# Introduction

Hej och så vidare…

## Problem

The task is to calculate function values by using an approximation of the Laplace’s equation (see Equation 1) over an interval from xmin to xmax and ymin to ymax (i.e. within some given boundary values of the two variables).

**Equation 1: Laplace’s equation (2D)**

The values of both x and y are to increase by fixed values from and to their corresponding minima and maxima which allows for a matrix representation of the problem.

|  |  |
| --- | --- |
| **y** |  |
|  | **\* \* \* \* \* \* \* \***  **\*** \* \* \* \* \* \* **\***  **\*** \* \* \* \* \* \* **\***  **\*** \* \* \* \* \* \* **\***  **\*** \* \* \* \* \* \* **\***  **\*** \* \* \* \* \* \* **\***  **\*** \* \* \* \* \* \* **\***  **\* \* \* \* \* \* \* \*** |
|  | **x** |

Figure 1. Each star represents a xy-pair for which Φ(x, y) is to be determined.

The Jacobi method is to be used to approximate the function values for the corresponding in-between points. This is an iterative method that for each iteration approximates every point’s new value as an average of its horizontal and vertical neighboring points. To start the Jacobi method iteration, see Equation 2, some initial values are required. The boundary values, in Figure 1 represented as bolded stars (**\***), must be calculated or set prior to the execution of the Jacobi iteration. They are to represent true values (not approximations) of the mathematical function Φ(x, y) since their values will trickle inwards the matrix as the only input that affects the interior points. Their values will, as the *Dirichlet boundary condition* dictates, not change during iterations. The initial values of the interior points can be set to any number, the closer these values are to the true values of the mathematical function the more precise the solution will be after n iterations (in theory). For practical reasons the boundary values can all be set to 1.0 and the interior values to 0.0 while testing and evaluating the algorithms.

**Equation 2: Jacobi iteration, previous values held in matrix G are   
used to generate new values for matrix N**

# Programs

Four different c programming solutions that implement an optimization of *equation 2* was created and evaluated. The different solutions were the following:

1. *A program using a sequential Jacobi iteration*
2. *A program using a parallel Jacobi iteration*
3. *A multigrid variant of the sequential program*
4. *A multigrid variant of the parallel program*

All solutions was optimized by the use of the improvements presented in the book ***Foundations of Multithreaded, Parallel, and Distributed Programming,*** *by Gregory R. Andrews* [GA s.539-540], which yielded an algorithm with the characteristics listed below.

* *N and G, as represented in equation 2, were referred to by pointers*
* *When all new values of N has been calculated, the N and G matrices' pointers gets swapped as the values gets updated a second time*
* *Two complete grid calculations are executed for every iteration*
* *Instead of the operation division by 4 for every new calculation, the operation multiplication by 0.25 was used*
* *The number of iterations are set at runtime rather than being dependent of the maximum difference from previous iteration's values*

Command line input parameters for all four programs are **gridSize** and **numIters** and for the parallel programs the additional parameter **numWorkers**. The **gridSize** parameter sets the grid size as the length of each side with the boundary points excluded. The **numIters** parameter sets the number of Jacobi iterations to be computed and the **numWorkers** parameter determines the number of processes/threads to be used for the parallel work.

The output is the same for all four programs. To standard output the command line arguments, the execution time of the computational part and the maximum error of the final values are printed. The final values of the grid gets written to a file named **filedata.out**.

The parallel versions were developed using the *OpenMP API*. The workload is divided evenly between processes/threads. A dissemination barrier using counter variables and busy waiting was used instead of the OpenMP built-in implicit barrier.

The multigrid solutions were programmed to use a 4-level V-cycle where the parameter **gridSize** determined the size of the coarsest/smallest grid and the subsequent grid sizes are defined as **[grid size] = 2 · [previous grid size] + 1**. The number of Jacobi iterations to be executed on the initial (smallest) grid is determined by the **numIters** parameter while 4 iterations are executed for each of the subsequent (larger) grids.

## Sequential one-grid method

## Parallel one-grid method

## Multigrid solutions

# Performance Evaluation

# Conclusion

# References

**[GA] Foundations of Multithreaded, Parallel, and Distributed Programming,** by Gregory R. Andrews, Addison-Wesley; (fifth printing 2006), ISBN 0-201-35752-6