CS 5334 Final Project

MPI Parallelization for 2D Heat Transfer Problem

Group members:

Shaikh Tanveer Hossain (80590803)

Mirza Mohammad Maqbule Elahi (80590755)

Instructor: Dr. Shirley Moore

Date: 05-11-2017 (Thursday)

Introduction:

In different field of science and engineering heat transfer problems are very common an important. Thousands of scenarios are available where we are interested in the distribution of temperature in the body of interest with respect to time. In some problems, there are situations where we can consider the three-dimensional domain of the body as 2d by considering the thickness as zero. Here we are interested on finding the Temperature distribution on a square shaped metal sheet for a given type of boundary and initial conditions.

There are lot of approach available to solve these problems. For example, Finite Element, Finite Volume, Finite difference etc. We here choose the finite difference method for solving our problem.

The mathematical approach behind the finite difference method is relatively easy, but the problem associated with these is that it is computationally intensive. The approach is basically dividing the total square space into lot of little square and the apply the basic heat transfer equation for those. So, the more we divide the domain into small elements the more the accuracy. So, the main challenge is to find the temperature distribution with respect to time for a large size plate with higher order accuracy. For instance, in our case we divided the 2d space into 512*512 blocks, 1024*1024 blocks and 2048*2048 blocks. For sure the last one is more accurate prediction. However, from the computational point of view when solving 512*512 blocks then the number of elements is 262144. So, we have to solve the heat equation for 262144 times for unit time difference. Which is in the case of less accuracy. Now when solving the last case then we need to solve 4194304 number of elements. From computational point of view which 16 times more work to do.

To solve this problem writing a serial code is also relatively easy. However, when the problem size Is big then for the serial code it would take a lot of time to solve this problem. Problem will become worst with a small-time difference. To solve this challenge the solution is parallel computing. We here applied the Massage Passing Interface (MPI) module in C programming to solving the problem.

MPI meaning message passing interface is a set of Application Programmer Interfaces(APIs), called from user programs written in C, C++, Fortran, Python and other languages. Some implementations include open source and generic. MPI runs on nodes/machines and the same program is executed but nodes work on different parts of the program's data. There is standard library for message passing used to program using C or Fortran.

Physical Problem Description:

