetrs, 48
 - PLASMA_chemm, 49
 - PLASMA_cher2k, 50
 - PLASMA_cherk, 51
 - PLASMA_clange, 52
 - PLASMA_clanhe, 53
 - PLASMA_clansy, 54
 - PLASMA_cLapack_to_Tile, 55
 - PLASMA_clauum, 55
 - PLASMA_cplghe, 56
 - PLASMA_cplgsy, 56
 - PLASMA_cplrnt, 57
 - PLASMA_cposv, 57
 - PLASMA_cpotrf, 58
 - PLASMA_cpotri, 59
 - PLASMA_cpotrs, 60
 - PLASMA_csymm, 61
 - PLASMA_csyr2k, 62
 - PLASMA_csyrrk, 63
 - PLASMA_cTile_to_Lapack, 64
 - PLASMA_ctrmm, 64
 - PLASMA_ctrsm, 65
 - PLASMA_ctrsmpl, 66
 - PLASMA_ctrtri, 67
 - PLASMA_cunglq, 67
 - PLASMA_cungqr, 68
 - PLASMA_cunmlq, 69
 - PLASMA_cunmqr, 70
- PLASMA_Complex32_t_Tile
 - PLASMA_cgelsqf_Tile, 152
 - PLASMA_cgelsqs_Tile, 152
 - PLASMA_cgels_Tile, 153
 - PLASMA_cgemm_Tile, 153
 - PLASMA_cgeqrf_Tile, 154
 - PLASMA_cgeqrs_Tile, 155
 - PLASMA_cgesv_Tile, 155
 - PLASMA_cgetrf_Tile, 156
 - PLASMA_cgetrs_Tile, 157
 - PLASMA_chemm_Tile, 157
 - PLASMA_cher2k_Tile, 158
 - PLASMA_cherk_Tile, 159
 - PLASMA_clange_Tile, 159
 - PLASMA_clanhe_Tile, 160
 - PLASMA_clansy_Tile, 160
 - PLASMA_clauum_Tile, 161
 - PLASMA_cplghe_Tile, 161
 - PLASMA_cplgsy_Tile, 162
 - PLASMA_cplrnt_Tile, 162
 - PLASMA_cposv_Tile, 163
 - PLASMA_cpotrf_Tile, 164
 - PLASMA_cpotri_Tile, 164
 - PLASMA_cpotrs_Tile, 165
 - PLASMA_csymm_Tile, 165
 - PLASMA_csyr2k_Tile, 166
 - PLASMA_csyrrk_Tile, 167
 - PLASMA_ctrmm_Tile, 168
 - PLASMA_ctrsm_Tile, 168
 - PLASMA_ctrsmpl_Tile, 169
 - PLASMA_ctrtri_Tile, 170
 - PLASMA_cunglq_Tile, 170
 - PLASMA_cungqr_Tile, 171
 - PLASMA_cunmlq_Tile, 171
 - PLASMA_cunmqr_Tile, 172
- PLASMA_Complex32_t_Tile_Async
 - PLASMA_cgelsqf_Tile_Async, 235
 - PLASMA_cgelsqs_Tile_Async, 236
 - PLASMA_cgels_Tile_Async, 236
 - PLASMA_cgemm_Tile_Async, 236
 - PLASMA_cgeqrf_Tile_Async, 237
 - PLASMA_cgeqrs_Tile_Async, 237
 - PLASMA_cgesv_Tile_Async, 238
 - PLASMA_cgetrf_Tile_Async, 238
 - PLASMA_cgetrs_Tile_Async, 239
 - PLASMA_chemm_Tile_Async, 239
 - PLASMA_cher2k_Tile_Async, 239
 - PLASMA_cherk_Tile_Async, 240
 - PLASMA_clange_Tile_Async, 240
 - PLASMA_clanhe_Tile_Async, 241
 - PLASMA_clansy_Tile_Async, 241
 - PLASMA_cLapack_to_Tile_Async, 241
 - PLASMA_clauum_Tile_Async, 242
 - PLASMA_cplghe_Tile_Async, 242
 - PLASMA_cplgsy_Tile_Async, 243
 - PLASMA_cplrnt_Tile_Async, 243
 - PLASMA_cposv_Tile_Async, 244
 - PLASMA_cpotrf_Tile_Async, 244
 - PLASMA_cpotri_Tile_Async, 244
 - PLASMA_cpotrs_Tile_Async, 245
 - PLASMA_csymm_Tile_Async, 245

