# CLEF: A Framework for Solving Conservation-law Equations

Andrew McMurry

April 7, 2014

# Finite volume method

$$U_t + \nabla \cdot F(U) = S(U) \tag{1}$$

# Finite volume method

$$U_t + \nabla \cdot F(U) = S(U) \tag{1}$$

$$\bar{U}^i = \frac{1}{V^i} \int_{V^i} U dV \tag{2}$$

# Finite volume method

$$U_t + \nabla \cdot F(U) = S(U) \tag{1}$$

$$\bar{U}^i = \frac{1}{V^i} \int_{V^i} U dV \tag{2}$$

$$\frac{1}{V^i} \int_{V^i} U_t dV + \frac{1}{V^i} \int_{V^i} \nabla \cdot F(U) dV = \frac{1}{V^i} \int_{V^i} S(U) dV \tag{3}$$

# Finite volume method

$$U_t + \nabla \cdot F(U) = S(U) \tag{1}$$

$$\bar{U}^i = \frac{1}{V^i} \int_{V^i} U dV \tag{2}$$

$$\bar{U}^i_t + \frac{1}{V^i} \int_{V^i} \nabla \cdot F(U) dV = \frac{1}{V^i} \int_{V^i} S(U) dV \tag{3}$$

# Finite volume method

$$U_t + \nabla \cdot F(U) = S(U) \tag{1}$$

$$\bar{U}^i = \frac{1}{V^i} \int_{V^i} U dV \tag{2}$$

$$\bar{U}^i_t + \frac{1}{V^i} \oint_{S^i} F(U) \cdot dS = \frac{1}{V^i} \int_{V^i} S(U) dV \tag{3}$$

# Finite volume method

$$U_t + \nabla \cdot F(U) = S(U) \tag{1}$$

$$\bar{U}^i = \frac{1}{V^i} \int_{V^i} U dV \tag{2}$$

$$\bar{U}_t^i + \sum_j \left( \frac{A^j}{V^i} \right) \bar{F}^j(\bar{U}) = \frac{1}{V^i} \int_{V^i} S(U) dV \tag{3}$$

# Finite volume method

$$U_t + \nabla \cdot F(U) = S(U) \tag{1}$$

$$\bar{U}^i = \frac{1}{V^i} \int_{V^i} U dV \tag{2}$$

$$\bar{U}^i_t + \sum_j \left( \frac{A^j}{V^i} \right) \bar{F}^j(\bar{U}) = \bar{S}^i(\bar{U}) \tag{3}$$

# Goals of the Framework

- Promote code re-use

# Goals of the Framework

- Promote code re-use
- Support a library of standard algorithms

# Goals of the Framework

- Promote code re-use
- Support a library of standard algorithms
- Avoid implementation specific dependencies

# Goals of the Framework

- Promote code re-use
- Support a library of standard algorithms
- Avoid implementation specific dependencies
- Standard interfaces for algorithm types

# Goals of the Framework

- Promote code re-use
- Support a library of standard algorithms
- Avoid implementation specific dependencies
- Standard interfaces for algorithm types
- Efficient executable

# Problem

- The System of Equations

# Problem

- The System of Equations
  - The Conserved Variables

# Problem

- The System of Equations
  - The Conserved Variables
  - The Flux Function

# Problem

- The System of Equations
  - The Conserved Variables
  - The Flux Function
  - Extra Information for Riemann Solver

# Problem

- The System of Equations
  - The Conserved Variables
  - The Flux Function
  - Extra Information for Riemann Solver
  - Source Terms

# Problem

- The System of Equations
  - The Conserved Variables
  - The Flux Function
  - Extra Information for Riemann Solver
  - Source Terms
- The Initial Conditions

# Problem

- The System of Equations
  - The Conserved Variables
  - The Flux Function
  - Extra Information for Riemann Solver
  - Source Terms
- The Initial Conditions
- Boundary Conditions

# Method

- Domain Discretization

# Method

- Domain Discretization
- Riemann Solver

# Method

- Domain Discretization
- Riemann Solver
- Order of Accuracy

# Method

- Domain Discretization
- Riemann Solver
- Order of Accuracy
- Methods for Source Terms

# Sections

## Problem

- The System of Equations
  - **The Conserved Variables**
  - The Flux Function
  - Extra Information for Riemann Solver
  - Source Terms
- The Initial Conditions
- Boundary Conditions

## Method

- Domain Discretization
- Riemann Solver
- Order of Accuracy
- Methods for Source Terms

