Documentation

# Parallelization Strategy

Heat2D\_mpi.java is a parallelized java program utilizing java MPI Communication strategy. User provides the number of computing node, size of the 2D grid for the heat simulation, time iteration size, time iteration size of heating, and the number of printing cycle. As illustrated in the figure 1, based on the given number of computing node, each node creates the portion of 2D grid to start heating simulation.

Then, each computing node has its own stripe. After this point, as illustrated in figure 2, the simulation starts by padding the grid. In a big picture, top, bottom, left, and right edges will be padded, so that the cells contain the same value as their immediate cells.

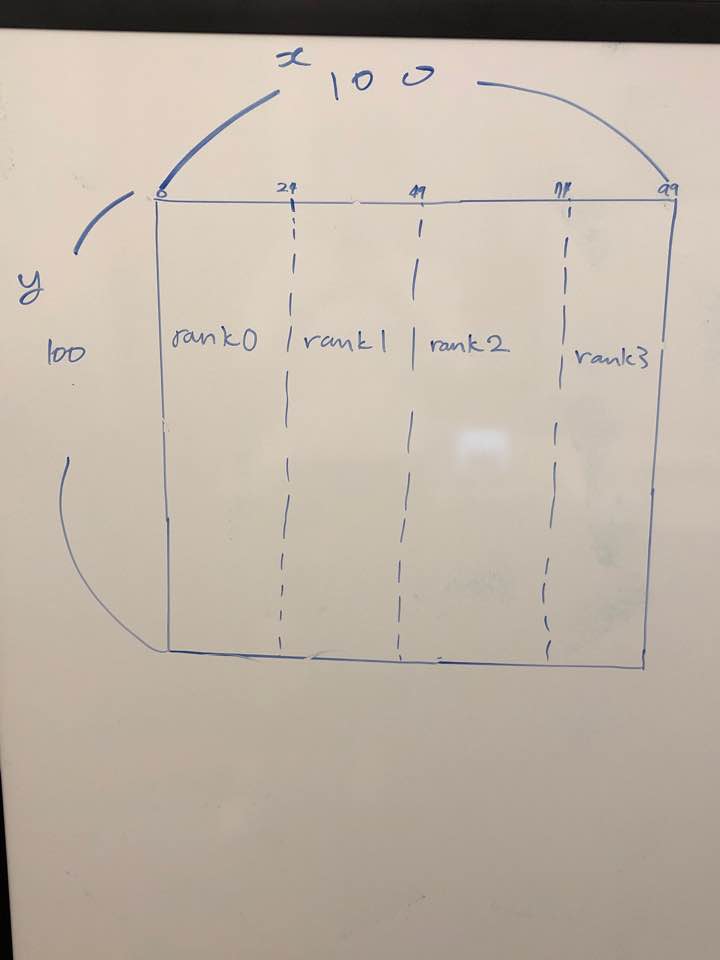
After padding stage, as illustrated in figure 3, the grid will be exposed to the heat only the portion of the top of the grid’s indices from one third of X size to the two third of the X size.

Before calculating the heat diffusion simulation, boundary of each stripe should have values of the boundary of their neighbors. Therefore, as illustrated in figure 4, each computing node sends and receives boundary information with their neighbors. In this step, even nodes send first and receive from the odd rank nodes. In this manner, the simulation can avoid deadlock situation. The leftmost and rightmost stripes are taken care to receive only one boundary to avoid exceptions.

