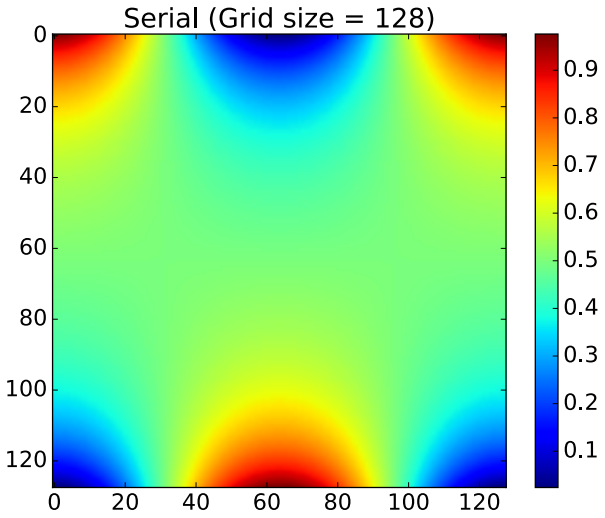
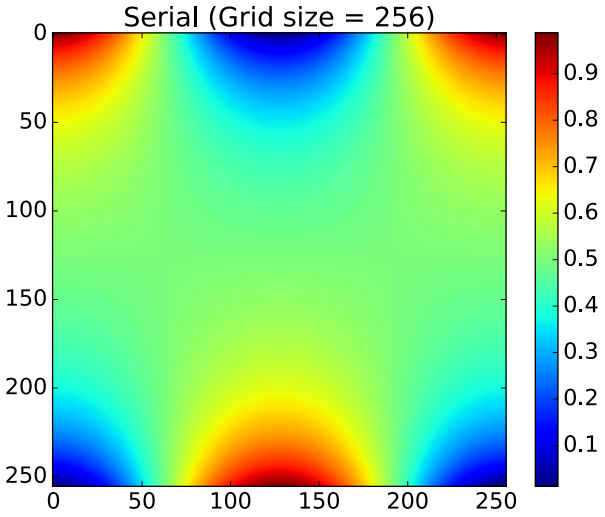
Summary

Figure 1 below shows the simulation results from heat_serial. It is seen that there is not much difference when varying grid size from 128 to 512, indicating that the size of 128 is already (visually) fine enough.





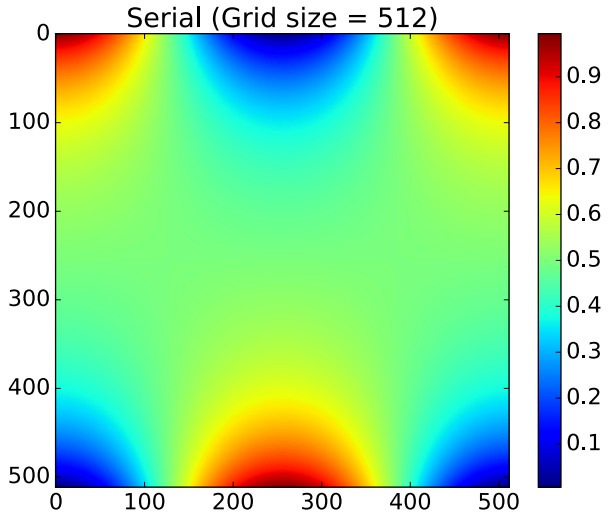


fig 1.

The average temperature for the serial case as well as the MPI case is reported in table 1. It is seen that it is largely independent of the parallelization method used.

Table 1.

Temperature		grid size 128	256	512
serial		0.496150	0.498060	
MPI	1			
	2	0.499975	0.499976	0.499976
	4	0.499973	0.499975	0.499976
	8	0.499969	0.499973	0.499975
	16	0.499959	0.499969	0.499973

The time consumed by each process is tabulated in table 2. For better comparison, a series of plots of the data in table 2 are also presented (Fig. 2 - 4). It should be noted that some of the data points in Table 2 are missing. This is because I don't have enough time to run those cases that are the most time-consuming. However, the remaining data would still illustrate the points.

Fig. 2 shows that OpenMP technique is able to reduce the percentage of time by ~80% from using 2 cores to 8 cores. If one looks at table 2, especially for the case of a grid size of 512, the reduction of time needed is remarkable (from ~50 min to ~12 min by using 8 cores instead of 2 cores).

As for the MPI technique, there is a plateauing trend for all 3 grid sizes, and the speed up is size-dependent, which seems to be less efficient than OpenMP. However, if one compares the actual time elapsed (Fig. 4), one will readily observe that MPI already offers a huge reduction of

time at very few cores (e.g., 2 cores). For the case of 512*512 grid points, the total time needed in OpenMP is 5 times more compared to the case of MPI when using 2 cores.

Table 2.

