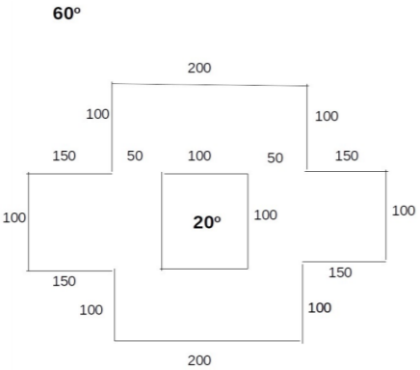
Introduction to Parallel Processing
Home Assignment #2

Lecturer: Dr. Guy Tel-Zur
Topic: Parallel Computations with MPI
Goal: Solve 2D heat equation – Laplace equation steady state
Student: Shir Chen 203869698

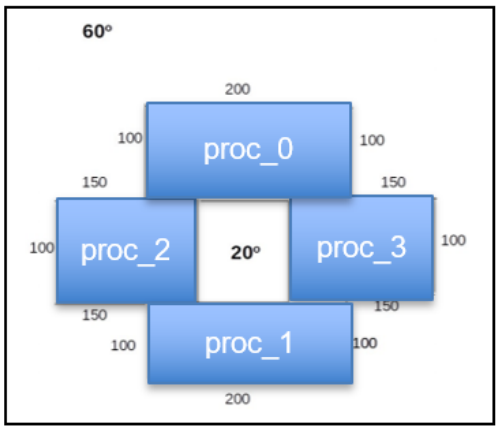
General Explanations:
The Laplace equation applies to many physical problems, such as temperature.
For temperature, it is the steady state heat equation.
In the Laplace equation each point is the average of its neighbors. Therefore, we can iteratively converge to that steady state by repeatedly computing new values at each point from the average of neighbors' points. We keep on iterating until the difference is small enough for us to tolerate. This is the Jacobi Iteration method:

$$M_{k+1}(i,j) = \frac{M_k(i-1,j) + M_k(i,j-1) + M_k(i+1,j) + M_k(i,j+1)}{4}$$

Is this code we solve the heat equation for the following geometric figure problem:



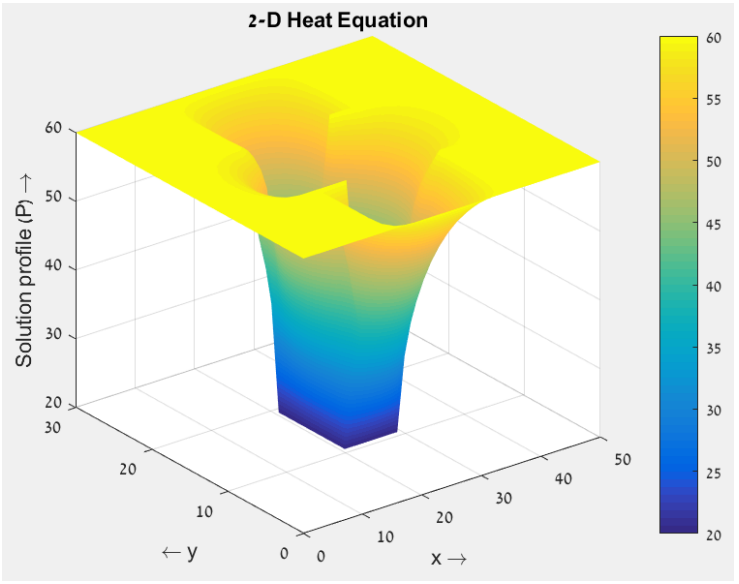
We were asked to use only 4 processes, due to that requirement I selected to use the following domain decomposition:

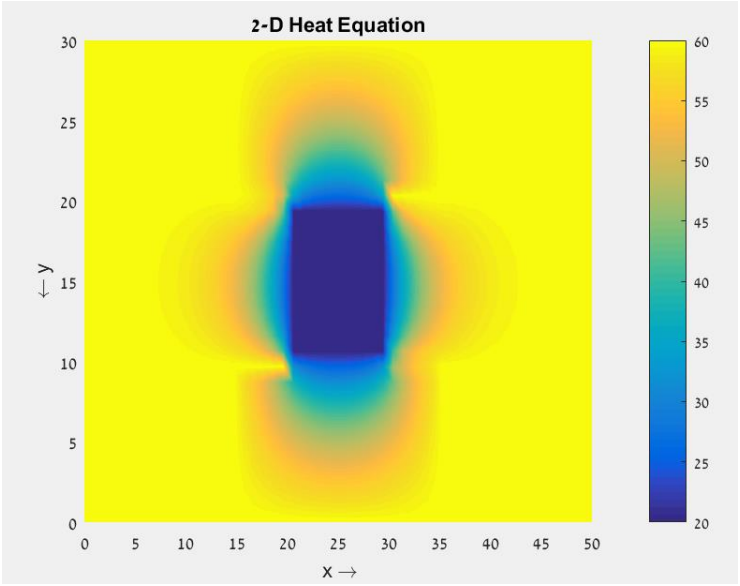


Using this domain decomposition, each process is calculating the same data amount, $200X100$ matrix, and each process must communicate with two other processes but only $50X1$ array data is transferred between them.