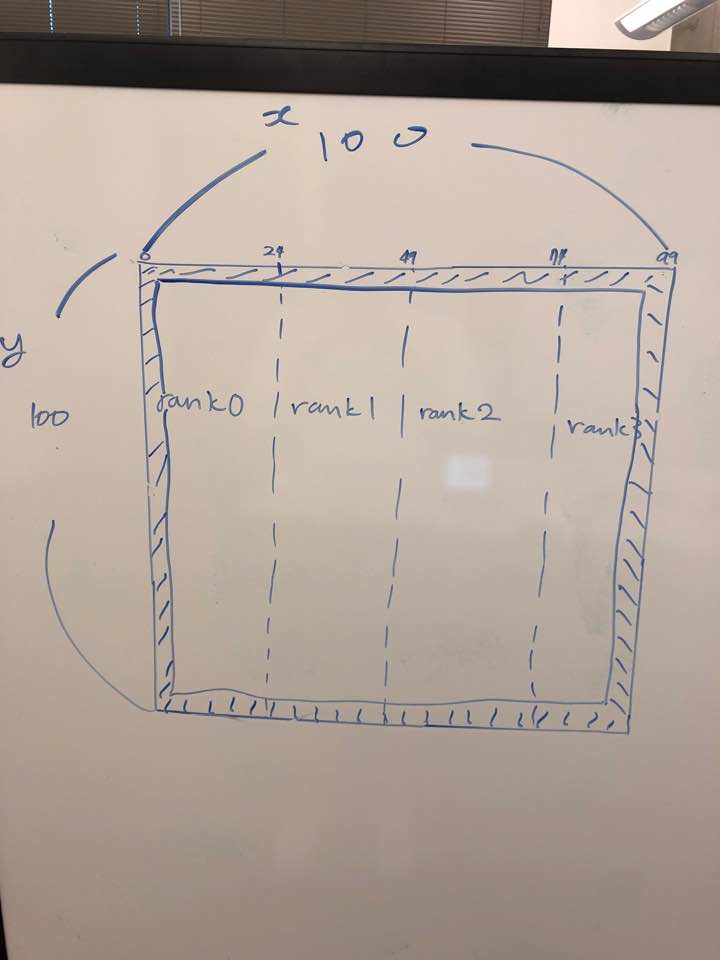
Before computing heat simulation, for every given interval, computing nodes communicate with rank zero to print out the current simulation. The zero ranked computing node is in charge of printing the current status to the console. After printing, calculation of the heat diffusion simulation (Forward Euler Method) starts in each computing node with the exchanged boundary cell data.

Above steps from padding to calculation are repeated for the given number of iterations.

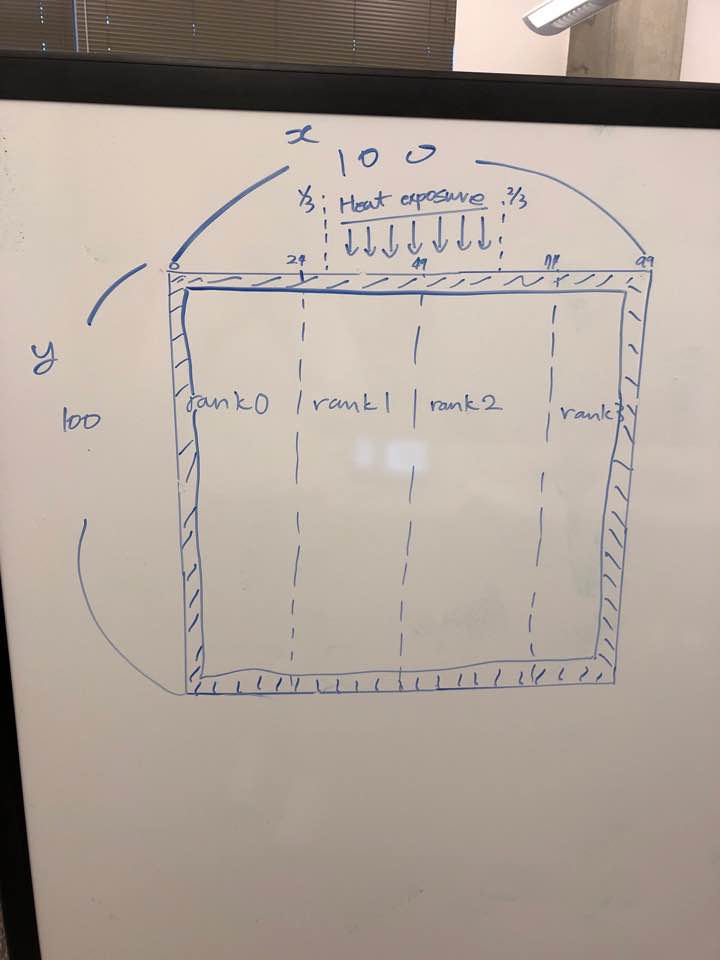
**Figure 1. Illustration of the Heat Grid Division.**