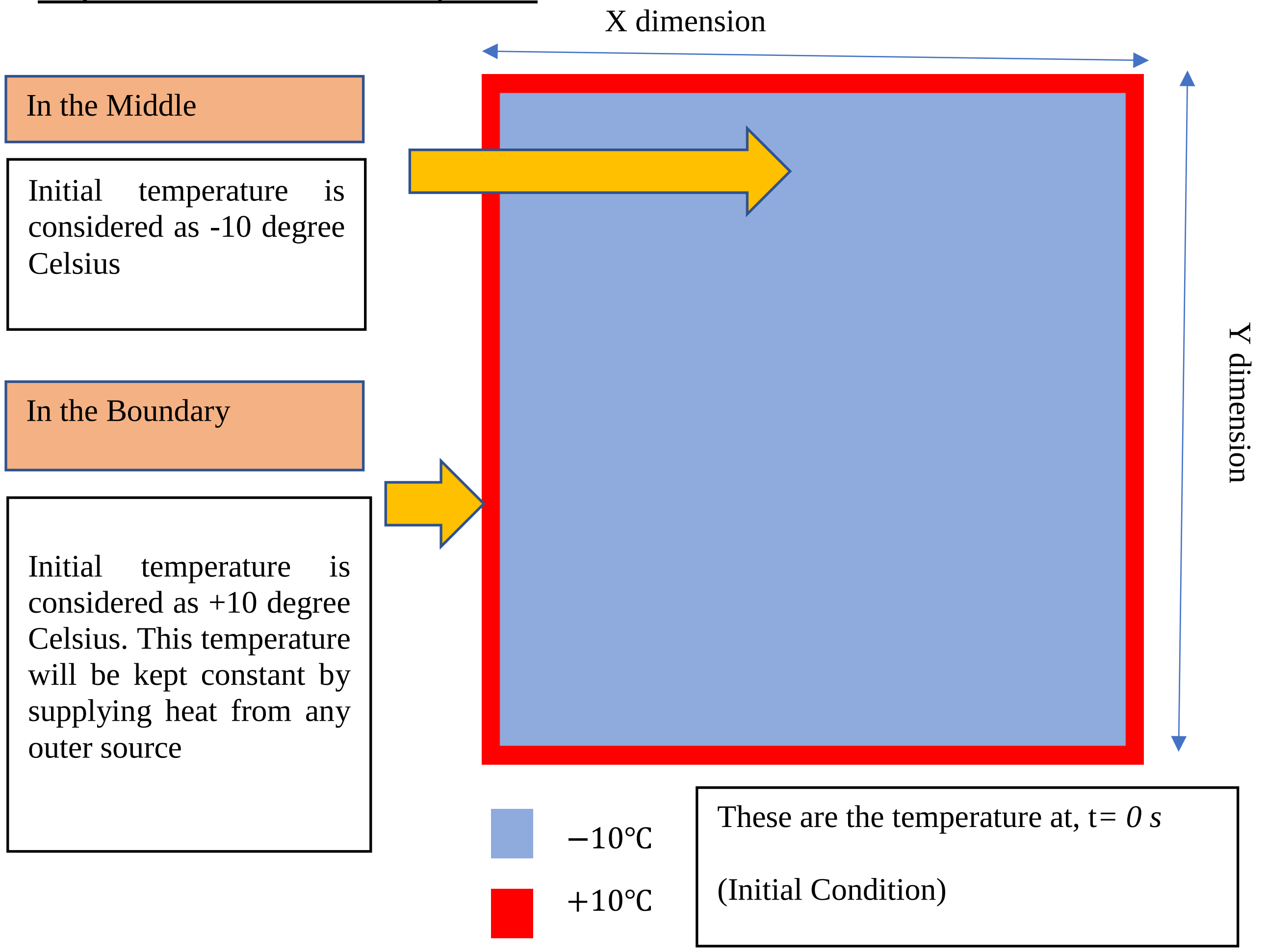

X dimension
Y dimension
In the Middle
Initial temperature is considered as -10 degree Celsius
In the Boundary
Initial temperature is considered as +10 degree Celsius. This temperature will be kept constant by supplying heat from any outer source
−10℃
+10℃
These are the temperature at, t= 0 s
(Initial Condition)

Figure 1: Initial and boundary conditions and problem description

We considered a rectangular sheet metal as top for our problem. Initially, the temperature of the total sheet is considered as uniform and equal to -10 degree Celsius. On the boundary that means on the four edges the temperature is considered as 10 degree Celsius and it is kept constant with time. Now if the time passes then as because heat always flows form the high temperature to low temperature so after some time the plate will gain some heat. The pattern will be look like that the places near the edges will be become warmer first then the middle parts will also become warm gradually. A sample of how the result may look like is described below in the figure.