- PLASMA_csyr2k_Tile_Async, 246
- PLASMA_csyrk_Tile_Async, 246
- PLASMA_cTile_to_Lapack_Async, 247
- PLASMA_ctrmm_Tile_Async, 247
- PLASMA_ctrsm_Tile_Async, 247
- PLASMA_ctrsmpl_Tile_Async, 248
- PLASMA_ctrtri_Tile_Async, 248
- PLASMA_cunglq_Tile_Async, 249
- PLASMA_cungqr_Tile_Async, 249
- PLASMA_cunmlq_Tile_Async, 250
- PLASMA_cunmqr_Tile_Async, 250
- PLASMA_Complex64_t
 - PLASMA_zcgels, 7
 - PLASMA_zcgesv, 8
 - PLASMA_zcposv, 9
 - PLASMA_zcungesv, 11
 - PLASMA_zgelqf, 12
 - PLASMA_zgelqs, 13
 - PLASMA_zgels, 13
 - PLASMA_zgemm, 14
 - PLASMA_zgeqrf, 15
 - PLASMA_zgeqrs, 16
 - PLASMA_zgesv, 17
 - PLASMA_zgetrf, 18
 - PLASMA_zgetrs, 18
 - PLASMA_zhemm, 19
 - PLASMA_zher2k, 20
 - PLASMA_zherk, 21
 - PLASMA_zlange, 22
 - PLASMA_zlanhe, 23
 - PLASMA_zlansy, 24
 - PLASMA_zLapack_to_Tile, 25
 - PLASMA_zlauum, 25
 - PLASMA_zplghe, 26
 - PLASMA_zplgsy, 26
 - PLASMA_zplrnt, 27
 - PLASMA_zposv, 27
 - PLASMA_zpotrf, 28
 - PLASMA_zpotri, 29
 - PLASMA_zpotrs, 30
 - PLASMA_zsymm, 31
 - PLASMA_zsyr2k, 32
 - PLASMA_zsyrk, 33
 - PLASMA_zTile_to_Lapack, 34
 - PLASMA_ztrmm, 34
 - PLASMA_ztrsm, 35
 - PLASMA_ztrsmpl, 36
 - PLASMA_ztrtri, 37
 - PLASMA_zunglq, 37
 - PLASMA_zungqr, 38
 - PLASMA_zunmlq, 39
 - PLASMA_zunmqr, 40
- PLASMA_Complex64_t_Tile
 - PLASMA_zcgels_Tile, 126
 - PLASMA_zcgesv_Tile, 127
 - PLASMA_zcposv_Tile, 127
 - PLASMA_zcungesv_Tile, 128
 - PLASMA_zgelqf_Tile, 129
 - PLASMA_zgelqs_Tile, 130
 - PLASMA_zgels_Tile, 130
 - PLASMA_zgemm_Tile, 131
 - PLASMA_zgeqrf_Tile, 131
 - PLASMA_zgeqrs_Tile, 132
 - PLASMA_zgesv_Tile, 133
 - PLASMA_zgetrf_Tile, 133
 - PLASMA_zgetrs_Tile, 134
 - PLASMA_zhemm_Tile, 134
 - PLASMA_zher2k_Tile, 135
 - PLASMA_zherk_Tile, 136
 - PLASMA_zlange_Tile, 137
 - PLASMA_zlanhe_Tile, 137
 - PLASMA_zlansy_Tile, 138
 - PLASMA_zlauum_Tile, 138
 - PLASMA_zplghe_Tile, 139
 - PLASMA_zplgsy_Tile, 139
 - PLASMA_zplrnt_Tile, 140
 - PLASMA_zposv_Tile, 140
 - PLASMA_zpotrf_Tile, 141
 - PLASMA_zpotri_Tile, 142
 - PLASMA_zpotrs_Tile, 142
 - PLASMA_zsymm_Tile, 143
 - PLASMA_zsyr2k_Tile, 144
 - PLASMA_zsyrk_Tile, 144
 - PLASMA_ztrmm_Tile, 145
 - PLASMA_ztrsm_Tile, 146
 - PLASMA_ztrsmpl_Tile, 147
 - PLASMA_ztrtri_Tile, 147
 - PLASMA_zunglq_Tile, 148
 - PLASMA_zungqr_Tile, 148
 - PLASMA_zunmlq_Tile, 149
 - PLASMA_zunmqr_Tile, 150
- PLASMA_Complex64_t_Tile_Async
 - PLASMA_zcgels_Tile_Async, 217
 - PLASMA_zcgesv_Tile_Async, 217
 - PLASMA_zcposv_Tile_Async, 217
 - PLASMA_zcungesv_Tile_Async, 218
 - PLASMA_zgelqf_Tile_Async, 218
 - PLASMA_zgelqs_Tile_Async, 219
 - PLASMA_zgels_Tile_Async, 219
 - PLASMA_zgemm_Tile_Async, 219
 - PLASMA_zgeqrf_Tile_Async, 220
 - PLASMA_zgeqrs_Tile_Async, 220
 - PLASMA_zgesv_Tile_Async, 221
 - PLASMA_zgetrf_Tile_Async, 221
 - PLASMA_zgetrs_Tile_Async, 222
 - PLASMA_zhemm_Tile_Async, 222
 - PLASMA_zher2k_Tile_Async, 222
 - PLASMA_zherk_Tile_Async, 223