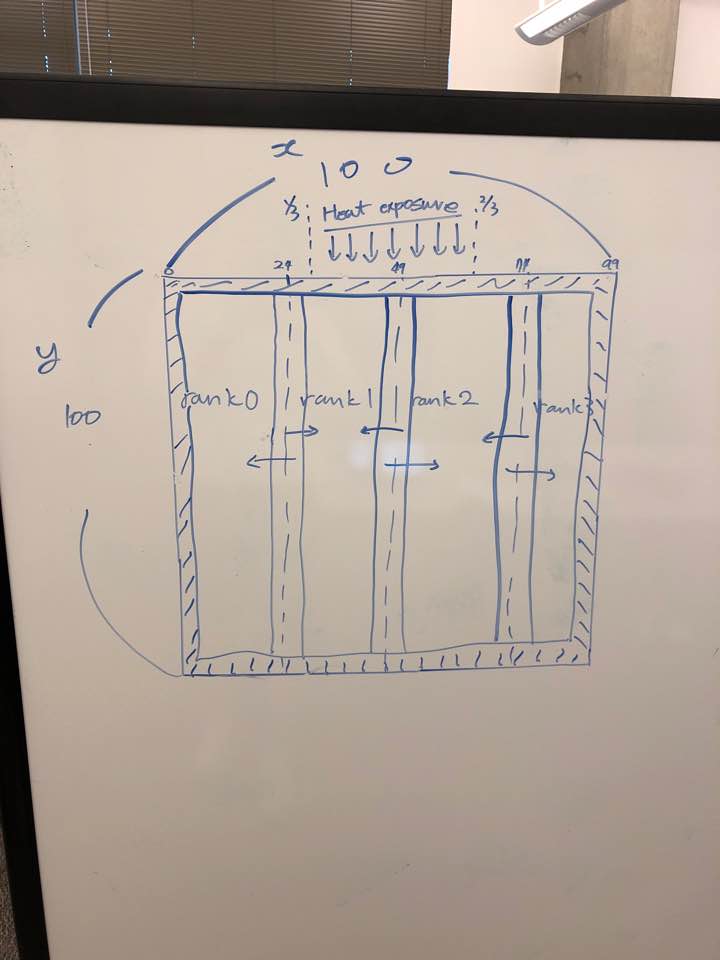
**Figure 2. Illustration of Padding**



**Figure 3. Illustration of Heat Exposure**



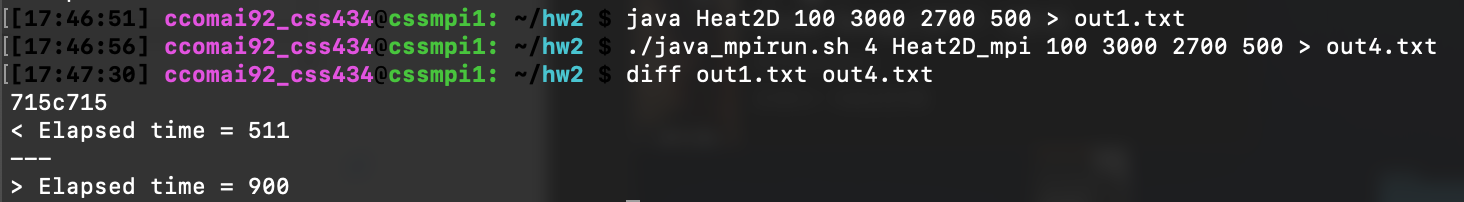
**Figure 4. Illustration of Exchanging Boundary Data**



