Assignment 2 - HPSC (ME766)
Parallel Matrix Multiplication

By
Naveen Himthani (120010001)

OpenMP Code

- Varied the size of the matrix from 50 to 3000.
-
- Measured the execution time of the code multiple times and took an average of them.
- The time execution was measured using the *gettimeofday()* function in C and the result was quite accurate. I did the sanity check using a stopwatch.
- All times are in seconds.

OpenMP Results

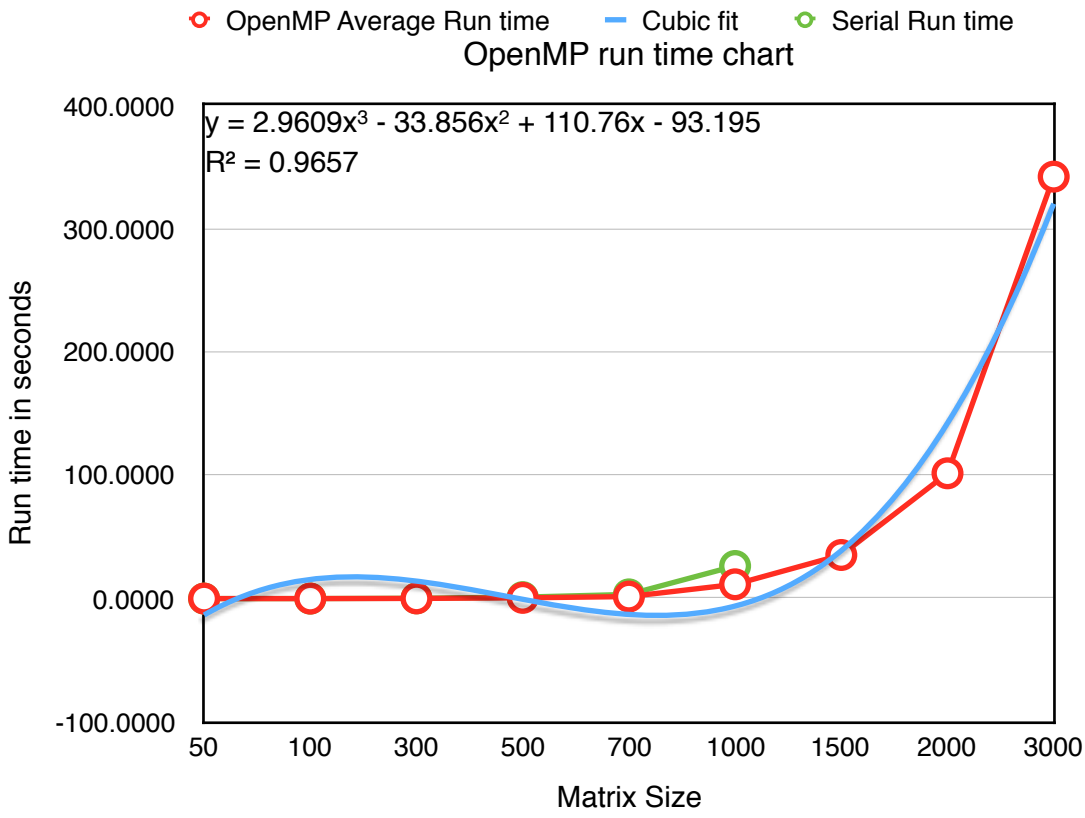
N	Run time 1	Run time 2	Run time 3	OpenMP Average Run time	Serial Run time	OpenMP Speed Up
50	0.0016	0.0019	0.0016	0.0017	0.0011	0.6352
100	0.0065	0.0082	0.0061	0.0069	0.0070	1.0071
300	0.1306	0.1088	0.1662	0.1352	0.2508	1.8550
500	0.6220	0.5999	0.5844	0.6021	1.1788	1.9578
700	1.5534	1.5708	1.7402	1.6214	3.3910	2.0914
1000	11.5341	11.5595	11.5489	11.5475	26.3934	2.2856
1500	34.5433	35.9480	35.4330	35.3081		
2000	102.0000	101.0840	101.2840	101.4560		
3000	341.4380	341.7680	341.8230	341.6763		