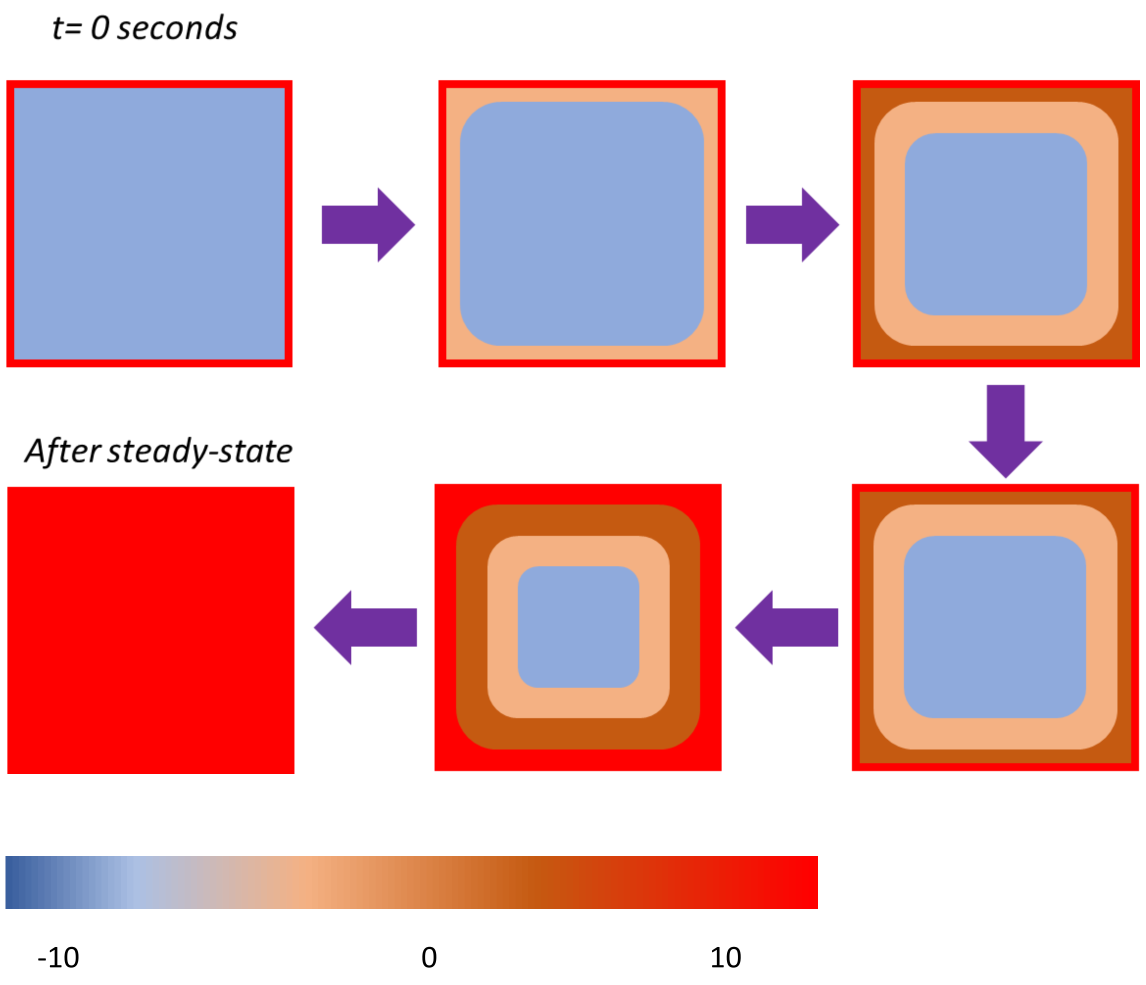


Figure 2: Temperature distribution change pattern

Mathematical Background:

The unsteady 2D heat equation is as follows,

$$\frac{\partial T}{\partial t} = k\Delta T \ \ with\, \Delta \ = \ \sum_{i=1}^{n} \frac{\partial^2}{\partial x_i^2} \qquad (1)$$

the Laplacian in n dimension

Where, k coefficient is the thermal diffusivity. Hence 2D heat equation becomes,

$$\frac{\partial \theta}{\partial t} = k\left(\frac{\partial^2 \theta}{\delta x^2} + \frac{\partial^2 \theta}{\delta y^2}\right) \qquad (2)$$

Taylor series expansion of the spatial domain,

$$\theta_{m+1} = \theta(x_m + h) = \theta(x_m) + h\theta'(x_m) + \frac{h^2}{2}\theta''(x_m) + \frac{h^3}{2}\theta'''(x_m) + O(h^4)$$
3.1

$$\theta_{m-1} = \theta(x_m - h) = \theta(x_m) + h\theta'(x_m) + \frac{h^2}{2}\theta''(x_m) + \frac{h^3}{2}\theta'''(x_m) + O(h^4) \qquad 3.2$$

The second derivative at x_m is obtained by adding (3.1) and (3.2),

$$\theta''(x_m) = \frac{\theta_{m+1} - 2\theta_m + \theta_{m-1}}{h^2} + O(h^2) \qquad (3.3)$$

There is same set of equation for ‘y‘ variable. Taking h_x= $size_x/N_x$ and h_y= $size_y/N_y$ and $\theta(x_m,y_m)$= θ[i][j]

$$\frac{\partial^2\theta}{\delta x^2}+\frac{\partial^2\theta}{\delta y^2}=\frac{\theta_{[i+1][j]}-2\theta_{[i][j]}+\theta_{[i-1][j]}}{h_x^2}+\frac{\theta_{[i][j+1]}-2\theta_{[i][j]}+\theta_{[i][j-1]}}{h_y^2} \quad (4)$$