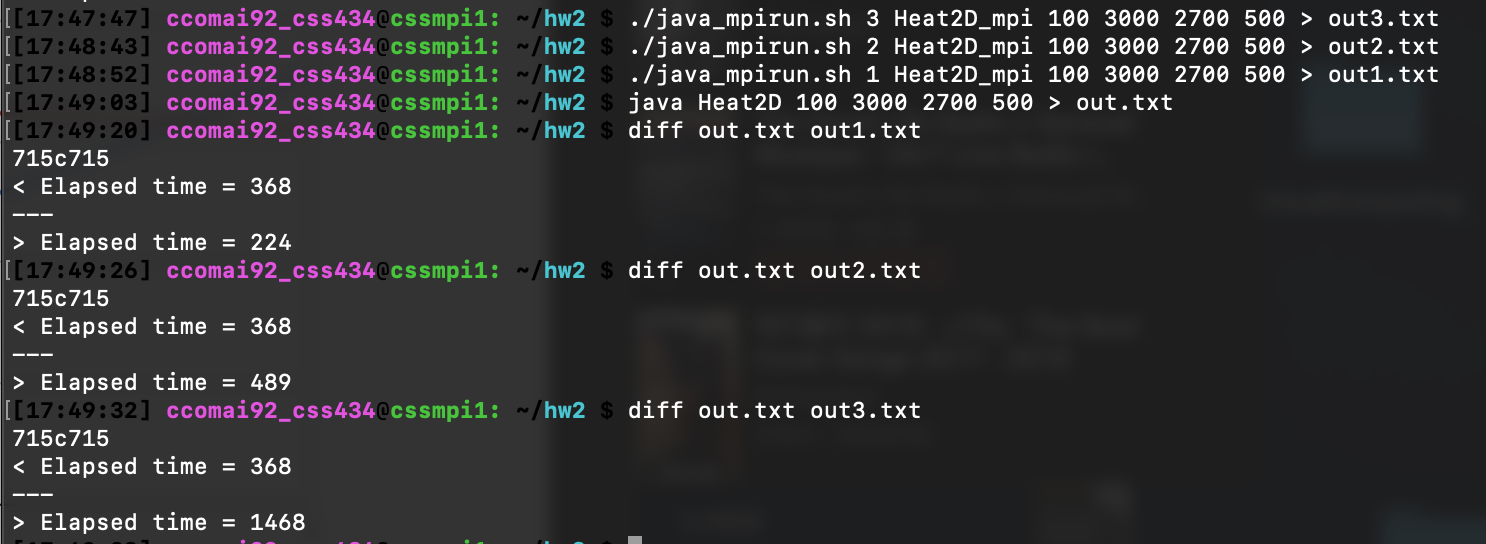
# Output Correctness Proof

In figure 5, difference of output text files, out1 and out4, is only the elapsed time. Therefore, the output of the parallelized simulation is correct. Moreover, as illustrated in figure 6, out1, 2, 3 also have the same result as the comparing with out4. Finally, the parallelized simulation program is efficiently tuned up in that the simulation with one computing node has faster elapsed time compare to the given Heat2D program.

**Figure 5. Output Correctness Verification 1.**

****

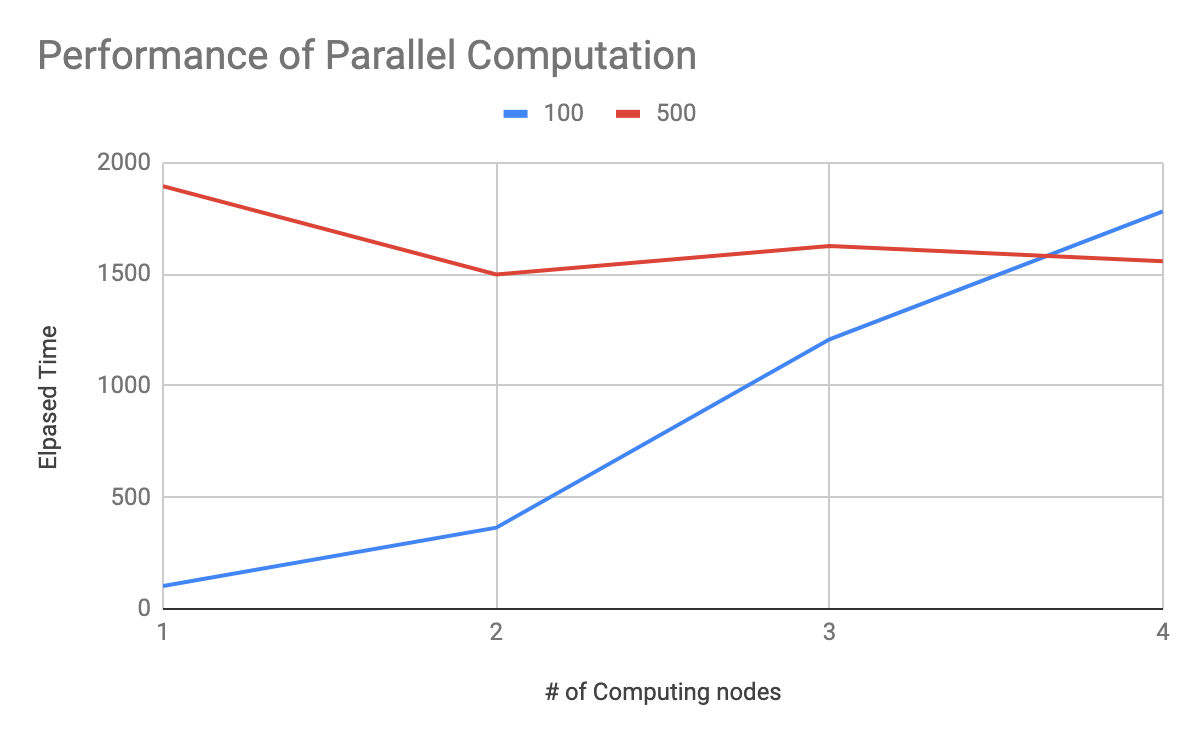
**Figure 6. Output Correctness Verification 2.**