Results

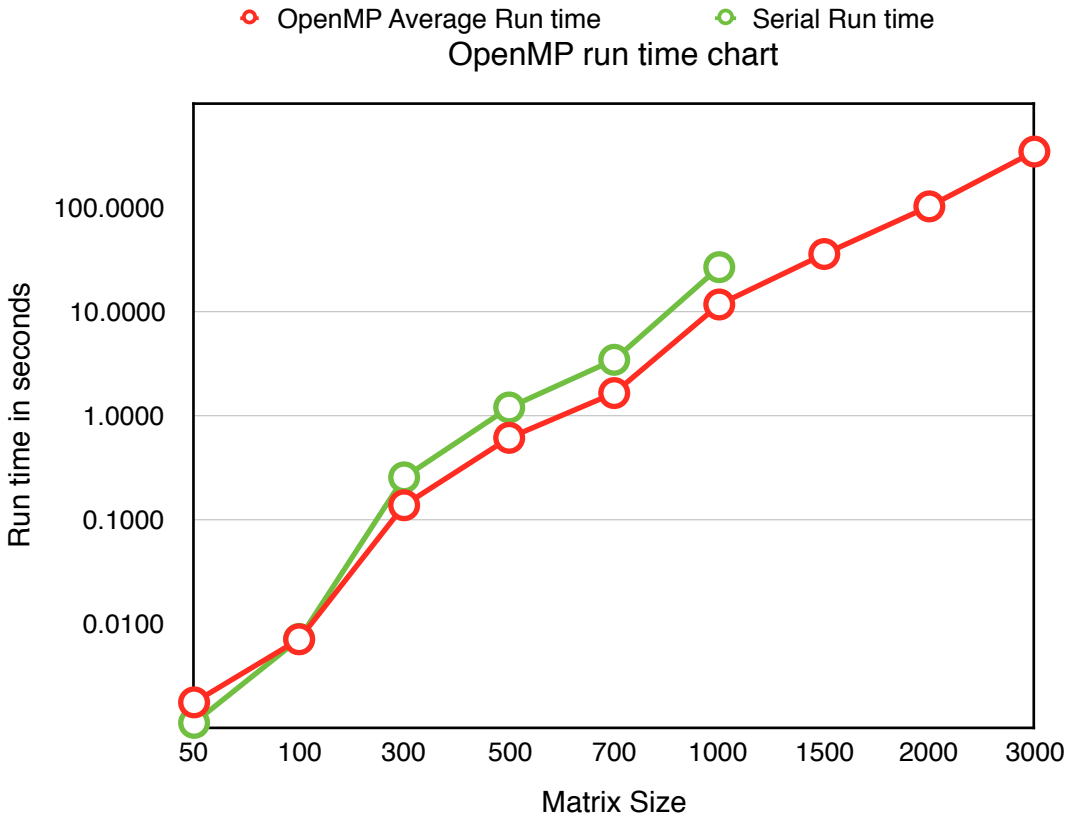
I used these OpenMP directives for computing