Time, s		grid size 128	256	512
serial		22.1749		3710.2703
openMP	1	23.8931		
	2	12.0190	189.6055	3009.5733
	4	6.1866	121.9515	1530.1537
	8	3.1471	48.8818	771.4350
MPI	1			
	2	3.3505	35.8500	561.4139
	4	2.3313	18.5579	292.4338
	8	2.0049	10.4016	171.7582
	16	1.8784	7.1850	96.4750

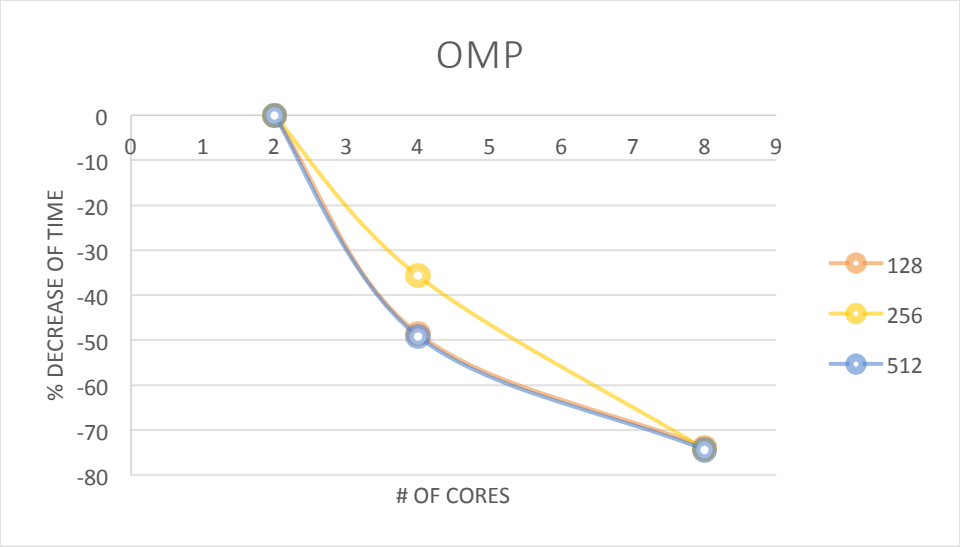

OMP
0
-10
-20
-30
-40
-50
-60
-70
-80
0 1 2 3 4 5 6 7 8 9
% DECREASE OF TIME
# OF CORES
128
256
512

fig. 2.

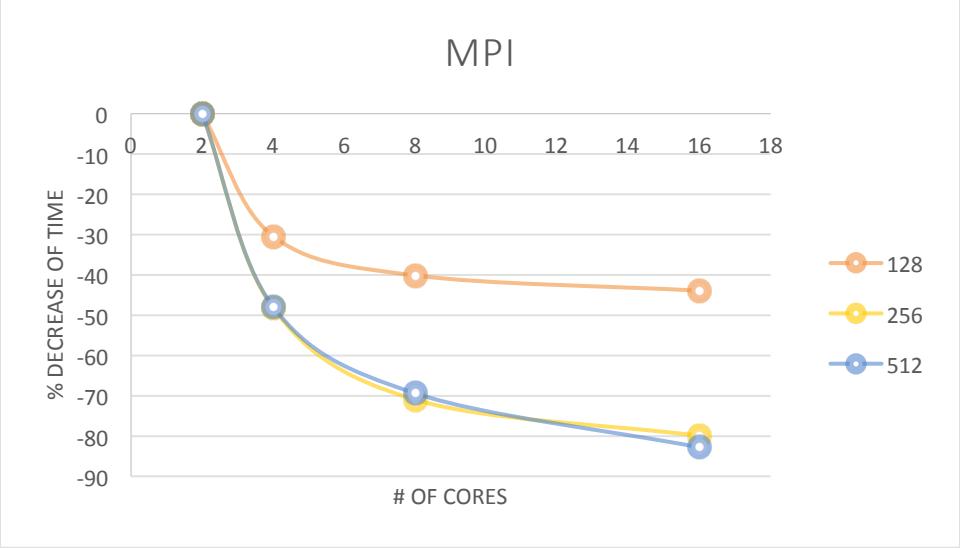

MPI
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
0 2 4 6 8 10 12 14 16 18
% DECREASE OF TIME
# OF CORES
128
256
512

fig. 3.

