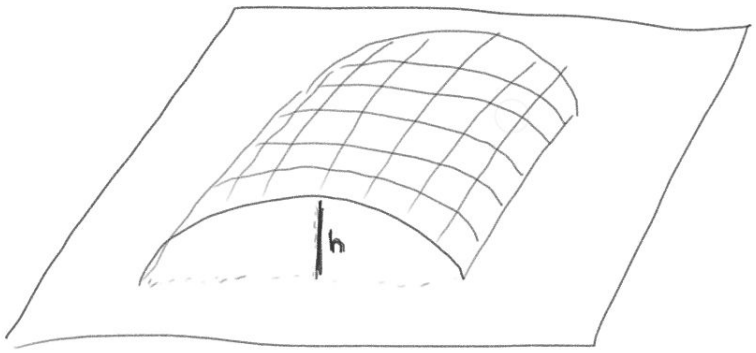
Parallel Computing

Membrane in the shape of a square

Author:
Jan Gorazda

1. **Task**

   There is a membrane in the shape of a square with the side of the length of ‘a’. Membrane is in a static state and is equally charged. Its side is rigidly attached and it is not deformed. Propose algorithm based on differential method of counting the shape (distance from the ground) assuming that this distance function fulfills Laplace equation.



$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

2. Equation solution

The task is to transform the given equation into computable form. The hard part is to transform it in the way that solving it in highly parallel manner is possible.

$$\frac{\sigma^2 h}{\sigma x^2} + \frac{\sigma^2 h}{\sigma y^2} = 0$$

① $$\frac{\sigma h}{\sigma x} = \frac{h(x+1,y) - h(x,y)}{\Delta h}$$

② $$\frac{\sigma h}{\sigma y} = \frac{h(x,y+1) - h(x,y)}{\Delta h}$$

③ $$\frac{\sigma^2 h}{\sigma x^2} = \frac{\frac{\sigma h(x,y)}{\sigma x} - \frac{\sigma h(x-1,y)}{\sigma x}}{\Delta h} =$$

$$\overset{①}{=} \frac{h(x+1,y) - h(x,y) - h(x,y) + h(x-1,y)}{\Delta h^2} =$$

$$= \frac{h(x+1,y) - 2h(x,y) + h(x-1,y)}{\Delta h^2}$$

④ $$\frac{\sigma^2 h}{\sigma y^2} = \frac{\frac{\sigma h(x,y)}{\sigma y} - \frac{\sigma h(x,y-1)}{\sigma y}}{\Delta h} =$$