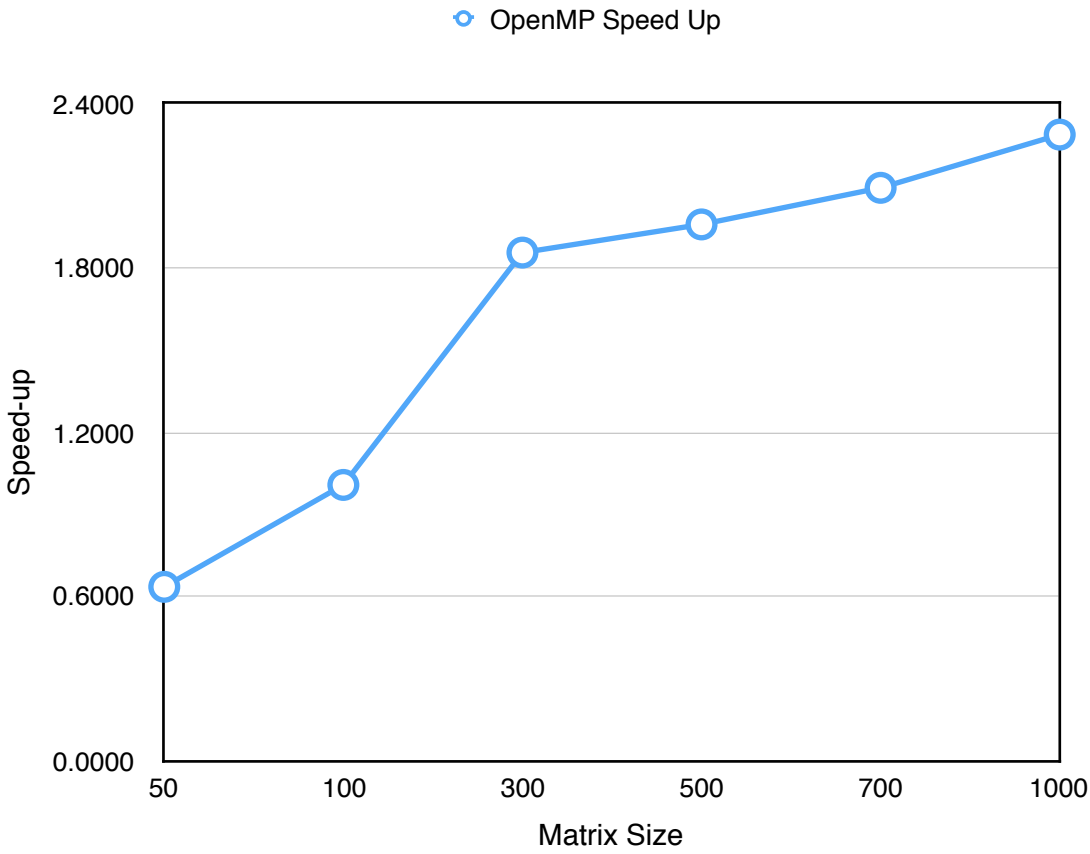
- *#pragma omp schedule (dynamic, chunk)* for initialising the matrices
- *#pragma omp schedule (static, chunk)* for calculating the product

Chunk size is constant and set to 15
The number of threads generated are equal to 4



As expected the OpenMP run time is cubic in nature. The run time difference is not visible in this picture because of the scale on the y axis. Below is a graph with a log scale on y axis.





Above is the speed up graph obtained by OpenMP. We can see that the graph crosses 1 for some matrix size. It indicates that OpenMP has some overhead and is ineffective for matrix sizes below some particular value.

MPI Code

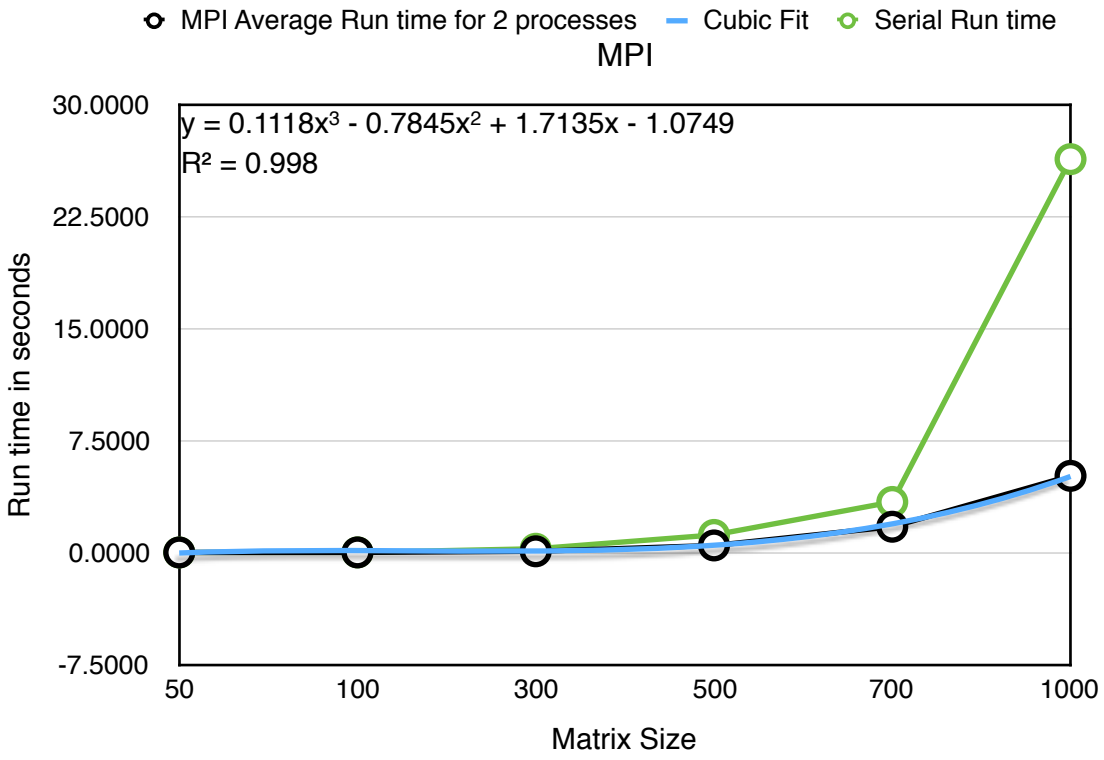
- *MPI_Gather() & MPI_Scatter()* were used to receive and send chunks of data from different processes instead of using multiple *MPI_Receive*() and *MPI_Send()* function calls.
- The code was written such the the matrix size should be divisible by the number of processes, otherwise the code will exit with an error message.
- The Matrix size was varied from 100 to 3400