# The Conserved Variables - Traditional Implementation

```
double r[NX][NY][NZ];
double rux[NX][NY][NZ];
double ruy[NX][NY][NZ];
double ruz[NX][NY][NZ];
double E[NX][NY][NZ];

void forward_euler(double drdt, double druxdt, double druydt, double druzdt,
                   double dEdt, int i, int j, int k, double dt)
{
  r[i][j][k] = r[i][j][k] + dt * drdt;
  rux[i][j][k] = rux[i][j][k] + dt * druxdt;
  ruy[i][j][k] = ruy[i][j][k] + dt * druydt;
  ruz[i][j][k] = ruz[i][j][k] + dt * druzdt;
  E[i][j][k] = E[i][j][k] + dt * dEdt;
}
```

# The Conserved Variables - Structural Implementation

```
class Variable {
public:
  double r;
  vector3 ru;
  double E;
};

Variable U[NX][NY][NZ];

void forward_euler(Variable dUdt, int i, int j, int k, double dt)
{
  U[i][j][k] = U[i][j][k] + dt * dUdt;
}
```

# The Conserved Variables - Operator Implementation

```
static inline Variable operator+(const Variable &a, const Variable &b)
{
  Variable v;
  v.r = a.r + b.r;
  v.ru = a.ru + b.ru;
  v.E = a.E + b.E;
  return v;
}

static inline Variable operator*(const Variable &a, double n)
{
  Variable v;
  v.r = a.r * n;
  v.ru = a.ru * n;
  v.E = a.E * n;
  return v;
}
```

# Automating Operator Implementation

- script part of the build process that generates these functions in C++.

# Automating Operator Implementation

- script part of the build process that generates these functions in C++.
- extend C++ with introspection.

# Automating Operator Implementation

- script part of the build process that generates these functions in C++.
- extend C++ with introspection.
- create a new programming language.

# The Conserved Variables - Automated Operators

```
static inline Variable operator+(const Variable &a, const Variable &b)
{
  Variable v;
  for (auto M : Variable) v.*M = a.*M + b.*M;
  return v;
}

static inline Variable operator*(const Variable &a, double n)
{
  Variable v;
  for (auto M : Variable) v.*M = a.*M * n;
  return v;
}
```

# Sections

**Problem**

- The System of Equations
  - *The Conserved Variables*
  - The Flux Function
  - Extra Information for Riemann Solver
  - Source Terms
- The Initial Conditions
- Boundary Conditions

**Method**

- **Domain Discretization**
- Riemann Solver
- Order of Accuracy
- Methods for Source Terms

# Domain Discretization - 3d Regular Grid

```
Variable U[NX][NY][NZ];
Variable Fx[NX+1][NY][NZ];
Variable Fy[NX][NY+1][NZ];
Variable Fz[NX][NY][NZ+1];

void update_variables(double dt)
{
  for (int i=0; i<NX; i++) {
    for (int j=0; i<NY; j++) {
      for (int k=0; i<NZ; k++) {

        U[i][j][k] -= dt * ((Fx[i+1][j][k] - Fx[i][j][k]) / dx +
                            (Fy[i][j+1][k] - Fy[i][j][k]) / dy +
                            (Fz[i][j][k+1] - Fz[i][j][k]) / dz);
      }
    }
  }
}
```

# Domain Discretization - General

```
array_over_cells<Variable> U;
array_over_faces<Variable> F;

void update_variables(double dt)
{
  for (cell i: allcells()) {
    for (face j: facesofcell(i)) {
      U[i] -= dt * (j.area() / i.volume()) * F[j];
    }
  }
}
```

# Domain Discretisation - Choices

- Regular Grid vs Irregular Mesh

# Domain Discretisation - Choices

- Regular Grid vs Irregular Mesh
- Adaptive Mesh implementations

# Domain Discretisation - Choices

- Regular Grid vs Irregular Mesh
- Adaptive Mesh implementations
- Domain parallelization

# Domain Discretisation - Choices

- Regular Grid vs Irregular Mesh
- Adaptive Mesh implementations
- Domain parallelization
- GPU or CPU

# Sections

**Problem**

- The System of Equations
  - *The Conserved Variables*
  - **The Flux Function**
  - **Extra Information for Riemann Solver**
  - Source Terms
- The Initial Conditions
- Boundary Conditions

**Method**

- *Domain Discretization*
- Riemann Solver
- Order of Accuracy
- Methods for Source Terms

# The Flux Function

```
void point_flux(Variable &F, Variable U, dir d)
{
  vector3 u = U.ru / U.r;
  double p = (GAMMA-1) * (U.E - 0.5*U.ru*u);

  F.r = U.ru * d;
  F.ru = U.ru * (u * d) + p * d;
  F.E = (U.E + p) * (u * d);
}
```

# Sections

**Problem**

- The System of Equations
  - *The Conserved Variables*
  - *The Flux Function*
  - *Extra Information for Riemann Solver*
  - Source Terms
- The Initial Conditions
- Boundary Conditions