The left-hand side of (2) can be written as,

$$\frac{\partial\theta(x_m,y_m,t_n)}{\partial t}=\frac{\partial\theta(x_m,y_m,t_{n+1})-\partial\theta(x_m,y_m,t_n)}{\Delta t}$$

$$\rightarrow \frac{\partial\theta(x_m,y_m,t_n)}{\partial t}=\frac{\theta_{n+1[i][j]}-\theta_{n[i][j]}}{\Delta t} \quad (5)$$

Combining (2), (4) and (5) we obtain,

$$\theta_{n+1[i][j]}=\theta_{n[i][j]}+\mathrm{k}\Delta t\left[\frac{\theta_{n[i+1][j]}-2\theta_{n[i][j]}+\theta_{n[i-1][j]}}{h_x^2}+\frac{\theta_{n[i][j+1]}-2\theta_{n[i][j]}+\theta_{n[i][j-1]}}{h_y^2}\right] \quad (7)$$

Convergence criteria is given by,

$$\Delta t \leq \frac{h^2}{4k} \quad (8)$$

Algorithm:

Serial code:

For the serial code the input parameters are: X & Y dimension, number of steps, time steps and convergence. X & Y determines the matrix size. Total run time for the simulation is total time steps times either maximum number of steps or convergence value (if convergence is achieved before the maximum steps). The calculations are done for the 5x5 matrix inside the 7x7 matrix. The differential distance h_x and h_y are the smallest we are considering.