$$\overset{②}{=} \frac{h(x,y+1) - h(x,y) - h(x,y) + h(x,y-1)}{\Delta h^2} =$$

$$= \frac{h(x,y+1) - 2h(x,y) + h(x,y-1)}{\Delta h^2}$$

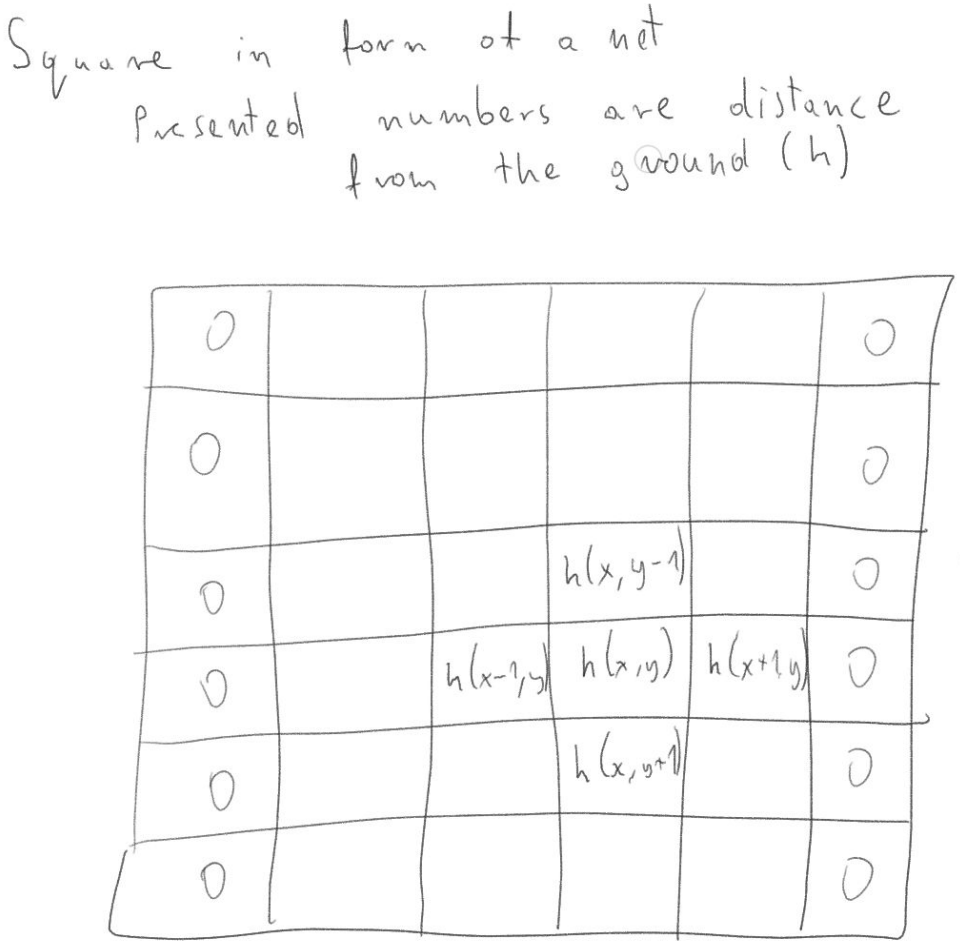
$$\Downarrow$$

$$\frac{\sigma^2 h}{\sigma x^2} + \frac{\sigma^2 h}{\sigma y^2} = 0$$

$$\Downarrow ③ \wedge ④$$

$$\frac{h(x+1,y) + h(x,y+1) + h(x-1,y) + h(x,y-1) - 4h(x,y)}{\Delta h^2} = 0$$

3. **Applying given equation into parallel algorithm**
   As we see from final equation there are 5 variables:
   a. h(x,y)
   b. h(x+1,y)
   c. h(x-1,y)
   d. h(x,y+1)
   e. h(x,y-1)
4. **Initial conditions**
   The goal is to count height of every point in the square (to a certain approximation) and the only way to do that is to use the initial conditions of assignment. We do know that the points closests (again to certain approximation) to the sides of the square are touching the ground so their height equals zero. To formalize it:
   a. h(0,y) = 0
   b. h(x,0) = 0
5. Graphical visualisation



Now we can easily form a system of equations.

6. System of equations

Denotation $h_{1,0} = h(1,0) = h(x,y)$, $x=1, y=0$

$h_{x,0}=0$ $h_{0,y}=0$, $h_{m,y}=0$, $h_{x,n}=0$, n – last index of a net

$$0 = \frac{h(x+1,y) + h(x,y+1) + h(x-1,y) + h(x,y-1) - 4h(x,y)}{\Delta h}$$

$$0 = h_{x+1,y} + h_{x,y+1} + h_{x-1,y} + h_{x,y-1} - 4h_{x,y}$$

The equation system is created:

$y, x \in \mathbb{N}^+$ $x, y \in \{1 \dots n-1\}$

$$\begin{cases} h_{2,1} + h_{1,2} + h_{0,1} + h_{1,0} - 4h_{1,1} = 0 \\ \vdots \\ h_{n,n-1} + h_{n-1,n} + h_{n-2,n-1} + h_{n-1,n-2} - 4h_{n-1,n-1} = 0 \end{cases}$$

7. Solution of such equation system

To solve this we will be using iterationing mesh relaxation method. First the vector h = [h(0,0) .. h(n,n)] will be generated, where