- PLASMA_zlange_Tile_Async, 223
- PLASMA_zlanhe_Tile_Async, 224
- PLASMA_zlansy_Tile_Async, 224
- PLASMA_zLapack_to_Tile_Async, 224
- PLASMA_zlauum_Tile_Async, 225
- PLASMA_zplghe_Tile_Async, 225
- PLASMA_zplgsy_Tile_Async, 226
- PLASMA_zplrnt_Tile_Async, 226
- PLASMA_zposv_Tile_Async, 227
- PLASMA_zpotrf_Tile_Async, 227
- PLASMA_zpotri_Tile_Async, 227
- PLASMA_zpotrs_Tile_Async, 228
- PLASMA_zsymm_Tile_Async, 228
- PLASMA_zsyr2k_Tile_Async, 229
- PLASMA_zsyrk_Tile_Async, 229
- PLASMA_zTile_to_Lapack_Async, 230
- PLASMA_ztrmm_Tile_Async, 230
- PLASMA_ztrsm_Tile_Async, 230
- PLASMA_ztrsmpl_Tile_Async, 231
- PLASMA_ztrtri_Tile_Async, 231
- PLASMA_zunglq_Tile_Async, 232
- PLASMA_zungqr_Tile_Async, 232
- PLASMA_zunmlq_Tile_Async, 233
- PLASMA_zunmqr_Tile_Async, 233
- plasma_context_map_struct, 283
- plasma_context_struct, 284
- PLASMA_cplghe
 - PLASMA_Complex32_t, 56
- PLASMA_cplghe_Tile
 - PLASMA_Complex32_t_Tile, 161
- PLASMA_cplghe_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 242
- PLASMA_cplgsy
 - PLASMA_Complex32_t, 56
- PLASMA_cplgsy_Tile
 - PLASMA_Complex32_t_Tile, 162
- PLASMA_cplgsy_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 243
- PLASMA_cplrnt
 - PLASMA_Complex32_t, 57
- PLASMA_cplrnt_Tile
 - PLASMA_Complex32_t_Tile, 162
- PLASMA_cplrnt_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 243
- PLASMA_cposv
 - PLASMA_Complex32_t, 57
- PLASMA_cposv_Tile
 - PLASMA_Complex32_t_Tile, 163
- PLASMA_cposv_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 244
- PLASMA_cpotrf
 - PLASMA_Complex32_t, 58
- PLASMA_cpotrf_Tile
 - PLASMA_Complex32_t_Tile, 164
- PLASMA_cpotrf_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 244
- PLASMA_cpotri
 - PLASMA_Complex32_t, 59
- PLASMA_cpotri_Tile
 - PLASMA_Complex32_t_Tile, 164
- PLASMA_cpotri_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 244
- PLASMA_cpotrs
 - PLASMA_Complex32_t, 60
- PLASMA_cpotrs_Tile
 - PLASMA_Complex32_t_Tile, 165
- PLASMA_cpotrs_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 245
- PLASMA_csymm
 - PLASMA_Complex32_t, 61
- PLASMA_csymm_Tile
 - PLASMA_Complex32_t_Tile, 165
- PLASMA_csymm_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 245
- PLASMA_csyr2k
 - PLASMA_Complex32_t, 62
- PLASMA_csyr2k_Tile
 - PLASMA_Complex32_t_Tile, 166
- PLASMA_csyr2k_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 246
- PLASMA_csyrrk
 - PLASMA_Complex32_t, 63
- PLASMA_csyrrk_Tile
 - PLASMA_Complex32_t_Tile, 167
- PLASMA_csyrrk_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 246
- PLASMA_cTile_to_Lapack
 - PLASMA_Complex32_t, 64
- PLASMA_cTile_to_Lapack_Async
 - PLASMA_Complex32_t_Tile_Async, 247
- PLASMA_ctrmm
 - PLASMA_Complex32_t, 64
- PLASMA_ctrmm_Tile
 - PLASMA_Complex32_t_Tile, 168
- PLASMA_ctrmm_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 247
- PLASMA_ctrsm
 - PLASMA_Complex32_t, 65
- PLASMA_ctrsm_Tile
 - PLASMA_Complex32_t_Tile, 168
- PLASMA_ctrsm_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 247
- PLASMA_ctrsmpl
 - PLASMA_Complex32_t, 66
- PLASMA_ctrsmpl_Tile
 - PLASMA_Complex32_t_Tile, 169
- PLASMA_ctrsmpl_Tile_Async
 - PLASMA_Complex32_t_Tile_Async, 248