Instructions How to Compile and Execute:

1. On a linux machine run: xhost +
   this command enables your connection to the hobbit node (adds you to the list allowed to make connection to the X server)
2. Next connect to the hobbit with your username with the following command:
   ssh -X (username)@hobbit(xx).ee.bgu.ac.il
3. Move to the c file folder: cd (your folder)
4. MPI compile your file: mpicc -o (exefilename) (yourfilename).c
5. MPI run the created executable file: mpirun -np (numofproc) ./(exefilename)
6. The code prints every iteration current error and when we reach the required number of iterations or the error is less than what we expected it update an output txt file with the final heat data. The file outputMatrix.txt can be found in the same directory.
7. The outputMatrix.txt file can be read by MATLAB program and show the heat map on a 3D axis using the surf function. The MATLAB code is also attached to this assignment.

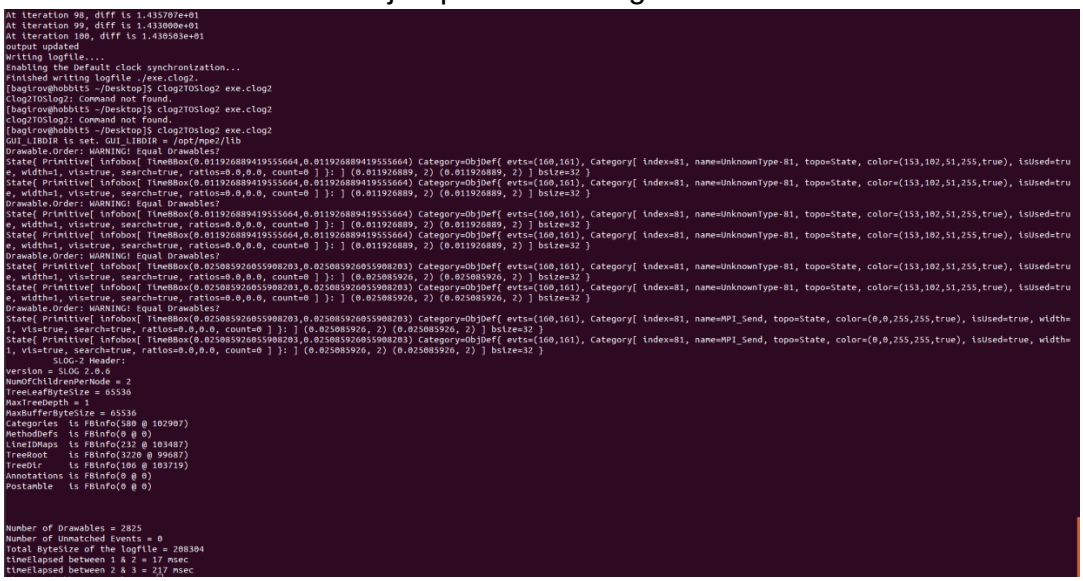




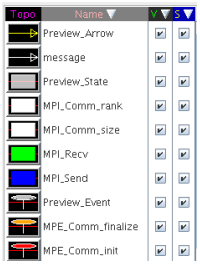
Code tracing and profiling using Jumpshot:

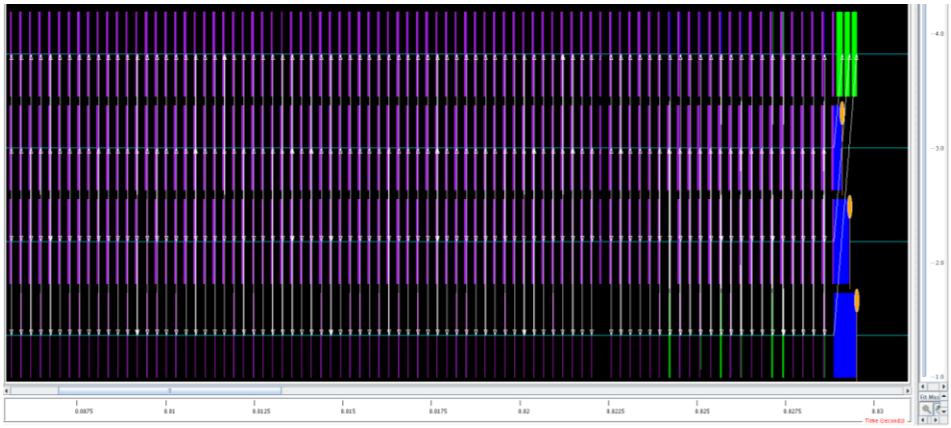
Running Jumpshot:

1. mpecc -mpilog main.c -o exe
2. mpirun -n 4 ./exe
3. clog2TOslog2 exe.clog2
4. jumpshot exe.slog2



- We can see that we have many messaging between all the processes.
- Every iteration each process communicates with two other processes. It sends and receives the 50X1 array of the neighbors' points that are not included in the matrix of the current process.
- When all iterations are completed the three workers processes sends their final calculated matrix to the master process. Therefore, we see MPI_Recv only for the master process and MPI_Send for all the other workers processes.