**Method**

- *Domain Discretization*
- **Riemann Solver**
- Order of Accuracy
- Methods for Source Terms

# Riemann Solver - HLL Implementation

```
void calculate_flux(Variable &F, const Variable &lU, const Variable &rU, dir d)
{
  real sl, sr;

  speeds2(sl, sr, lU, rU, d);
  if (sl > 0) point_flux(F, lU, d);
  else if (sr < 0) point_flux(F, rU, d);
  else {
    Variable lF, rF;

    point_flux(lF, lU, d);
    point_flux(rF, rU, d);
    F = (sr*lF - sl*rF + sr*sl*(rU - lU))/(sr-sl);
  }
}
```

# Sections

**Problem**

- The System of Equations
  - *The Conserved Variables*
  - *The Flux Function*
  - *Extra Information for Riemann Solver*
  - Source Terms
- The Initial Conditions
- Boundary Conditions

**Method**

- *Domain Discretization*
- *Riemann Solver*
- **Order of Accuracy**
- Methods for Source Terms

# Second Order - Minmod

```
void interpolate(Variable &fU, face f, dir d, real pos)
{
  cell i = f.cell();
  Variable dU = 0.5*minmod(U[i+d]-U[i], U[i]-U[i-d]);
  if (f.d == d) fU = U[i] - dU;
  else fU = U[i] + dU;
}
```

# Sections

**Problem**

- The System of Equations
  - *The Conserved Variables*
  - *The Flux Function*
  - *Extra Information for Riemann Solver*
  - Source Terms
- **The Initial Conditions**
- Boundary Conditions

**Method**

- *Domain Discretization*
- *Riemann Solver*
- *Order of Accuracy*
- Methods for Source Terms

# Sections

**Problem**

- The System of Equations
  - *The Conserved Variables*
  - *The Flux Function*
  - *Extra Information for Riemann Solver*
  - Source Terms
- *The Initial Conditions*
- **Boundary Conditions**

**Method**

- *Domain Discretization*
- *Riemann Solver*
- *Order of Accuracy*
- Methods for Source Terms

# Sections

**Problem**

- The System of Equations
  - *The Conserved Variables*
  - *The Flux Function*
  - *Extra Information for Riemann Solver*
  - **Source Terms**
- *The Initial Conditions*
- *Boundary Conditions*

**Method**

- *Domain Discretization*
- *Riemann Solver*
- *Order of Accuracy*
- **Methods for Source Terms**

# Source Terms

- Every Source Term is different
- Even simple ones might require well balancing
- Intrusive implementations
- ‘Aspect Oriented’ style

# Code Insertion

```
const vector grav(0, 0, GRAVITY);

alter start calculate_simultaneous_sources(cell c)
{
  S[c].m -= grav * U[c].rho;
  S[c].E += grav * U[c].m;
}
```

# Finite Difference Method

- Don't use Riemann Solver or Finite-volume high order methods

# Finite Difference Method

- Don't use Riemann Solver or Finite-volume high order methods
- Need to provide a finite difference divergence formula
- And compatible source term methods

# Finite Difference Method

- Don't use Riemann Solver or Finite-volume high order methods
- Need to provide a finite difference divergence formula
- And compatible source term methods
- Or function for $U_t$ for other hyperbolic equations

# For the Scientist

- Scientist has new problem

# For the Scientist

- Scientist has new problem
- Implements model and maybe system of equations

# For the Scientist

- Scientist has new problem
- Implements model and maybe system of equations
- Lots of standard solvers and other features already available

# For the Scientist

- Scientist has new problem
- Implements model and maybe system of equations
- Lots of standard solvers and other features already available
- Can easily try all of them to see which one works best

## For the Applied Mathematician

- Applied Mathematician designs new solver

# For the Applied Mathematician

- Applied Mathematician designs new solver
- Implements solver using the framework, maybe altering similar one

# For the Applied Mathematician

- Applied Mathematician designs new solver
- Implements solver using the framework, maybe altering similar one
- Lots of standard systems of equations and test cases already available

# For the Applied Mathematician

- Applied Mathematician designs new solver
- Implements solver using the framework, maybe altering similar one
- Lots of standard systems of equations and test cases already available
- New test cases easy to add

# Conclusions

- Implement algorithms without adding dependencies

# Conclusions

- Implement algorithms without adding dependencies
- Easy to swap between alternative algorithms

# Conclusions

- Implement algorithms without adding dependencies
- Easy to swap between alternative algorithms
- Less programming required for a new problem

# Conclusions

- Implement algorithms without adding dependencies
- Easy to swap between alternative algorithms
- Less programming required for a new problem
- Future proofing as much as we can

# Availability

http://bitbucket.org/clef/clef/