- PLASMA_ctrtri
 PLASMA_Complex32_t, 67
 PLASMA_ctrtri_Tile
 PLASMA_Complex32_t_Tile, 170
 PLASMA_ctrtri_Tile_Async
 PLASMA_Complex32_t_Tile_Async, 248
 PLASMA_cunglq
 PLASMA_Complex32_t, 67
 PLASMA_cunglq_Tile
 PLASMA_Complex32_t_Tile, 170
 PLASMA_cunglq_Tile_Async
 PLASMA_Complex32_t_Tile_Async, 249
 PLASMA_cungqr
 PLASMA_Complex32_t, 68
 PLASMA_cungqr_Tile
 PLASMA_Complex32_t_Tile, 171
 PLASMA_cungqr_Tile_Async
 PLASMA_Complex32_t_Tile_Async, 249
 PLASMA_cunmlq
 PLASMA_Complex32_t, 69
 PLASMA_cunmlq_Tile
 PLASMA_Complex32_t_Tile, 171
 PLASMA_cunmlq_Tile_Async
 PLASMA_Complex32_t_Tile_Async, 250
 PLASMA_cunmqr
 PLASMA_Complex32_t, 70
 PLASMA_cunmqr_Tile
 PLASMA_Complex32_t_Tile, 172
 PLASMA_cunmqr_Tile_Async
 PLASMA_Complex32_t_Tile_Async, 250
 PLASMA_dgelqf
 double, 72
 PLASMA_dgelqf_Tile
 double_Tile, 175
 PLASMA_dgelqf_Tile_Async
 double_Tile_Async, 252
 PLASMA_dgelqs
 double, 72
 PLASMA_dgelqs_Tile
 double_Tile, 175
 PLASMA_dgelqs_Tile_Async
 double_Tile_Async, 253
 PLASMA_dgels
 double, 73
 PLASMA_dgels_Tile
 double_Tile, 176
 PLASMA_dgels_Tile_Async
 double_Tile_Async, 253
 PLASMA_dgemm
 double, 74
 PLASMA_dgemm_Tile
 double_Tile, 176
 PLASMA_dgemm_Tile_Async
 double_Tile_Async, 253
 PLASMA_dgeqrf
 double, 75
 PLASMA_dgeqrf_Tile
 double_Tile, 177
 PLASMA_dgeqrf_Tile_Async
 double_Tile_Async, 254
 PLASMA_dgeqrs
 double, 76
 PLASMA_dgeqrs_Tile
 double_Tile, 178
 PLASMA_dgeqrs_Tile_Async
 double_Tile_Async, 254
 PLASMA_dgesv
 double, 76
 PLASMA_dgesv_Tile
 double_Tile, 178
 PLASMA_dgesv_Tile_Async
 double_Tile_Async, 255
 PLASMA_dgetrf
 double, 77
 PLASMA_dgetrf_Tile
 double_Tile, 179
 PLASMA_dgetrf_Tile_Async
 double_Tile_Async, 255
 PLASMA_dgetrs
 double, 78
 PLASMA_dgetrs_Tile
 double_Tile, 179
 PLASMA_dgetrs_Tile_Async
 double_Tile_Async, 256
 PLASMA_dlange
 double, 79
 PLASMA_dlange_Tile
 double_Tile, 180
 PLASMA_dlange_Tile_Async
 double_Tile_Async, 256
 PLASMA_dlansy
 double, 79
 PLASMA_dlansy_Tile
 double_Tile, 181
 PLASMA_dlansy_Tile_Async
 double_Tile_Async, 256
 PLASMA_dLapack_to_Tile
 double, 80
 PLASMA_dLapack_to_Tile_Async
 double_Tile_Async, 257
 PLASMA_dlauum
 double, 81
 PLASMA_dlauum_Tile
 double_Tile, 181
 PLASMA_dlauum_Tile_Async
 double_Tile_Async, 257
 PLASMA_dorglq
 double, 81

