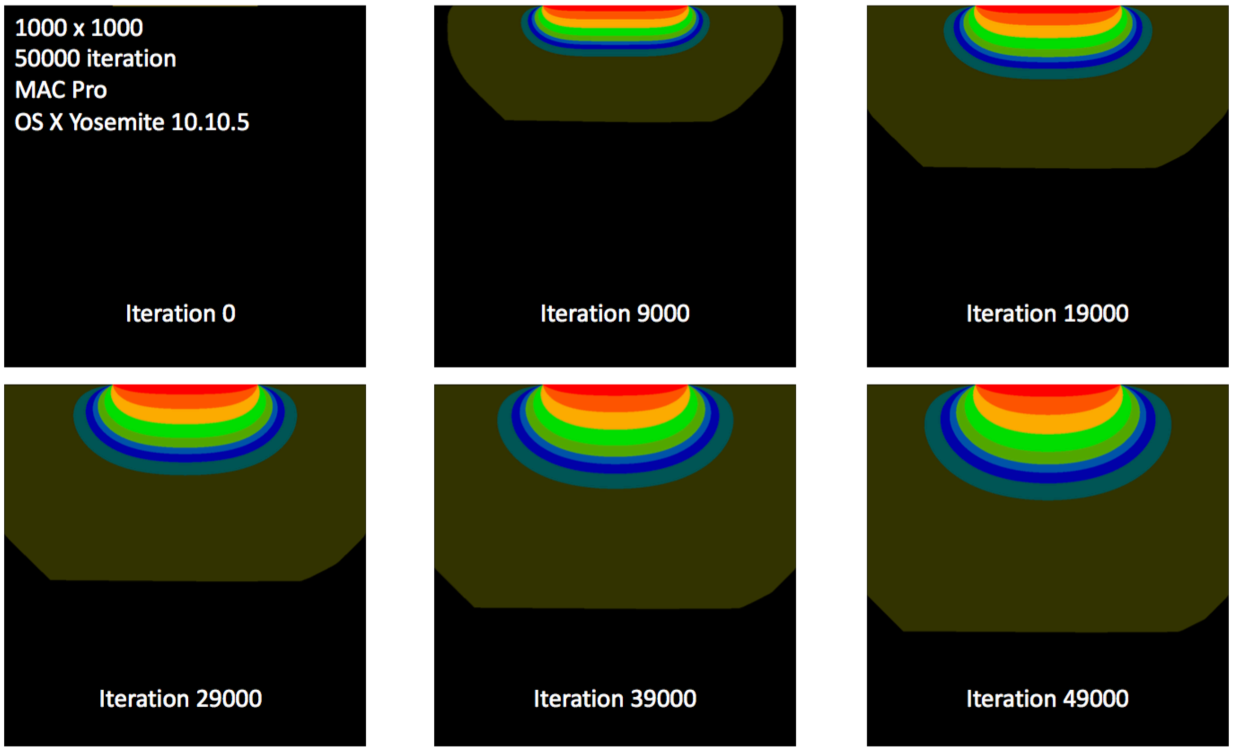
CPSC6770 Assignment 2: Heat Distribution Simulation