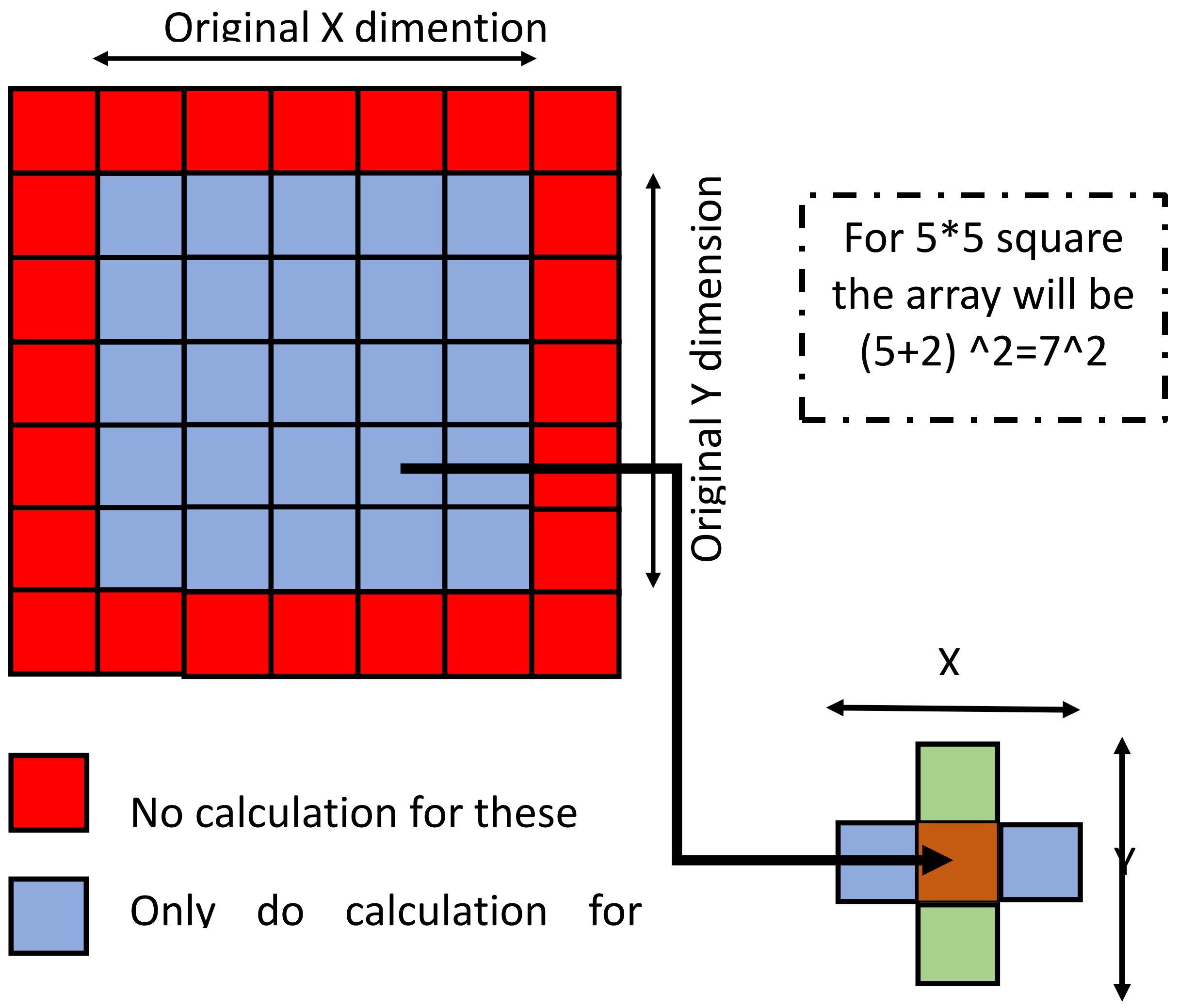
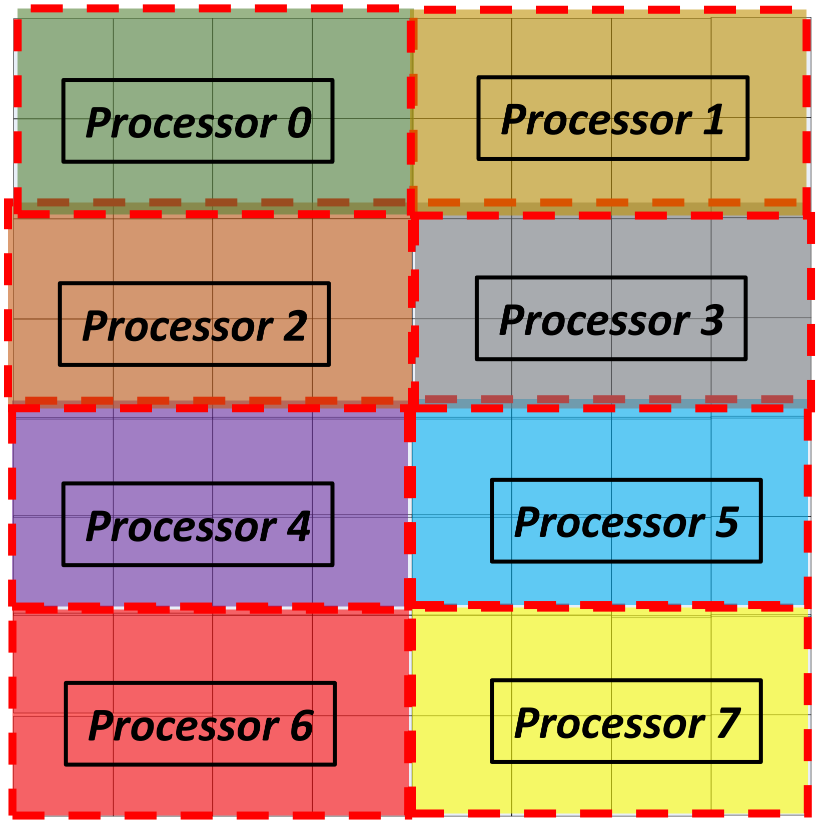


Figure 3: Memory allocation and calculation technique for serial code

The elements in the boundary are allocated only to assign the boundary condition. The temperature calculation is also done for the elements which are inside the boundary block. Here in the figure the blue elements are the elements for doing calculation. The red elements are the boundary elements not for calculation. In the figure the neighbor elements for one elements is shown. There are four neighboring elements here. They are top, bottom, left and right.

Parallel code:



```
MPI_Cart_create(comm,    ndims,    dims,
periods, reorganisation, &comm2d);
```

Figure 3: Memory allocation and calculation technique for serial code

For parallel programming, multiple processors are used and they form a 2D array. The input arguments are read from a file and here also the parameters are same except we need two extra parameters. They are the number of processors in the x direction and the number of processors in the y direction. These are to arrange the processors in a 2d array. Form the figure in the case shown it will be 2 for X dimension and 4 for y dimension. MPI_Bcast was used to send these information to all the processors. Here we used the MPI_Cart_create command to arrange the processors in the 2D array, which uses the cartesian coordinate x and y. For this command, the comm is the basic communicator and comm2d is the output communicator. ndims is the number of dimensions and the dims is a 1d array which contains the dimension in the x and y directions respectively. The periods are kept as false or zero which means that the processors at the edges are not wrapped with the processors in the opposite edge. Here one of the very important thing to remember is that the number of elements in the x direction must be divisible by number of processors in the x direction and this is also true for y direction.