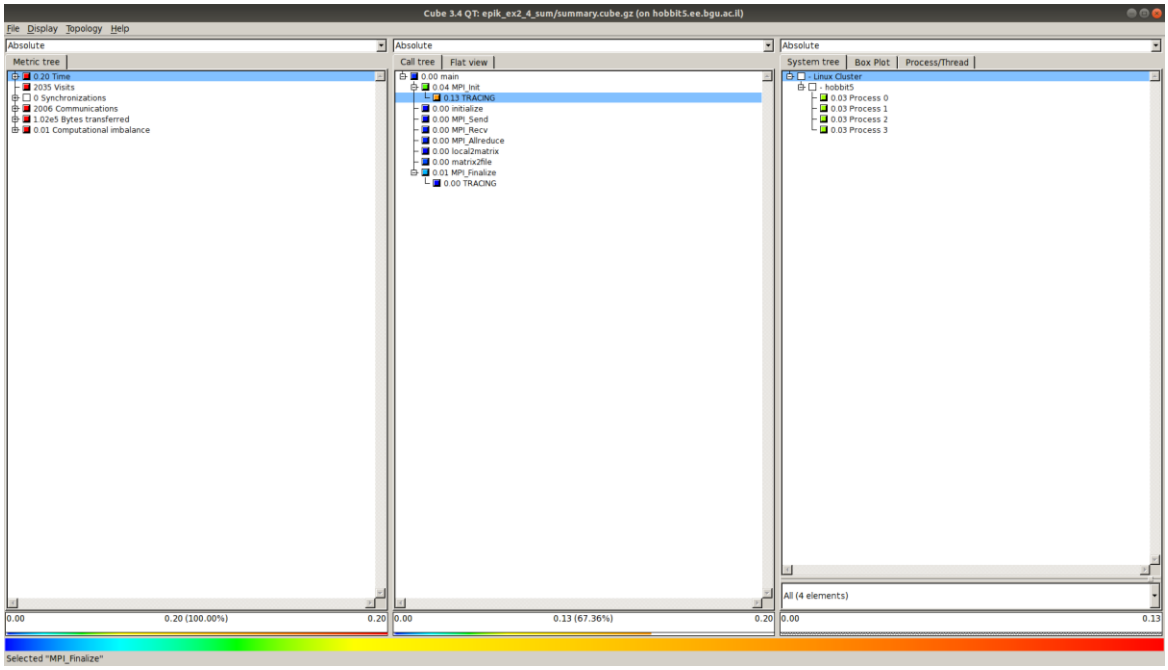




Code tracing and profiling using Scalasca:

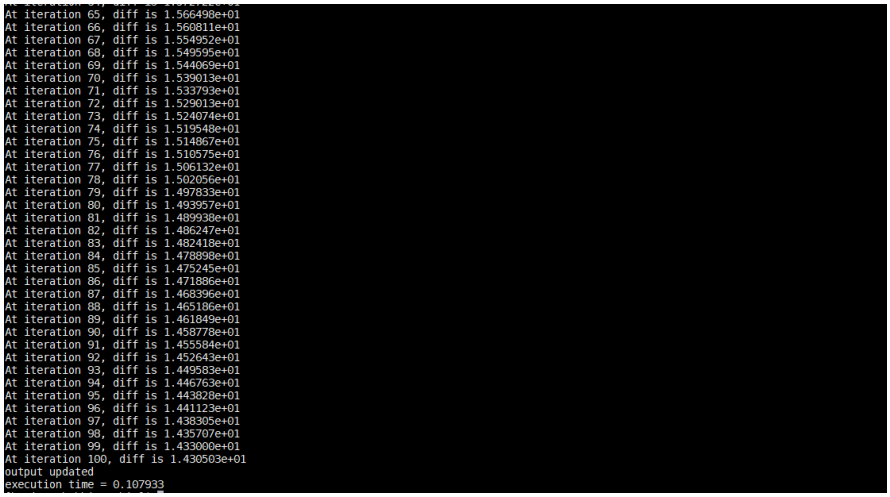
Running Scalasca:

5. scalasca -instrument mpicc -o exe ./main.c
6. scalasca -analyze mpirun -np 4 ./exe
7. scalasca -examine ./epik_exe_4_sum



- Respectively MPI_SEND and MPI_Recv takes almost no time at all.
- On the other hand, MPI_Init is being done by all processes and has significant executable time.

Timing and Conclusions:



The execution time is 0.107933 seconds, meaning it took only about 108 ms to perform the total iterations of calculations and data transformation.

As the number of parallel processes increase the run time of the calculation drops. In addition, the domain decomposition allows me to transfer relatively small arrays, so the messaging time is also not high. The last send and receive of the total data calculated by each process is the only significant amount of data transferred between the processes. We can also see that in the code tracing using jumpshot.