

# Concordance in Parallel

## The problem

Given a text extract the location of equal word strings for strings of words of lengths 1..N in terms of the starting location of the word string in the text, provided the word string is repeated a minimum number of times.

## The solution outline

The solution comprises five stages as follow.

### Stage 1

Processing word strings and in particular comparing them is complex and time consuming so we shall extract all the words from the text, removing unnecessary punctuation, and then calculate a corresponding integer value for each word based upon summing the letter codes for the word. We shall store these words and the list of word values.

### Stage 2

The next stage is to create the sums of sequence of values for strings of length 1 to N, which generate N valueLists.

### Stage 3

The next stage is to create N maps each of which comprises a key based on a value from a valueList and an entry that contains all the locations where that value occurs. These maps are called indicesMap.

### Stage 4

The next stage disambiguates indicesMap because a key value may refer to different word sequences. Thus for each key in an indicesMap we extract the word sequence to which it corresponds, recall that the entry contains the location of the value in the text and we have stored the words. Thus we can build up a map comprising a key of a word string together with an entry comprising the locations of that specific word string. We shall call the N maps the wordsMap.

### Stage 5

The final stage is to output the wordsMap in a human readable form ensuring that only strings that are repeated at least the minimum number of times are output.

## Transformation to an Algorithm

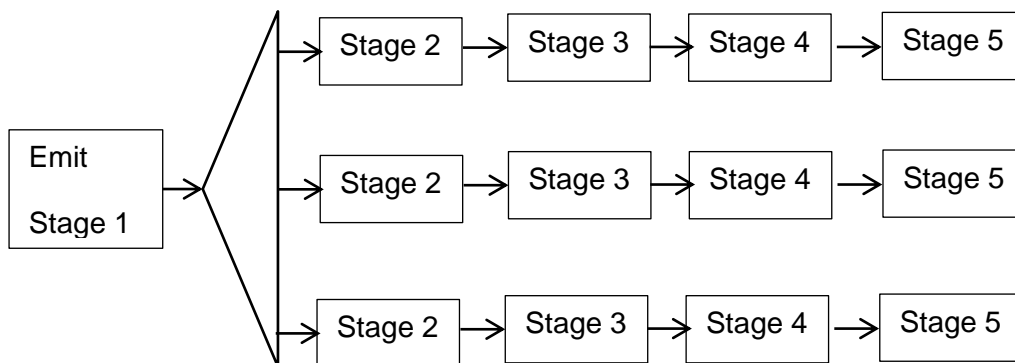
### Sequential Solution

Each stage is carried out in turn for each value of N

### Parallel Solution

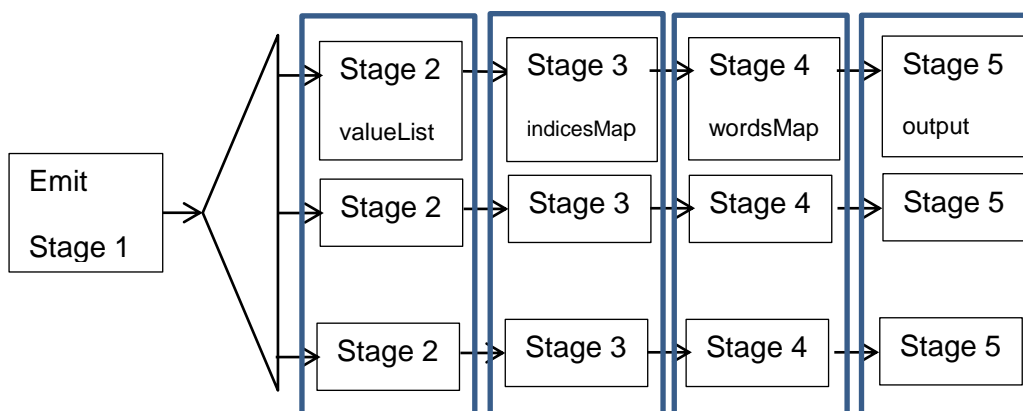
This can be achieved in a number of ways of which there are two obvious ones. We shall assume that we can do Stage 5 in parallel.

Stage 1 will be allocated to an Emit process. The remaining Stages can be allocated to a parallel composition as follows, assuming 3 worker steams.

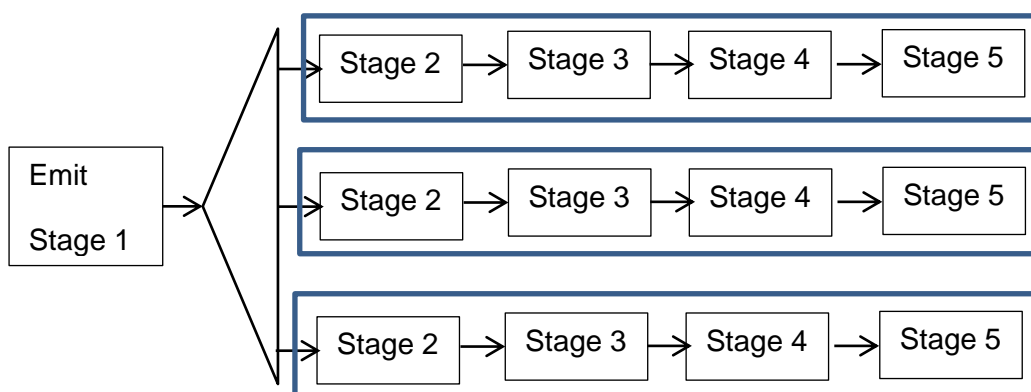


However, these can be partitioned in two different ways

A Pipeline of Groups where each Group undertakes the same stage. In this case a Group could be considered as an unrolling of a For-loop. This is implemented using the pattern `PipelineOfGroupsPattern`.



As a parallel collection (Farm ?) of Pipelines



The functions allocated to each stage are

Stage1 ConcordanceData.initClass ( [N, fileName, outFileName] )

ConcordanceData.createInstance ( null )

Stage 2 ConcordanceData.invoke ( fn (valueList, null) )

Stage 3 ConcordanceData.invoke ( fn (indicesMap, null) )

Stage 4 ConcordanceData.invoke ( fn (wordsMap, null) )

Stage 5 ConcordanceResult.initClass ( [ minSeqLen ] )

ConcordanceResult.collector ( concordanceData\_instance )

ConcordanceResult.finalise()

Where fn( f, operandList ) is shorthand for run the function identified by the integer value f using the parameters given in operandList