Yuzhe Yang

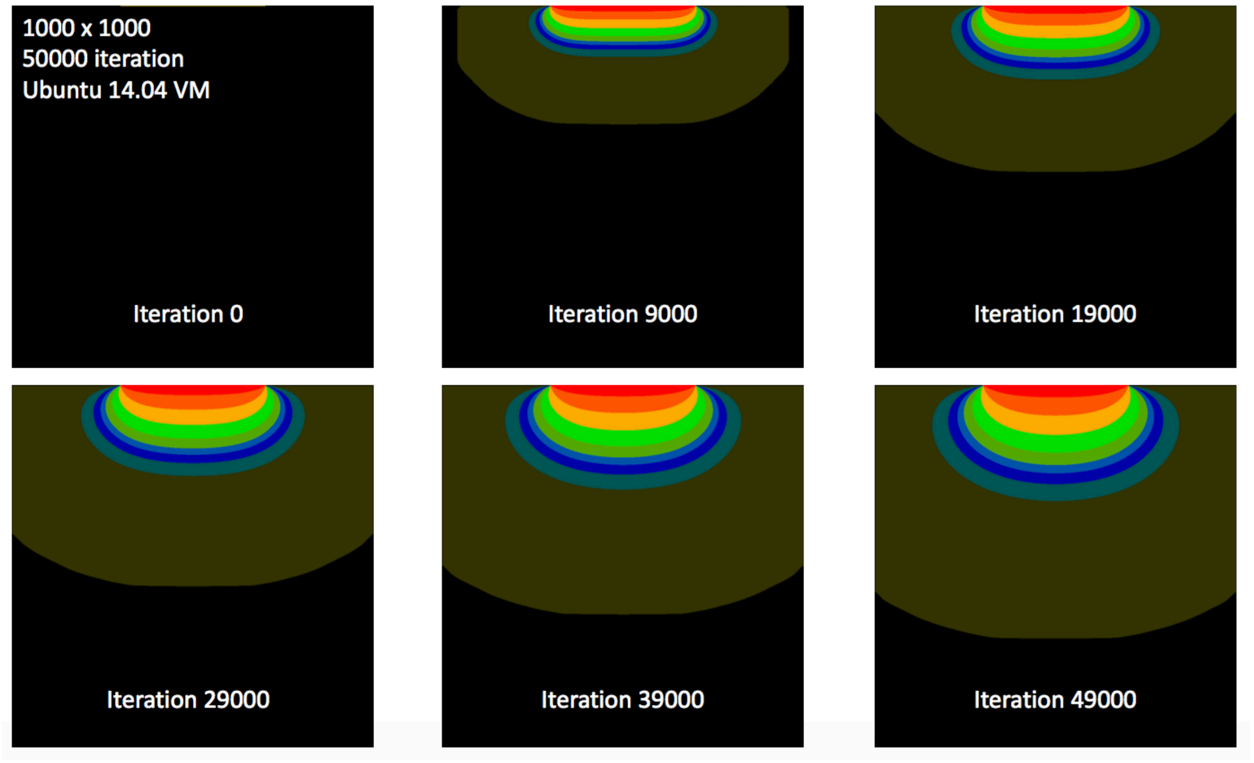
This assignment requires to simulate 2D heat distribution in sequential and parallel approaches.

Sequential Implementation

I first coded a sequential program to solve the simulation problem. The code **printcolor.c** can be found in the **sequential_code** folder (**compile: gcc or g++ printcolor.c; execute: ./a.out**). I composed three functions CopyNewToOld, CalculateNew, and PrintGrid according to the assignment description. The program simulates a space of 1000 pixels wide and 1000 pixel tall. The total iteration is set as 50,000 and the program gives bitmap outputs in every 1,000 iterations. Representative bitmaps are presented in **sequential_heatmap_summary.pdf** file. To note, I executed the same code on my own MacPro as well as using Ubuntu VM. The outputs were slightly different in regard to the boundaries of the 20-degree circle. I am not sure about the reason behind this observation.



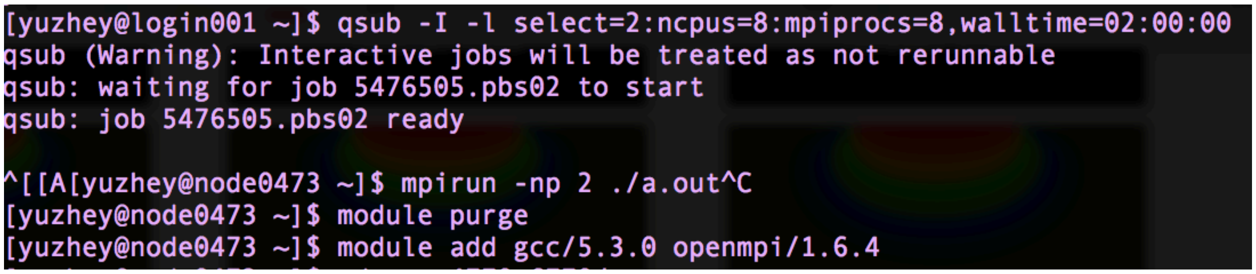
Representative bitmap outputs from sequential program executed on MacPro OS X Yosemite 10.10.5