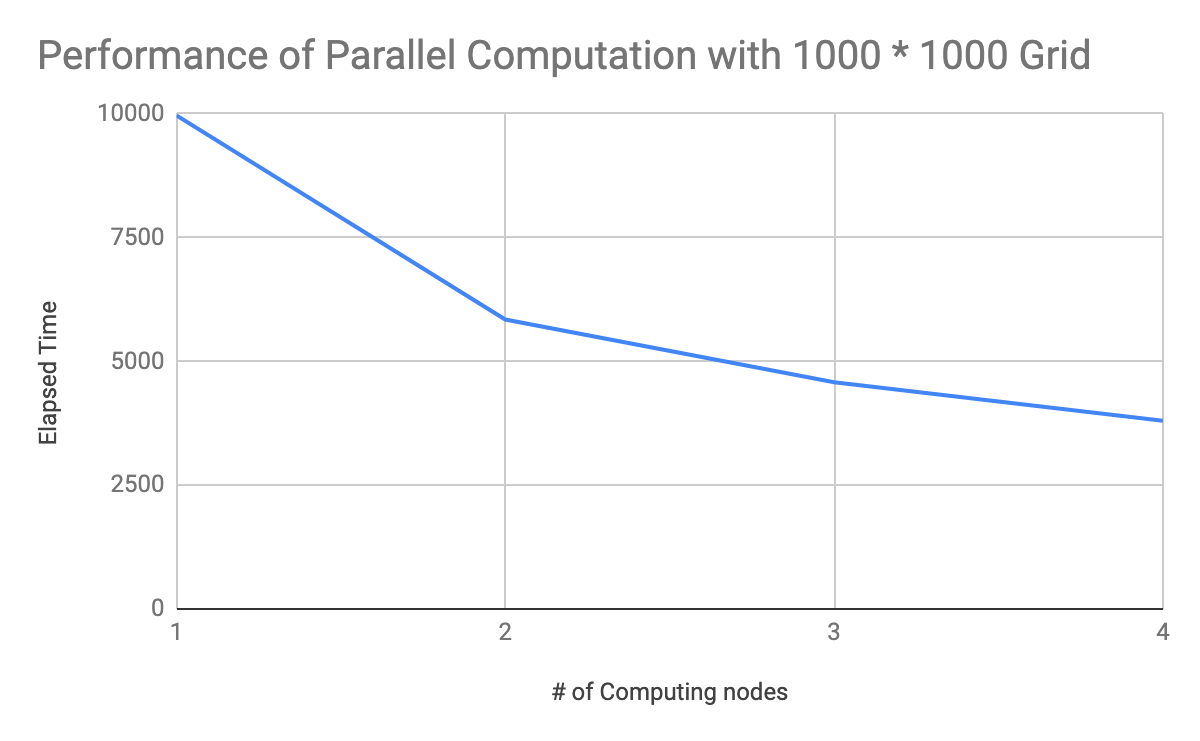
# Performance Analysis

As illustrated in figure 7, the performance of the parallelized heat simulation is different than expectation when the size of the grid is less than 1000. However, as illustrated in figure 8 and 9, expected performance is achieved when the size is bigger or equal to the size 1000. As shown in figure 10 to 13, the analysis graph was drawn based on the data generated by running parallelized program with different input values.