- PLASMA_dorglq_Tile
double_Tile, [182](#)
- PLASMA_dorglq_Tile_Async
double_Tile_Async, [258](#)
- PLASMA_dorgqr
double, [82](#)
- PLASMA_dorgqr_Tile
double_Tile, [182](#)
- PLASMA_dorgqr_Tile_Async
double_Tile_Async, [258](#)
- PLASMA_dormlq
double, [83](#)
- PLASMA_dormlq_Tile
double_Tile, [183](#)
- PLASMA_dormlq_Tile_Async
double_Tile_Async, [258](#)
- PLASMA_dormqr
double, [84](#)
- PLASMA_dormqr_Tile
double_Tile, [183](#)
- PLASMA_dormqr_Tile_Async
double_Tile_Async, [259](#)
- PLASMA_dpLgsy
double, [84](#)
- PLASMA_dpLgsy_Tile
double_Tile, [184](#)
- PLASMA_dpLgsy_Tile_Async
double_Tile_Async, [259](#)
- PLASMA_dplrint
double, [85](#)
- PLASMA_dplrint_Tile
double_Tile, [184](#)
- PLASMA_dplrint_Tile_Async
double_Tile_Async, [260](#)
- PLASMA_dposv
double, [86](#)
- PLASMA_dposv_Tile
double_Tile, [185](#)
- PLASMA_dposv_Tile_Async
double_Tile_Async, [260](#)
- PLASMA_dpotrf
double, [87](#)
- PLASMA_dpotrf_Tile
double_Tile, [186](#)
- PLASMA_dpotrf_Tile_Async
double_Tile_Async, [261](#)
- PLASMA_dpotri
double, [87](#)
- PLASMA_dpotri_Tile
double_Tile, [186](#)
- PLASMA_dpotri_Tile_Async
double_Tile_Async, [261](#)
- PLASMA_dpotrs
double, [88](#)
- PLASMA_dpotrs_Tile
double_Tile, [187](#)
- PLASMA_dpotrs_Tile_Async
double_Tile_Async, [261](#)
- PLASMA_dsgesv
double, [89](#)
- PLASMA_dsgesv_Tile
double_Tile, [187](#)
- PLASMA_dsgesv_Tile_Async
double_Tile_Async, [262](#)
- PLASMA_dsposv
double, [90](#)
- PLASMA_dsposv_Tile
double_Tile, [188](#)
- PLASMA_dsposv_Tile_Async
double_Tile_Async, [262](#)
- PLASMA_dsungesv
double, [91](#)
- PLASMA_dsungesv_Tile
double_Tile, [189](#)
- PLASMA_dsungesv_Tile_Async
double_Tile_Async, [263](#)
- PLASMA_dsymm
double, [93](#)
- PLASMA_dsymm_Tile
double_Tile, [190](#)
- PLASMA_dsymm_Tile_Async
double_Tile_Async, [263](#)
- PLASMA_dsyr2k
double, [94](#)
- PLASMA_dsyr2k_Tile
double_Tile, [190](#)
- PLASMA_dsyr2k_Tile_Async
double_Tile_Async, [264](#)
- PLASMA_dsyrk
double, [95](#)
- PLASMA_dsyrk_Tile
double_Tile, [191](#)
- PLASMA_dsyrk_Tile_Async
double_Tile_Async, [264](#)
- PLASMA_dTile_to_Lapack
double, [96](#)
- PLASMA_dTile_to_Lapack_Async
double_Tile_Async, [264](#)
- PLASMA_dtrmm
double, [96](#)
- PLASMA_dtrmm_Tile
double_Tile, [192](#)
- PLASMA_dtrmm_Tile_Async
double_Tile_Async, [265](#)
- PLASMA_dtrsm
double, [97](#)
- PLASMA_dtrsm_Tile
double_Tile, [193](#)