- **a.** h(x,0) = h(0,y) = h(x,n) = h(n,y) = 0
- **b.** 0 <= h(x,y) <=a/2, where a is a length of a square
- **c.** b is a zero vertex with the length of n
- **d.** A is a correct solution matrix to our system

e. In this conditions following is correct:

$A \cdot h = b$

$A = B + R$, R - rest , B - matrix, with easy to solve B^{-1}

$\Downarrow$

$Bh + Rh = b$

$Bh = -Rh + b$

$Bh = -(A - B)h + b$

h^0 - vertex of solutions generated at first

$Bh^{(t+1)} = -(A - B)h^t + b$

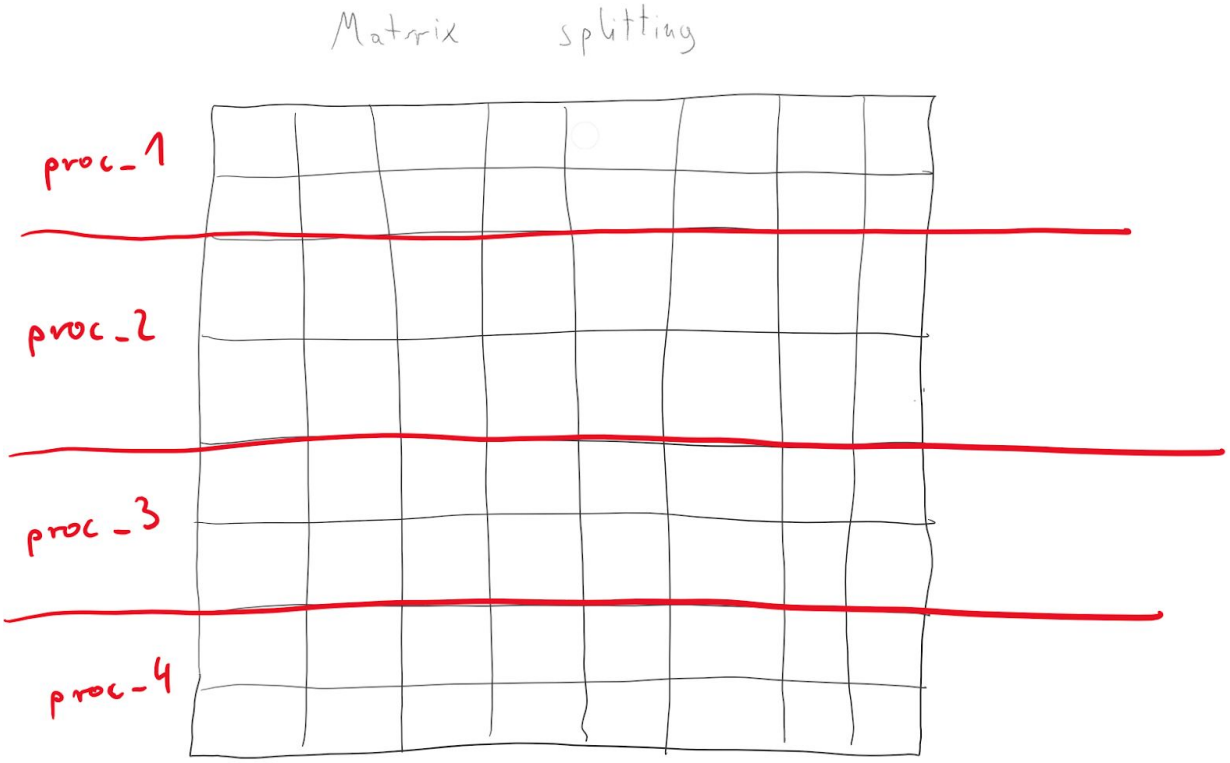
$h^{(t+1)} = \underbrace{-B^{-1}(A - B)h^t}_{M - \text{iteration matrix}} + \underbrace{B^{-1}b}_{\text{in our case } b \text{ is zero vertex so } = 0}$

$h^{(t+1)} = Mh^t$

$M = I - B^{-1}A$

8. **Parallelising iterational method**

To solve this problem in the parallel manner one must split the vertex h into smaller parts. The are multiple ways to split it but this particular solution will focus on splitting by rows.