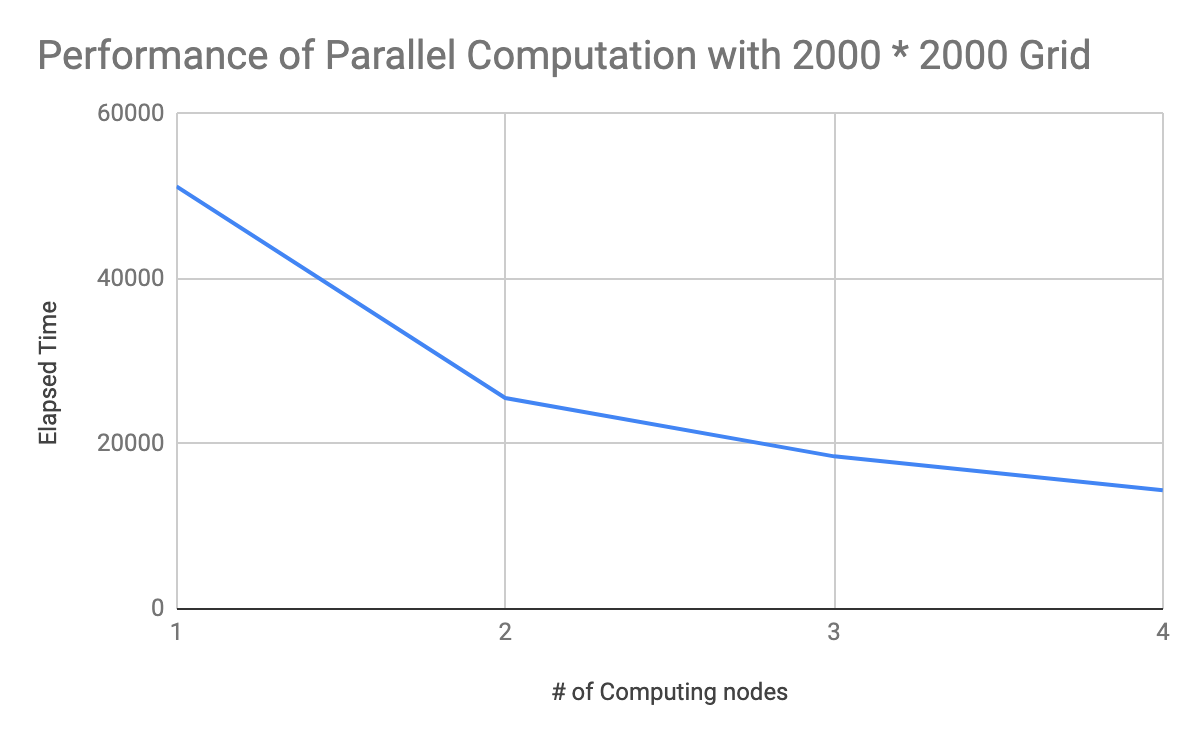
**Figure 7. Performance of Parallel Computation with size less than 1000**



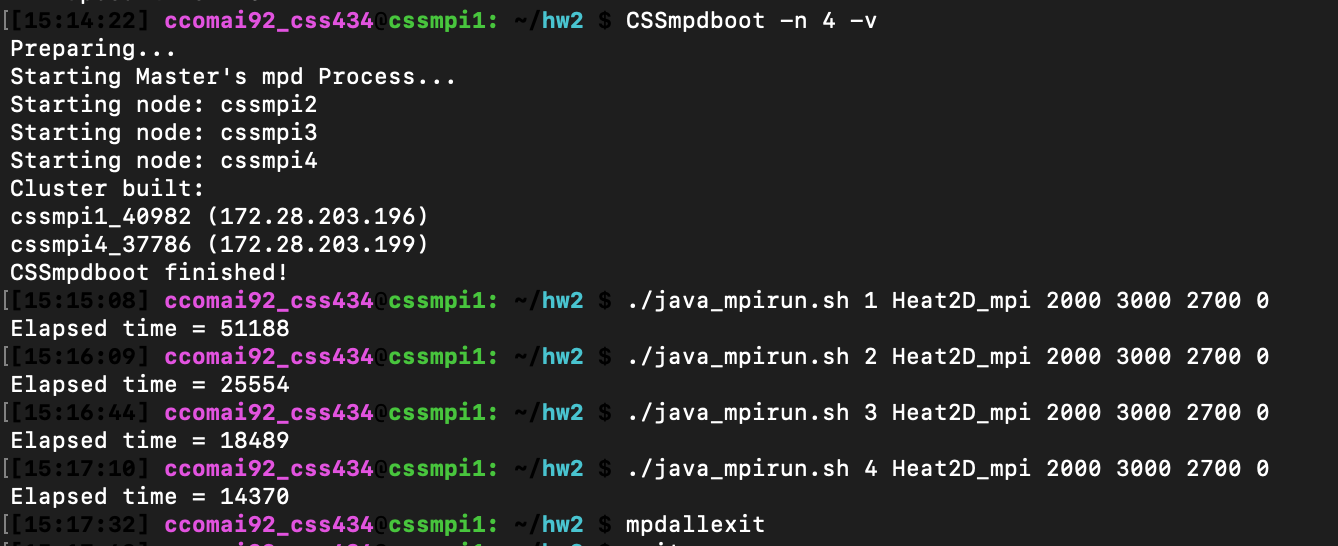
**Figure 8. Performance of Parallel Computation with size 1000**