Results

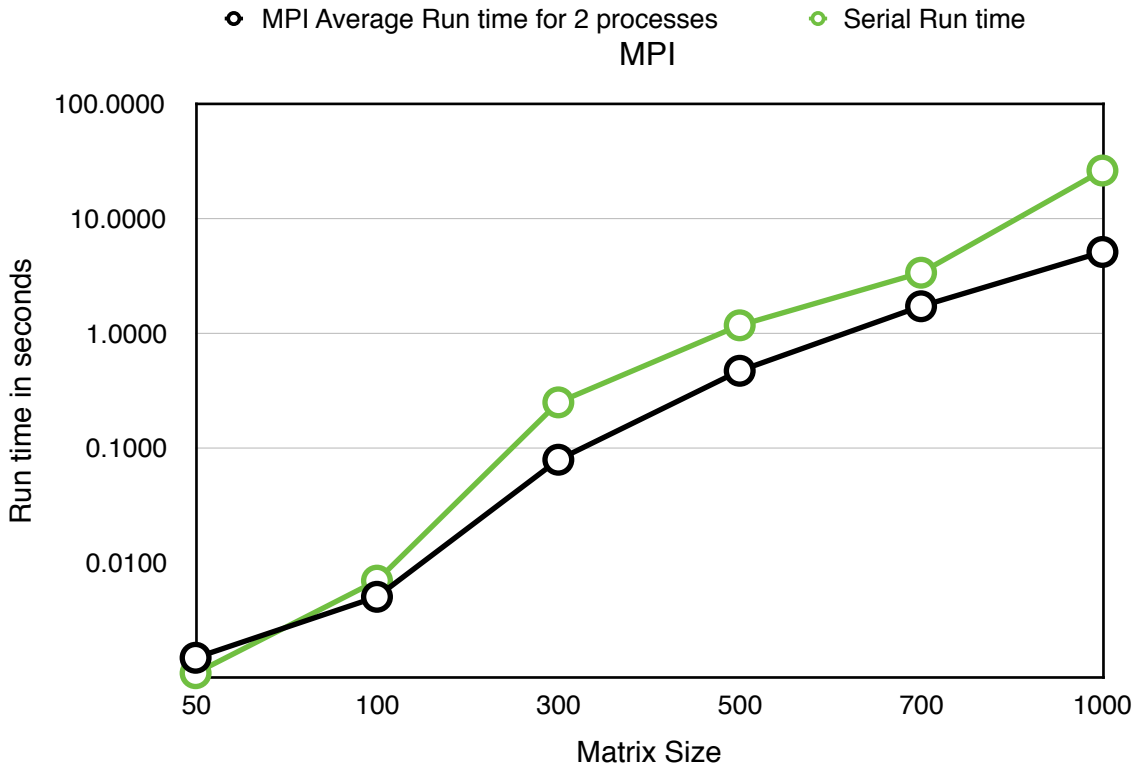
Here are some graphs showing run time averages for various matrix sizes