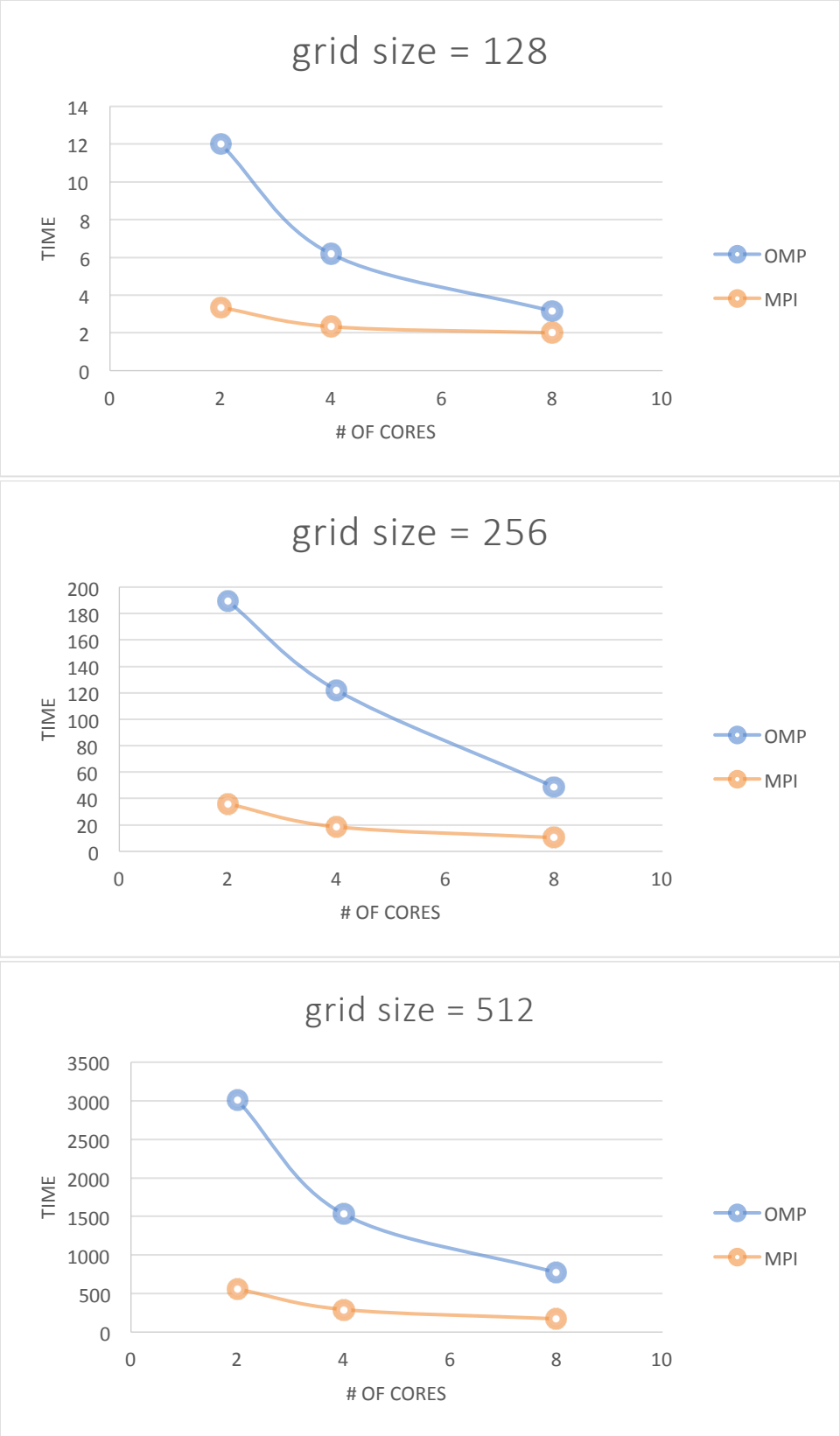

grid size = 128
TIME
OF CORES
OMP
MPI
grid size = 256
TIME
OF CORES
OMP
MPI
grid size = 512
TIME
OF CORES
OMP
MPI

fig. 4

the data generated by heat_mpi with the grid size of 128 is plotted as fig. 5. The case of 256 and 512 grid size is not shown due to their similarity to the 128 grid size case. The way I handle the I/O in MPI is to output the small chunk of data by each processor, and I concatenate and plot them in a separate python script. Part of the reason of not outputting the data to a single file using the "MPI I/O" method is that it is computationally expensive to output ASCII type of data instead of binary, and I don't have a good way to handle binary data on top of my head.

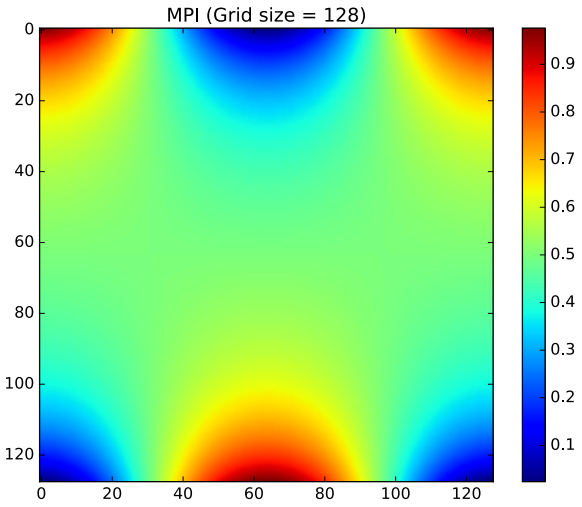


fig. 5.

There are pros and cons for the OpenMP and MPI techniques. The foremost advantage of using OpenMP is its simplicity. It only requires a few lines of codes, which offers a reasonable speedup to run the code. The problem of it is that it does not scale well with the problem size, because there are rarely machines with a huge amount of shared memory. MPI, by contrast, scales much better than OpenMP because it decomposes the problem, and assigns small amounts of jobs to each processor with its own memory. As long as the communication between processors is not a big issue, the speedup by using MPI is generally proportional to the number of decomposed regions (number of cores used). That being said, the disadvantage of MPI is thus the pain to decompose your problem yourself, which makes coding a lot more complicated compared to using OpenMP.