Representative bitmap outputs from sequential program executed on Ubuntu 14.04 VM

Parallel Implementation

The parallel implementation of heat distribution simulation is coded in **printcolor_mpi.c** in **parallel_code** folder. In order to execute the code, I requested node on palmetto cluster by command: **qsub -I -l select=2:ncpus=8:mpiprocs=8,walltime=02:00:00**. Then I loaded modules by: **module purge; module add gcc/5.3.0 openmpi/1.6.4. To compile: mpicc printcolor_mpi.c. To execute: mpirun -np (numProcs) ./a.out.** To note, openmpi/1.8.1 cannot be used for this mpi program. In assignment 1, I used openmpi/1.8.1 and successfully executed all my mpi codes on Palmetto, however, this time if I loaded openmpi/1.8.1, the programs in assignment 1 cannot be executed anymore. Therefore, I suspected the problem might be on the Palmetto Cluster end and I tried 1.6.4 version which worked well for all my mpi programs.

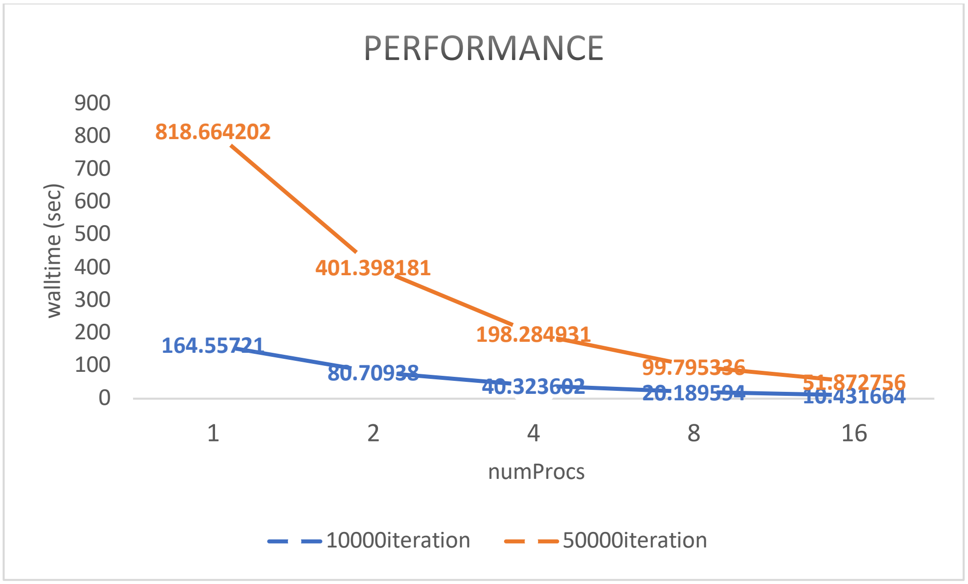


```
[yuzhey@login001 ~]$ qsub -I -l select=2:ncpus=8:mpiprocs=8,walltime=02:00:00
qsub (Warning): Interactive jobs will be treated as not rerunnable
qsub: waiting for job 5476505.pbs02 to start
qsub: job 5476505.pbs02 ready

^[[A[yuzhey@node0473 ~]$ mpirun -np 2 ./a.out^C
[yuzhey@node0473 ~]$ module purge
[yuzhey@node0473 ~]$ module add gcc/5.3.0 openmpi/1.6.4
```