Run Time for MPI Code for 2 processes

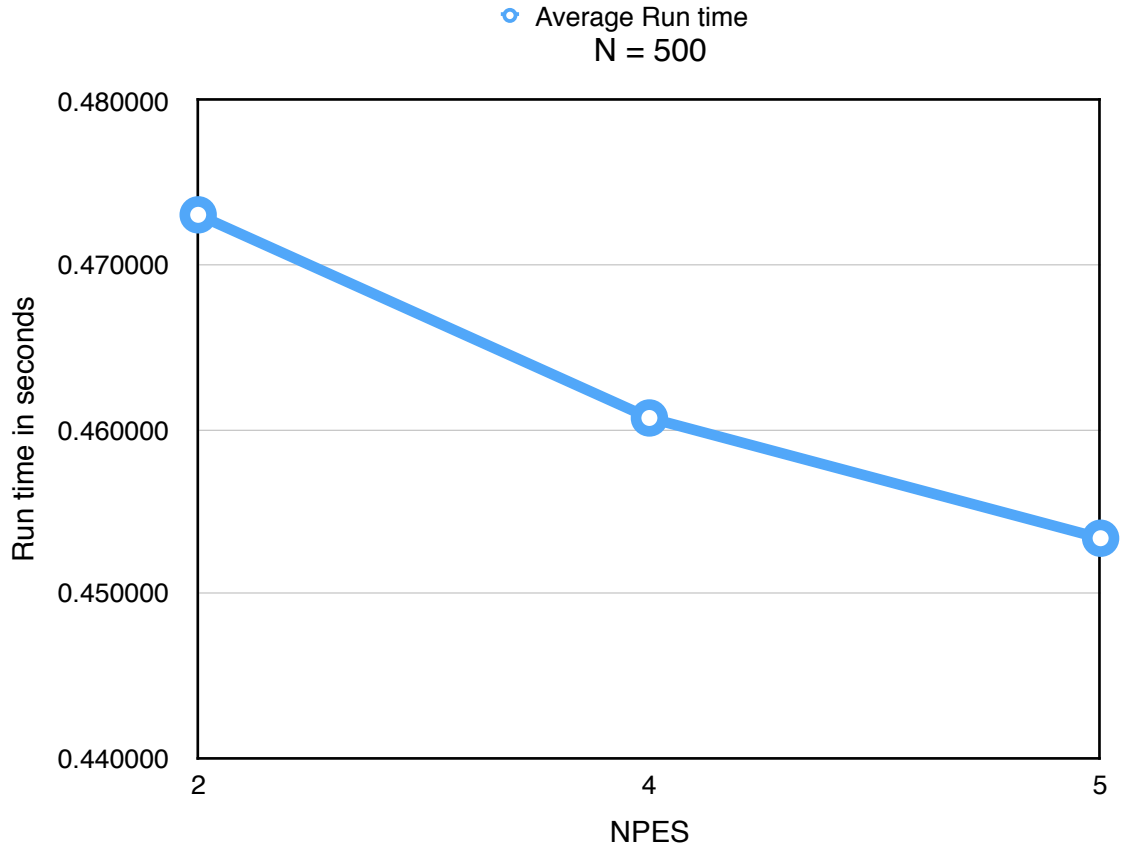
N	Run time 1	Run time 2	MPI Average Run time for 2 processes	Serial Run time	Speed Up
50	0.0014	0.0015	0.0015	0.0011	0.7339
100	0.0048	0.0053	0.0050	0.0070	1.3841
300	0.0810	0.0775	0.0793	0.2508	3.1642
500	0.4414	0.5047	0.4730	1.1788	2.4920
700	1.6857	1.7712	1.7285	3.3910	1.9619
1000	5.1715	5.1148	5.1431	26.3934	5.1318