- PLASMA_dtrsm_Tile_Async
 double_Tile_Async, 265
 PLASMA_dtrsmp1
 double, 98
 PLASMA_dtrsmp1_Tile
 double_Tile, 193
 PLASMA_dtrsmp1_Tile_Async
 double_Tile_Async, 266
 PLASMA_dtrtri
 double, 99
 PLASMA_dtrtri_Tile
 double_Tile, 194
 PLASMA_dtrtri_Tile_Async
 double_Tile_Async, 266
 PLASMA_sgelqf
 float, 101
 PLASMA_sgelqf_Tile
 float_Tile, 197
 PLASMA_sgelqf_Tile_Async
 float_Tile_Async, 268
 PLASMA_sgelqs
 float, 101
 PLASMA_sgelqs_Tile
 float_Tile, 197
 PLASMA_sgelqs_Tile_Async
 float_Tile_Async, 268
 PLASMA_sgels
 float, 102
 PLASMA_sgels_Tile
 float_Tile, 198
 PLASMA_sgels_Tile_Async
 float_Tile_Async, 269
 PLASMA_sgemm
 float, 103
 PLASMA_sgemm_Tile
 float_Tile, 198
 PLASMA_sgemm_Tile_Async
 float_Tile_Async, 269
 PLASMA_sgeqrf
 float, 104
 PLASMA_sgeqrf_Tile
 float_Tile, 199
 PLASMA_sgeqrf_Tile_Async
 float_Tile_Async, 270
 PLASMA_sgeqrs
 float, 104
 PLASMA_sgeqrs_Tile
 float_Tile, 199
 PLASMA_sgeqrs_Tile_Async
 float_Tile_Async, 270
 PLASMA_sgesv
 float, 105
 PLASMA_sgesv_Tile
 float_Tile, 200
 PLASMA_sgesv_Tile_Async
 float_Tile_Async, 271
 PLASMA_sgetrf
 float, 106
 PLASMA_sgetrf_Tile
 float_Tile, 201
 PLASMA_sgetrf_Tile_Async
 float_Tile_Async, 271
 PLASMA_sgetrs
 float, 107
 PLASMA_sgetrs_Tile
 float_Tile, 201
 PLASMA_sgetrs_Tile_Async
 float_Tile_Async, 271
 PLASMA_slange
 float, 107
 PLASMA_slange_Tile
 float_Tile, 202
 PLASMA_slange_Tile_Async
 float_Tile_Async, 272
 PLASMA_slansy
 float, 108
 PLASMA_slansy_Tile
 float_Tile, 202
 PLASMA_slansy_Tile_Async
 float_Tile_Async, 272
 PLASMA_sLapack_to_Tile
 float, 109
 PLASMA_sLapack_to_Tile_Async
 float_Tile_Async, 273
 PLASMA_slauum
 float, 109
 PLASMA_slauum_Tile
 float_Tile, 203
 PLASMA_slauum_Tile_Async
 float_Tile_Async, 273
 PLASMA_sorglq
 float, 110
 PLASMA_sorglq_Tile
 float_Tile, 204
 PLASMA_sorglq_Tile_Async
 float_Tile_Async, 273
 PLASMA_sorgqr
 float, 111
 PLASMA_sorgqr_Tile
 float_Tile, 204
 PLASMA_sorgqr_Tile_Async
 float_Tile_Async, 274
 PLASMA_sormlq
 float, 111
 PLASMA_sormlq_Tile
 float_Tile, 205
 PLASMA_sormlq_Tile_Async
 float_Tile_Async, 274

- PLASMA_sormqr
 - float, [112](#)
- PLASMA_sormqr_Tile
 - float_Tile, [205](#)
- PLASMA_sormqr_Tile_Async
 - float_Tile_Async, [275](#)
- PLASMA_splgsy
 - float, [113](#)
- PLASMA_splgsy_Tile
 - float_Tile, [206](#)
- PLASMA_splgsy_Tile_Async
 - float_Tile_Async, [275](#)
- PLASMA_splrnt
 - float, [114](#)
- PLASMA_splrnt_Tile
 - float_Tile, [206](#)
- PLASMA_splrnt_Tile_Async
 - float_Tile_Async, [276](#)
- PLASMA_sposv
 - float, [114](#)
- PLASMA_sposv_Tile
 - float_Tile, [207](#)
- PLASMA_sposv_Tile_Async
 - float_Tile_Async, [276](#)
- PLASMA_spotrf
 - float, [115](#)
- PLASMA_spotrf_Tile
 - float_Tile, [208](#)
- PLASMA_spotrf_Tile_Async
 - float_Tile_Async, [276](#)
- PLASMA_spotri
 - float, [116](#)
- PLASMA_spotri_Tile
 - float_Tile, [208](#)
- PLASMA_spotri_Tile_Async
 - float_Tile_Async, [277](#)
- PLASMA_spotrs
 - float, [117](#)
- PLASMA_spotrs_Tile
 - float_Tile, [209](#)
- PLASMA_spotrs_Tile_Async
 - float_Tile_Async, [277](#)
- PLASMA_ssymm
 - float, [117](#)
- PLASMA_ssymm_Tile
 - float_Tile, [209](#)
- PLASMA_ssymm_Tile_Async
 - float_Tile_Async, [278](#)
- PLASMA_ssy2k
 - float, [118](#)
- PLASMA_ssy2k_Tile
 - float_Tile, [210](#)
- PLASMA_ssy2k_Tile_Async
 - float_Tile_Async, [278](#)
- PLASMA_ssy2k
 - float, [119](#)
- PLASMA_ssy2k_Tile
 - float_Tile, [211](#)
- PLASMA_ssy2k_Tile_Async
 - float_Tile_Async, [279](#)
- PLASMA_sTile_to_Lapack
 - float, [120](#)
- PLASMA_sTile_to_Lapack_Async
 - float_Tile_Async, [279](#)
- PLASMA_strmm
 - float, [121](#)
- PLASMA_strmm_Tile
 - float_Tile, [211](#)
- PLASMA_strmm_Tile_Async
 - float_Tile_Async, [279](#)
- PLASMA_strsm
 - float, [122](#)
- PLASMA_strsm_Tile
 - float_Tile, [212](#)
- PLASMA_strsm_Tile_Async
 - float_Tile_Async, [280](#)
- PLASMA_strsmpl
 - float, [123](#)
- PLASMA_strsmpl_Tile
 - float_Tile, [213](#)
- PLASMA_strsmpl_Tile_Async
 - float_Tile_Async, [280](#)
- PLASMA_strtri
 - float, [123](#)
- PLASMA_strtri_Tile
 - float_Tile, [214](#)
- PLASMA_strtri_Tile_Async
 - float_Tile_Async, [281](#)
- PLASMA_zcgels
 - PLASMA_Complex64_t, [7](#)
- PLASMA_zcgels_Tile
 - PLASMA_Complex64_t_Tile, [126](#)
- PLASMA_zcgels_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, [217](#)
- PLASMA_zcgesv
 - PLASMA_Complex64_t, [8](#)
- PLASMA_zcgesv_Tile
 - PLASMA_Complex64_t_Tile, [127](#)
- PLASMA_zcgesv_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, [217](#)
- PLASMA_zcposv
 - PLASMA_Complex64_t, [9](#)
- PLASMA_zcposv_Tile
 - PLASMA_Complex64_t_Tile, [127](#)
- PLASMA_zcposv_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, [217](#)
- PLASMA_zcungesv
 - PLASMA_Complex64_t, [11](#)