**Figure 9. Performance of Parallel Computation with size 2000**

****

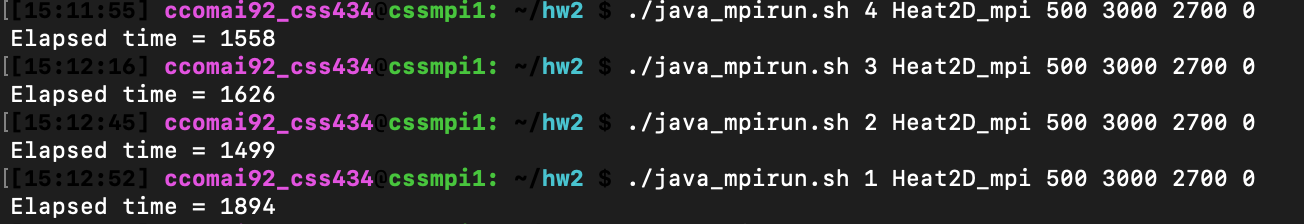
**Figure 10. Elapsed time for the simulation with size 2000 of 2D grid**



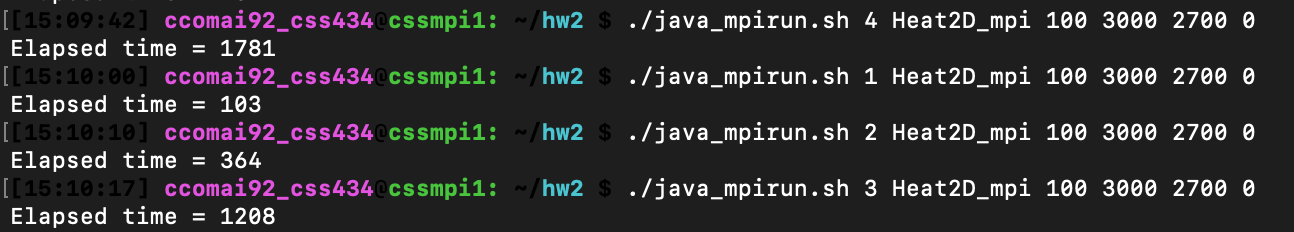
**Figure 11. Elapsed time for the simulation with size 1000 of 2D grid**



**Figure 12. Elapsed time for the simulation with size 500 of 2D grid**



**Figure 13. Elapsed time for the simulation with size 100 of 2D grid**



# Discussion

* Current Limitations and performance problems
  + As mentioned in the performance analysis, parallelized program does not take advantage of the parallelization when the size of the grid is smaller than 1000 due to the network latency and the overheads. Otherwise, the performance graph is showing the expected shape of the curve.
  + In this example, using three computing nodes is the only odd number division which generates the remainder stripes. Therefore, the implementation just added the last one remainder at the end of the last computing node. However, this could be problematic when the number of computing node is odd and not divisible by three. In such cases, last computing node could have more cells, not just one line, for their computation. As a result, having the last computing node would not affect much on the performance of the computation.
* Possible performance improvement
  + First limitation cannot be improved because the overheads are necessary and the network latency is practically fast enough, 1Gbps network.
  + Second, limitation in implementing the program can be improved by allocating remainder grid cells evenly throughout the computing nodes.