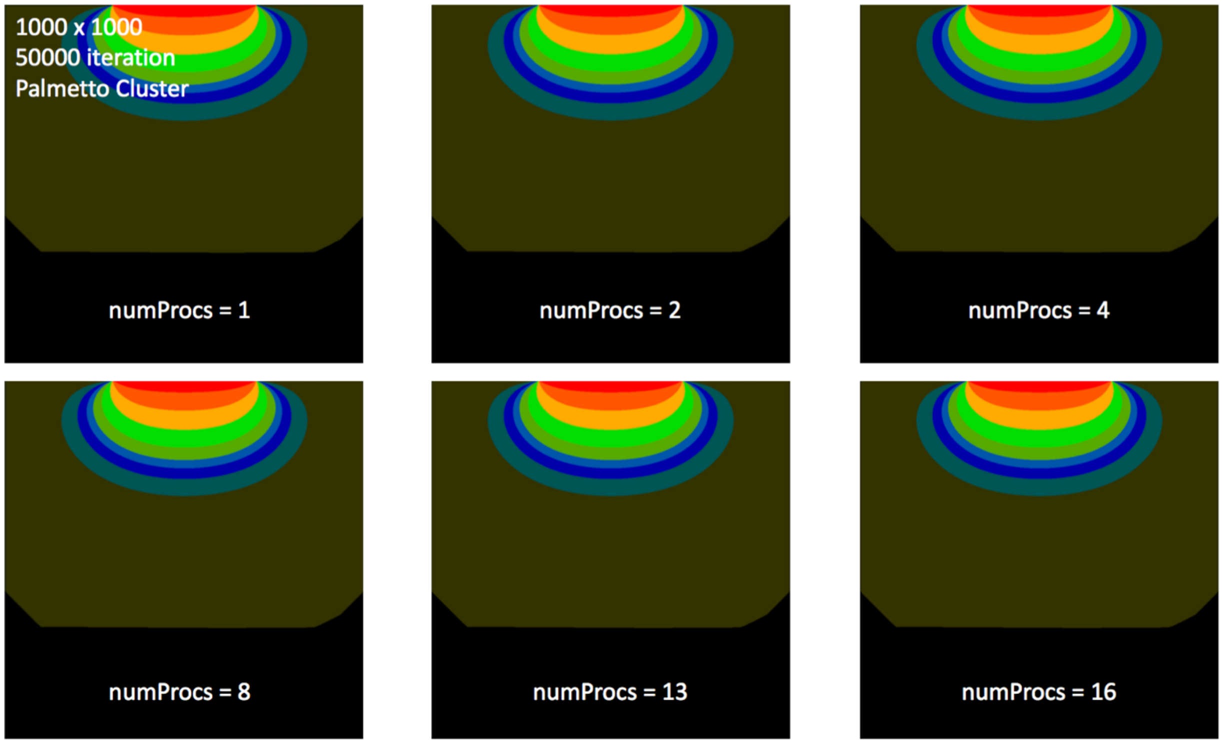
In the **printcolor_mpi.c** code, I equally divided rows to processes with ghost rows as needed. I implemented Send Recv to communicate between master and the rest processes. I tested the performances (runtime) under different iteration (10,000 and 50,000) and number of processes (1, 2, 4, 8, and 16) conditions and generated graph in **heat_distribution_parallel_performance.xlsx** in **parallel_code** folder. The graph shows that for different number of processors used, the time for 50,000 iterations is just five times of that for 10,000 iterations, indicating the time increase with iterations is linear. Within the same iteration number, when the number of processors doubles, the processing time decreases in half. For example, for 10,000 iteration, one processor needs 165 seconds to complete, while two processors need 81 seconds and 4 processors need only 40 seconds. This verifies that the program is running parallel and there is almost no overhead consumption between each parallel process.

The mpi code is also capable of handling situations when the number of rows cannot be divided evenly by the number of processors. Representative final bitmap outputs by different numbers of processes are presented in **parallel_heatmap_summary.pdf** in the **parallel_code** folder.



Performance summary of parallelization.

1000 pixels x 1000 pixels		
	10,000 iteration	50,000 iteration
numProcs	walltime (sec)	walltime (sec)
1	164.55721	818.664202
2	80.70938	401.398181
4	40.323602	198.284931
8	20.189594	99.795336
16	10.431664	51.872756



Representative bitmap outputs from parallel program executed on Palmetto Cluster