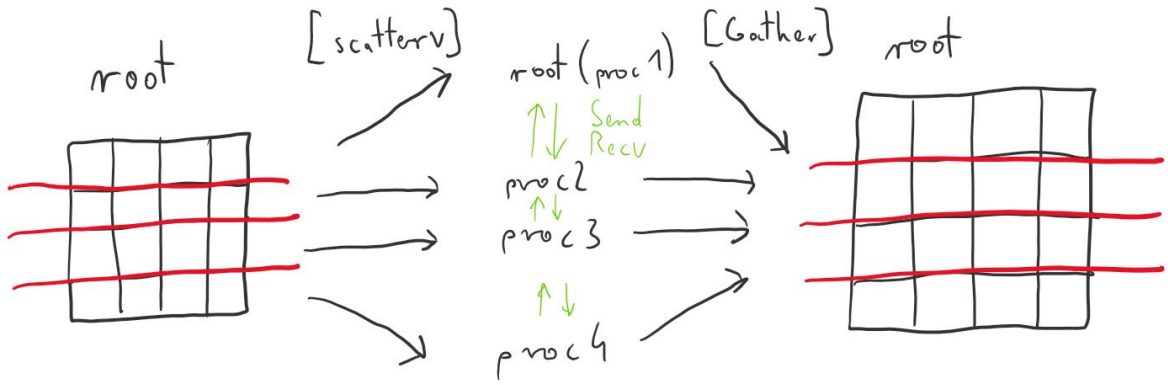
In that way every batch of rows will be computed on a different core and communication between processes will be minimised. Additionally this method could be used several times to use even more threads. Downsides of this method are the requirement that the grid width must be divisible by number of cores

9. **Implementation**

Implementation is presented in the file “grid.py”. It is written in Python language with the use of MPI library. Algorithm:

1. Root processor divides grid into appropriate number of vectors
2. Root is adding upper row and lower row into each vector
3. Root sends those vectors into every processor (including itself)
4. Every processor counts first iteration
5. Every processor sends its first row to previous processor and bottom row to next processor
6. Every processor receives its upper row and lower row
7. Process is repeated n times
8. Every row sends its final vector to root
9. Root builds the matrix out of the vectors

