Documentation for practical application

# Problem Specification

The problem requires to compute 2 \* (N – 2) linear systems over a set of matrixes until a certain number of steps are performed or when an error function returns an acceptable amount.

Scenarios:

* A test case when the limits of the problem are met N = 1000
* Various values for the error threshold

# Description of a sequential algorithm that solves the problem

## Pseudocode

Solve() {

While (steps < max\_Steps && error > min\_error)

solveX1()

solveX2()

computeError()

copyX2intoX0()

}

solveX1(){

for j = 1, n-1

new B

B[0] = border(0, X0)

For i = 1, n-1

B[i] = f(X0[i][j], X0[i+1][j], X0[i-1][j],X0[i][j+1],X0[i][j-1])

B[n-1] = border(1, X0)

Thomas(a, b, c, res, B)

Copy res into X1, column j

Copy colum 0 of X0 into X1

Copy column n – 1 of X0 into X1

}

Similarly for X2

Then error is computed as the maximal absolute difference of elements between X2, X0

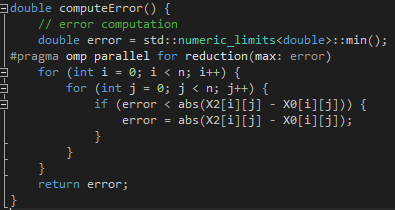
## Complexity analysis

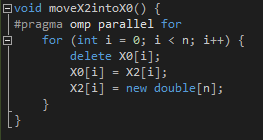
The main loops depends on how fast the algoritm converges. The complexity for calculating X1 is given by the main loop (N) and the inner loops (4\*N), to be roughly N^2. The same applies for calculating X2, and for getting the error. The application stores all 3 matrixes in memory, so 3 \* N^2 memory usage is estimated.

In conclusion the time complexity for the serial solution: O(N^2), and the space complexity O(N^2).

# Description of a parallel solution

## Informal description of the solution

The most obvious target for optimization is the error computation function which can be parallelized using the reduction pattern. Each row is processed in parallel and the resulting errors are combined.



Another quick optimization is during the moving of X2 into X0. We can do the pointer moving and allocations in parallel.

For the computation of X1 and X2 we applied a message passing approach. Following the master slave pattern, a master process sends out the initial matrix to the workers. Each worker computes the B array, solves the tridiagonal system and stores the result in a buffer. The indexes for which equation systems are solved by each worker are determined from its id as follows: #workers % id, with an increment of 1 if the result is 0 (the master process). This ensures the distribution of work in a round robin fashion.

Each worker then sends the buffer containing the concatenated results to the master process. This approach was taken in favour of sending the result of each Thomas run because it implies less overhead with message passing.

The master process waits for the results, each receive in a different thread, to deal with incoming messages asynchronously. The X1 matrix is then built by parsing the incoming buffers. A very similar approach is taken for the computation of X2.