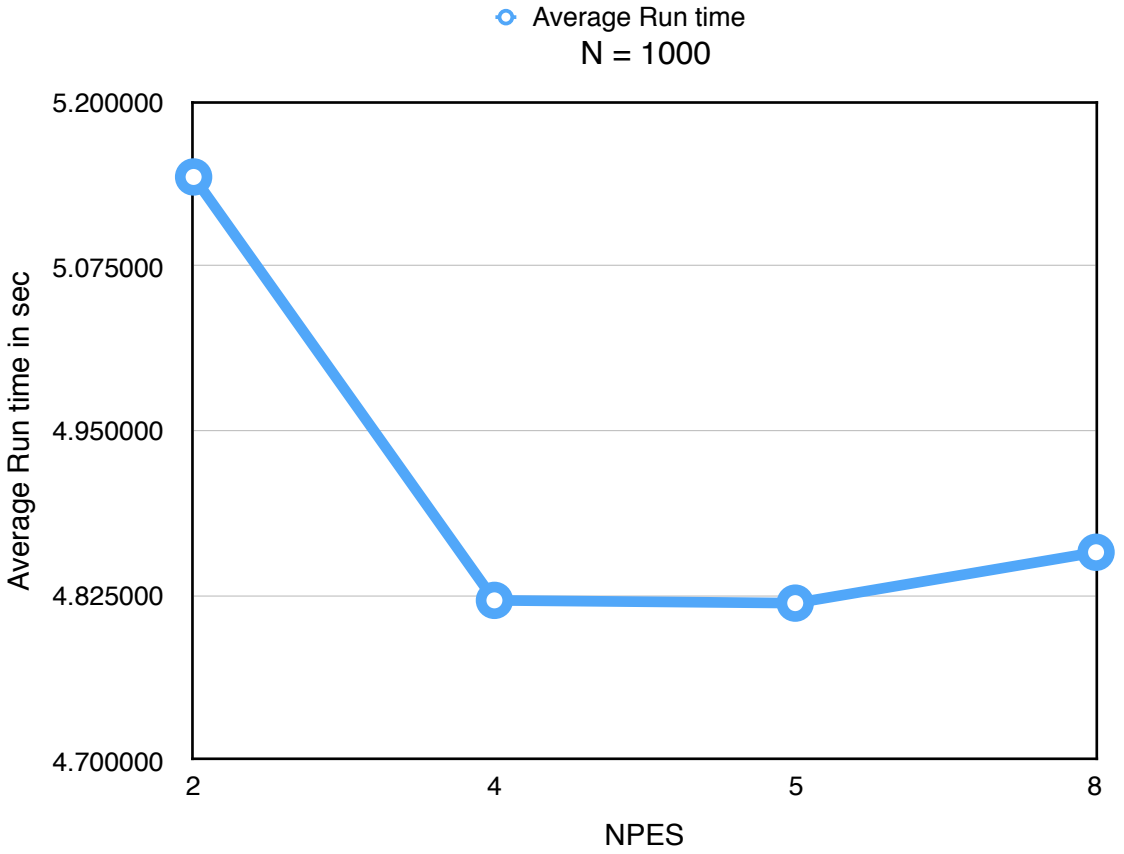


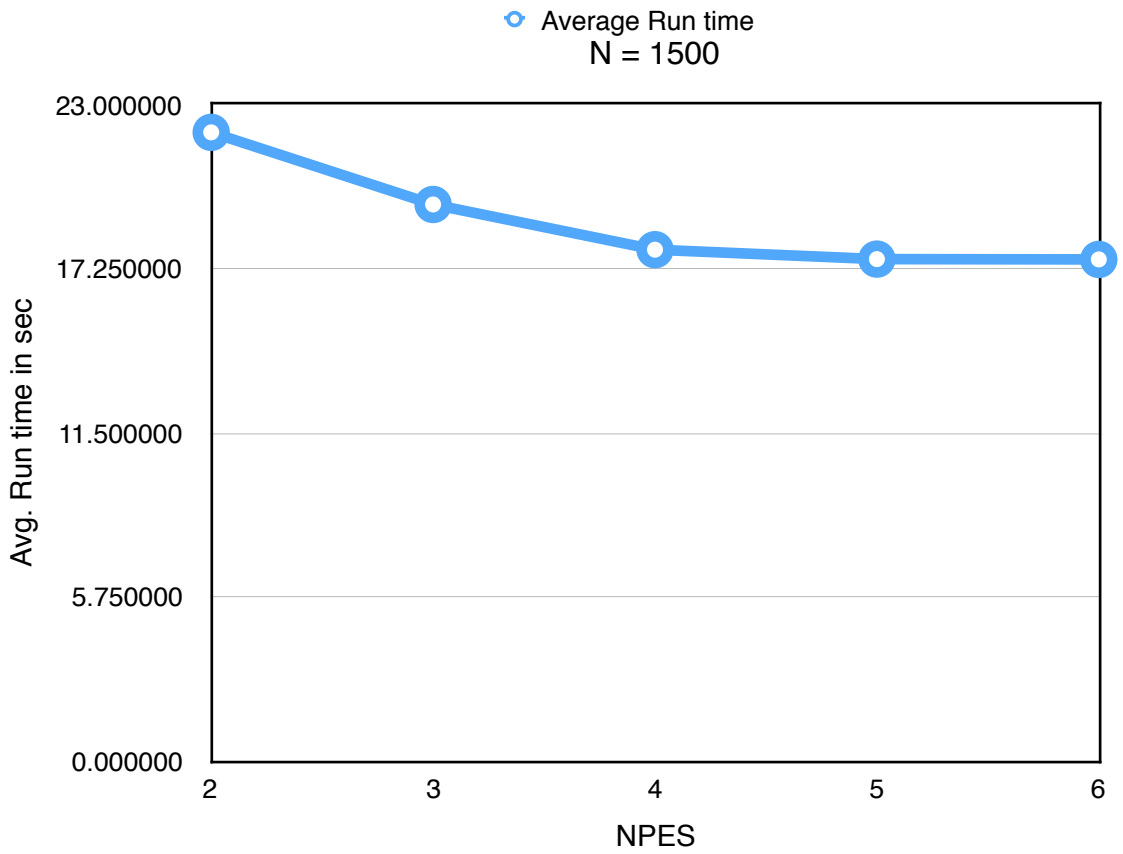
The graph above shows the cubic fit for MPI code run time and also the serial run time. The graph below shows it on the logarithmic scale for better understanding.