- PLASMA_zcungesv_Tile
 PLASMA_Complex64_t_Tile, 128
 PLASMA_zcungesv_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 218
 PLASMA_zgelqf
 PLASMA_Complex64_t, 12
 PLASMA_zgelqf_Tile
 PLASMA_Complex64_t_Tile, 129
 PLASMA_zgelqf_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 218
 PLASMA_zgelqs
 PLASMA_Complex64_t, 13
 PLASMA_zgelqs_Tile
 PLASMA_Complex64_t_Tile, 130
 PLASMA_zgelqs_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 219
 PLASMA_zgels
 PLASMA_Complex64_t, 13
 PLASMA_zgels_Tile
 PLASMA_Complex64_t_Tile, 130
 PLASMA_zgels_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 219
 PLASMA_zgemm
 PLASMA_Complex64_t, 14
 PLASMA_zgemm_Tile
 PLASMA_Complex64_t_Tile, 131
 PLASMA_zgemm_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 219
 PLASMA_zgeqrf
 PLASMA_Complex64_t, 15
 PLASMA_zgeqrf_Tile
 PLASMA_Complex64_t_Tile, 131
 PLASMA_zgeqrf_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 220
 PLASMA_zgeqrs
 PLASMA_Complex64_t, 16
 PLASMA_zgeqrs_Tile
 PLASMA_Complex64_t_Tile, 132
 PLASMA_zgeqrs_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 220
 PLASMA_zgesv
 PLASMA_Complex64_t, 17
 PLASMA_zgesv_Tile
 PLASMA_Complex64_t_Tile, 133
 PLASMA_zgesv_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 221
 PLASMA_zgetrf
 PLASMA_Complex64_t, 18
 PLASMA_zgetrf_Tile
 PLASMA_Complex64_t_Tile, 133
 PLASMA_zgetrf_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 221
 PLASMA_zgetrs
 PLASMA_Complex64_t, 18
 PLASMA_zgetrs_Tile
 PLASMA_Complex64_t_Tile, 134
 PLASMA_zgetrs_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 222
 PLASMA_zhemm
 PLASMA_Complex64_t, 19
 PLASMA_zhemm_Tile
 PLASMA_Complex64_t_Tile, 134
 PLASMA_zhemm_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 222
 PLASMA_zher2k
 PLASMA_Complex64_t, 20
 PLASMA_zher2k_Tile
 PLASMA_Complex64_t_Tile, 135
 PLASMA_zher2k_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 222
 PLASMA_zherk
 PLASMA_Complex64_t, 21
 PLASMA_zherk_Tile
 PLASMA_Complex64_t_Tile, 136
 PLASMA_zherk_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 223
 PLASMA_zlange
 PLASMA_Complex64_t, 22
 PLASMA_zlange_Tile
 PLASMA_Complex64_t_Tile, 137
 PLASMA_zlange_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 223
 PLASMA_zlanhe
 PLASMA_Complex64_t, 23
 PLASMA_zlanhe_Tile
 PLASMA_Complex64_t_Tile, 137
 PLASMA_zlanhe_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 224
 PLASMA_zlansy
 PLASMA_Complex64_t, 24
 PLASMA_zlansy_Tile
 PLASMA_Complex64_t_Tile, 138
 PLASMA_zlansy_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 224
 PLASMA_zLapack_to_Tile
 PLASMA_Complex64_t, 25
 PLASMA_zLapack_to_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 224
 PLASMA_zlauum
 PLASMA_Complex64_t, 25
 PLASMA_zlauum_Tile
 PLASMA_Complex64_t_Tile, 138
 PLASMA_zlauum_Tile_Async
 PLASMA_Complex64_t_Tile_Async, 225
 PLASMA_zplghe
 PLASMA_Complex64_t, 26
 PLASMA_zplghe_Tile
 PLASMA_Complex64_t_Tile, 139