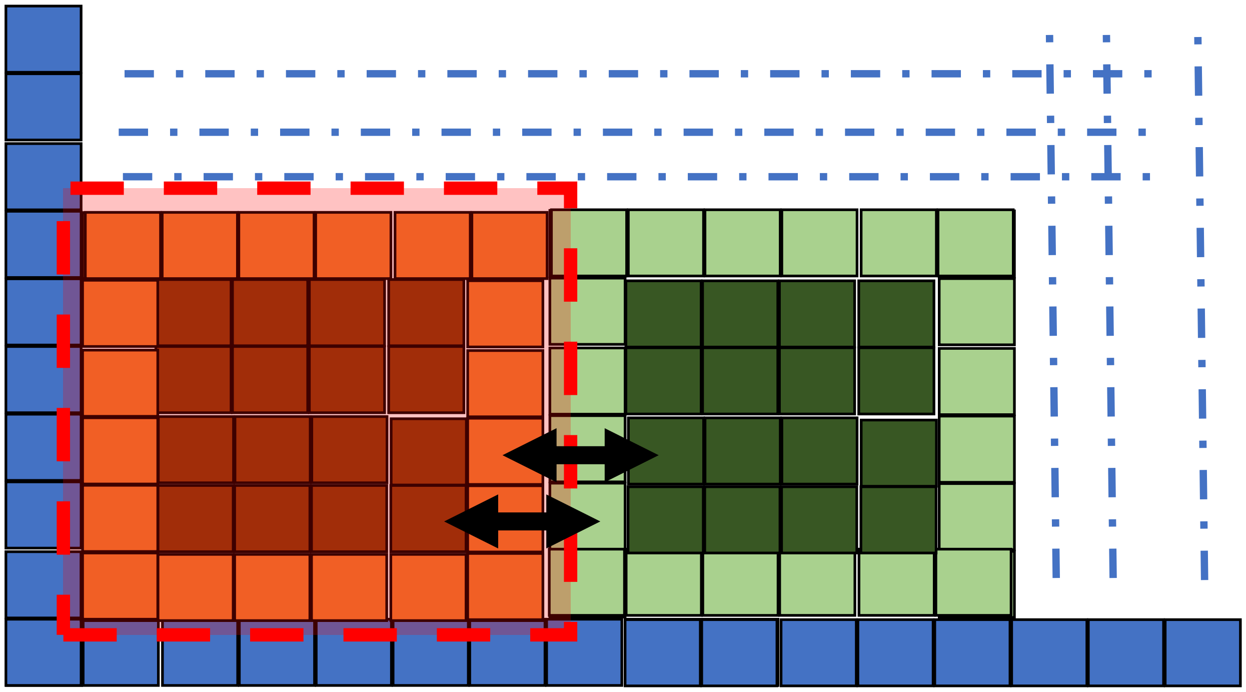


Figure 4: Memory allocation in parallel code

For parallel code, we need the calculation elements and the boundary condition elements just like the serial code but beside that for communication purpose we declared and extra layer of elements around the calculation element block. Here the red dotted box the area consumed by one single processor and the deep colored elements are the elements for calculation where light colored are for communication purpose. When processors are calculating elements at the edge of its own block then they need not only its own data but also data from the neighbor elements which is consumed by the neighboring processor. So the chance is one processor have to wait until the neighboring one is not done with its work. This will make the system tremendously slow. To avoid this problem, we have declared those extra layers of elements. Before getting started with the next step of calculation the data are copied form the edge cells to the same sided communication cells of the neighboring processors. So, after that each processor has all the data needed to do calculation for all its elements.