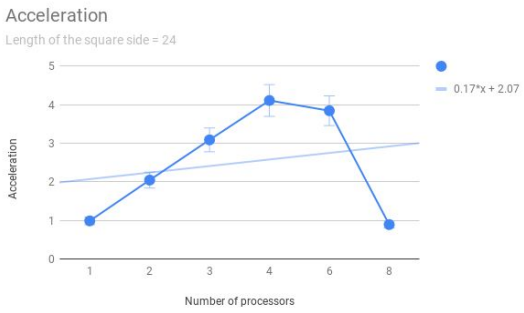
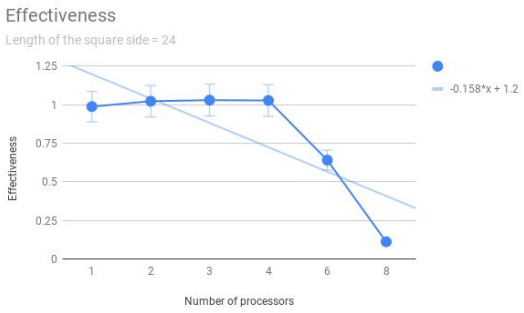
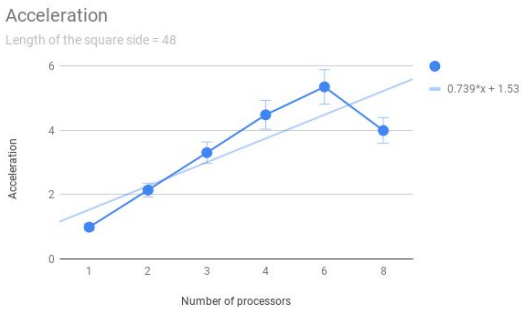
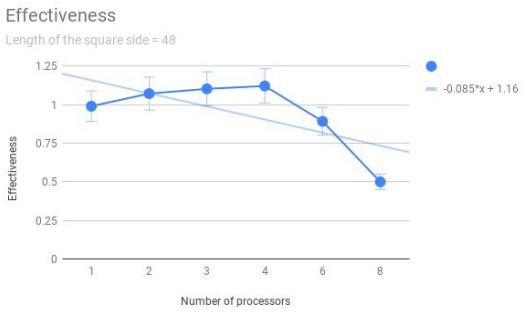
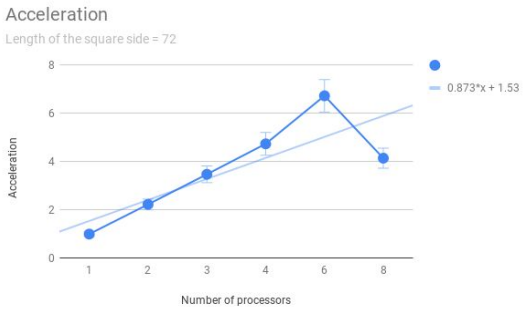
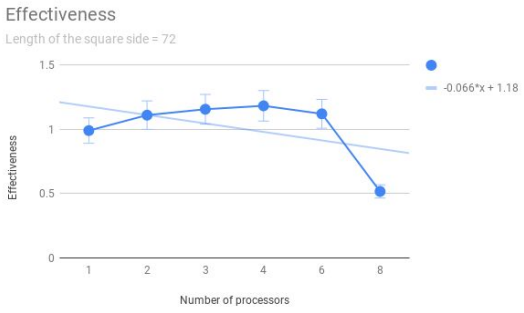
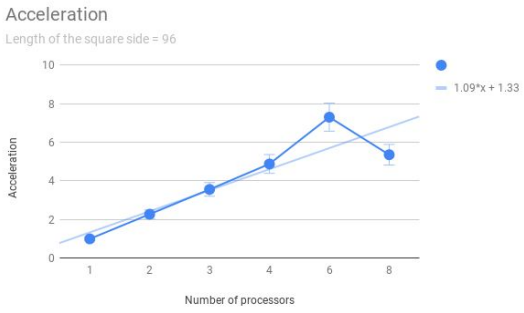
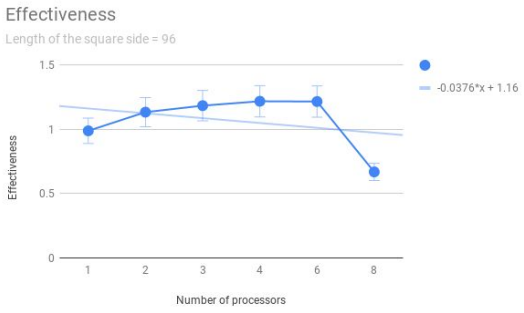
Graphical representation:



10. Environment

Written algorithm was run multiple times on two nodes. Each node had 4 cores and communication between them was done by network. Due to this setup the performance drop when reaching more than 4 processors is to be expected.

11. Results

Acceleration	Effectiveness
	
	
	
	

12. Conclusions

As predicted there is a drop in both acceleration and effectiveness when reaching beyond 4 cores. It is the fault of communication by network. However the drop is less apparent when computing bigger problems. The reason is that computation/communication ratio is increasing. However the one unsettling thing is the ultra performance the program is achieving working on 2,3 and 4 cores. The most obvious explanation is the fact that MPI library might not be optimised for working with only one core and therefore the results of one core computations might be distorted.

13. Equation results

1st iteration	50th iteration	99th iteration