- PLASMA_zplghe_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 225
- PLASMA_zplgsy
 - PLASMA_Complex64_t, 26
- PLASMA_zplgsy_Tile
 - PLASMA_Complex64_t_Tile, 139
- PLASMA_zplgsy_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 226
- PLASMA_zplrnt
 - PLASMA_Complex64_t, 27
- PLASMA_zplrnt_Tile
 - PLASMA_Complex64_t_Tile, 140
- PLASMA_zplrnt_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 226
- PLASMA_zposv
 - PLASMA_Complex64_t, 27
- PLASMA_zposv_Tile
 - PLASMA_Complex64_t_Tile, 140
- PLASMA_zposv_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 227
- PLASMA_zpotrf
 - PLASMA_Complex64_t, 28
- PLASMA_zpotrf_Tile
 - PLASMA_Complex64_t_Tile, 141
- PLASMA_zpotrf_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 227
- PLASMA_zpotri
 - PLASMA_Complex64_t, 29
- PLASMA_zpotri_Tile
 - PLASMA_Complex64_t_Tile, 142
- PLASMA_zpotri_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 227
- PLASMA_zpotrs
 - PLASMA_Complex64_t, 30
- PLASMA_zpotrs_Tile
 - PLASMA_Complex64_t_Tile, 142
- PLASMA_zpotrs_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 228
- PLASMA_zsymm
 - PLASMA_Complex64_t, 31
- PLASMA_zsymm_Tile
 - PLASMA_Complex64_t_Tile, 143
- PLASMA_zsymm_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 228
- PLASMA_zsyr2k
 - PLASMA_Complex64_t, 32
- PLASMA_zsyr2k_Tile
 - PLASMA_Complex64_t_Tile, 144
- PLASMA_zsyr2k_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 229
- PLASMA_zsyrk
 - PLASMA_Complex64_t, 33
- PLASMA_zsyrk_Tile
 - PLASMA_Complex64_t_Tile, 144
- PLASMA_zsyrk_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 229
- PLASMA_zTile_to_Lapack
 - PLASMA_Complex64_t, 34
- PLASMA_zTile_to_Lapack_Async
 - PLASMA_Complex64_t_Tile_Async, 230
- PLASMA_ztrmm
 - PLASMA_Complex64_t, 34
- PLASMA_ztrmm_Tile
 - PLASMA_Complex64_t_Tile, 145
- PLASMA_ztrmm_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 230
- PLASMA_ztrsm
 - PLASMA_Complex64_t, 35
- PLASMA_ztrsm_Tile
 - PLASMA_Complex64_t_Tile, 146
- PLASMA_ztrsm_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 230
- PLASMA_ztrsml
 - PLASMA_Complex64_t, 36
- PLASMA_ztrsml_Tile
 - PLASMA_Complex64_t_Tile, 147
- PLASMA_ztrsml_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 231
- PLASMA_ztrtri
 - PLASMA_Complex64_t, 37
- PLASMA_ztrtri_Tile
 - PLASMA_Complex64_t_Tile, 147
- PLASMA_ztrtri_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 231
- PLASMA_zunglq
 - PLASMA_Complex64_t, 37
- PLASMA_zunglq_Tile
 - PLASMA_Complex64_t_Tile, 148
- PLASMA_zunglq_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 232
- PLASMA_zungqr
 - PLASMA_Complex64_t, 38
- PLASMA_zungqr_Tile
 - PLASMA_Complex64_t_Tile, 148
- PLASMA_zungqr_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 232
- PLASMA_zunmlq
 - PLASMA_Complex64_t, 39
- PLASMA_zunmlq_Tile
 - PLASMA_Complex64_t_Tile, 149
- PLASMA_zunmlq_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 233
- PLASMA_zunmqr
 - PLASMA_Complex64_t, 40
- PLASMA_zunmqr_Tile
 - PLASMA_Complex64_t_Tile, 150
- PLASMA_zunmqr_Tile_Async
 - PLASMA_Complex64_t_Tile_Async, 233

primedec, [285](#)

pthread_cond_s, [286](#)

pthread_s, [287](#)

Simple Interface - Double Complex, [5](#)

Simple Interface - Double Real, [71](#)

Simple Interface - Single Complex, [41](#)

Simple Interface - Single Real, [100](#)