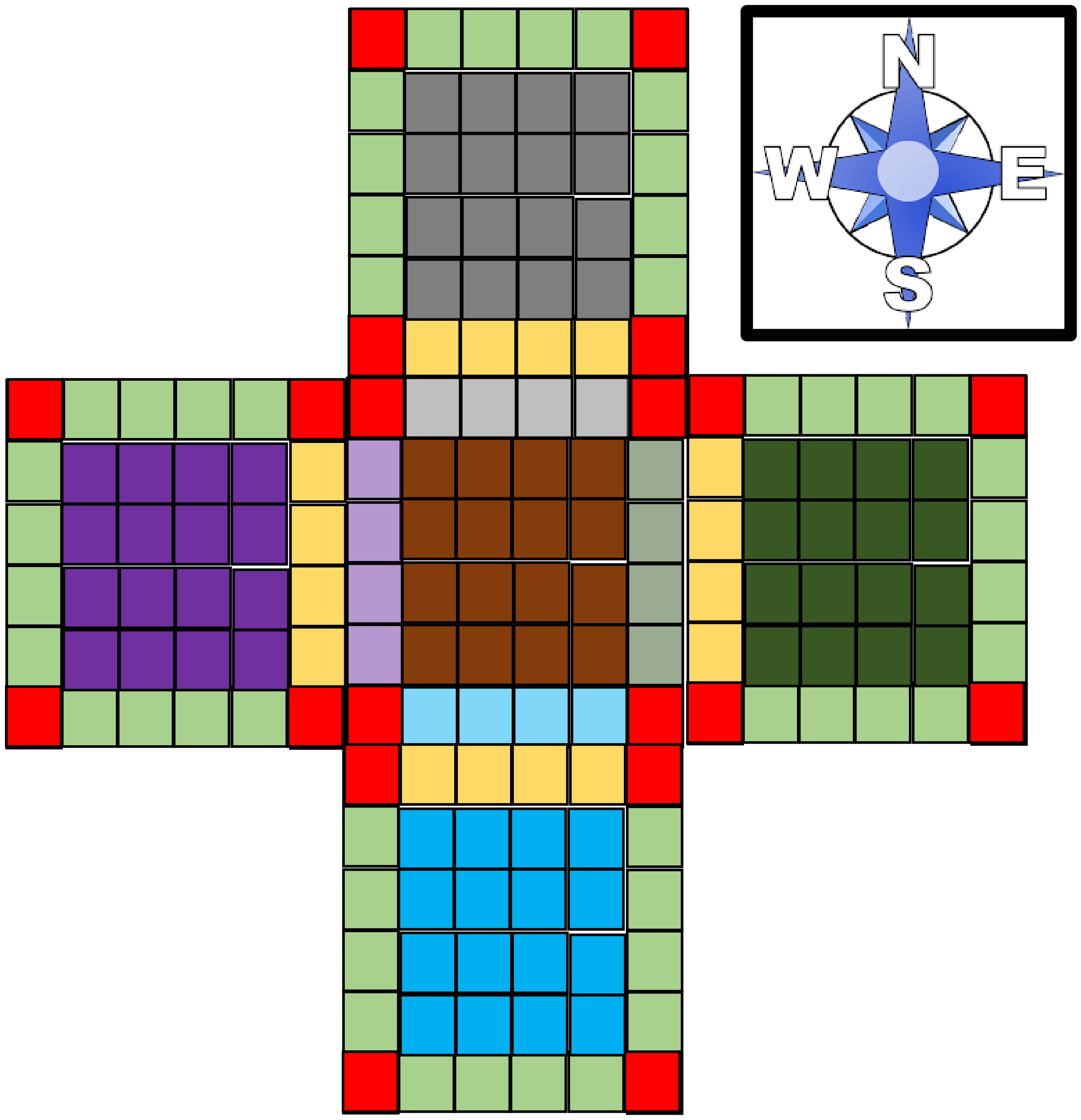
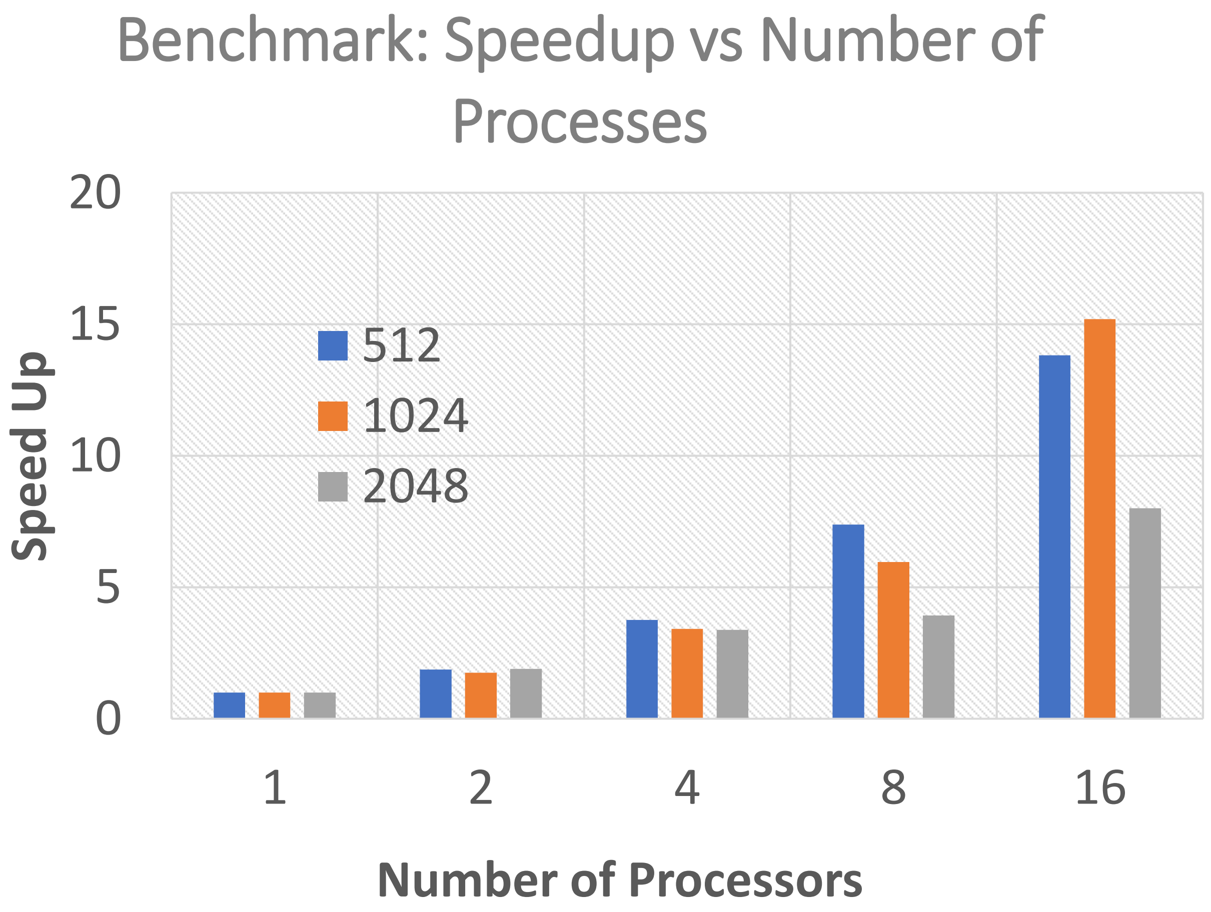


Figure 5: Communication between processors

In the figure 5 the communication is described in more details. For example, the data of the elements at the right edge the violet box will be transferred to the light violet elements which at one column distance from the source column. This an example of west to east communication. Now if we observe deeply we will notice this a non-contagious data communication which is in efficient in traditional way. This same problem will occur in case of east to west communication. This happens because the elements are arranged in a row wise manner in C code. To avoid these problems and save time we use mpi type vectors which helps to do fast communication for non-contagious data. However, in case of north south or south north communication we do not need the use of mpi type vector as because if we just send the address for the first element in the row then a simple for loop could be used to copy the data in quick. Because the operations are in same row.

Benchmarking:

For comparing the speed up we ran our code in stampede with different number of processors and different problem size. We have taken the square matrix of 512,1024 and 2048. And taken processors up to 16. The following figure shows the speed up:



The graph shows almost linear speed up for the starting but then the speedup decreases. It is happening because the communication time increases with increment of processors.

Further Development scope:

We do have a lot of scope to develop the code for more speedup. The memory management have some deficiency for which we are taking much memory in the ram and which ultimately also affecting the speed. Also form the problem side the mathematical aspects can be changed to make it more robust.