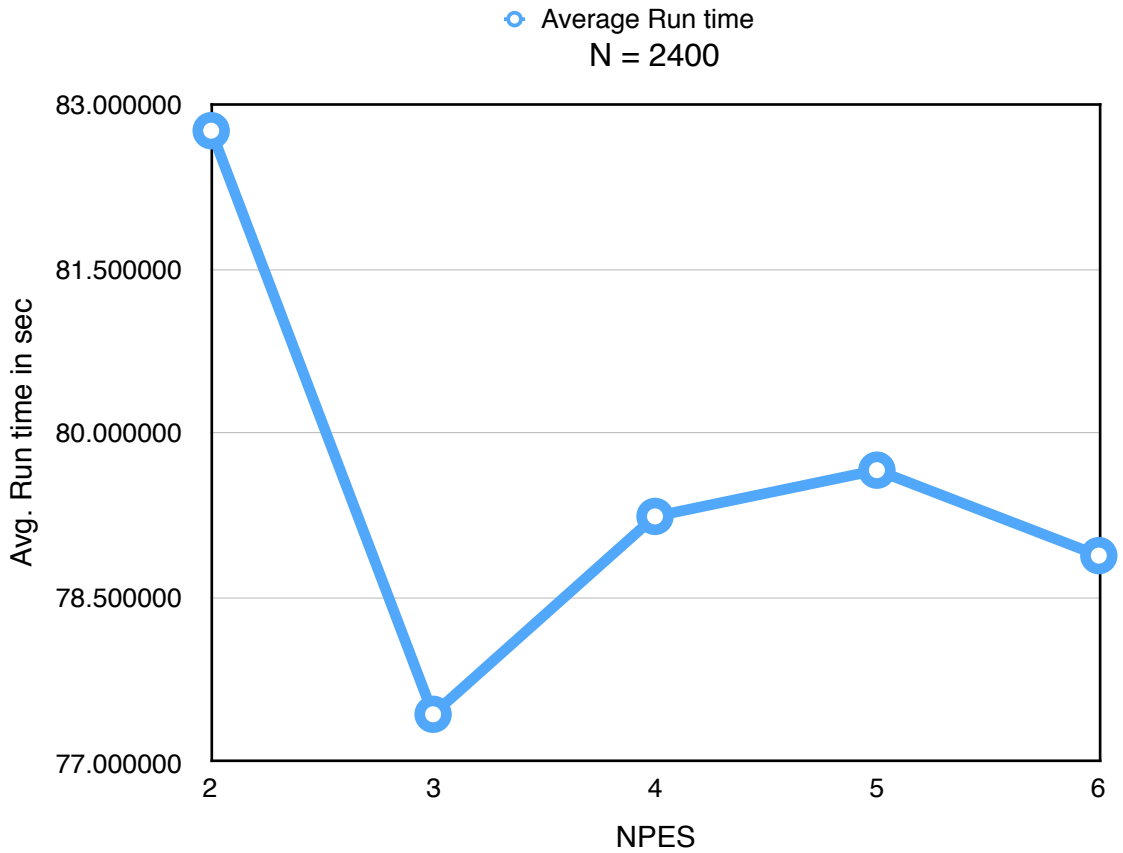


Below are some graphs which show the average run time variation for fixed matrix size and varying number of processes.


Average Run time
N = 500
0.480000
0.470000
0.460000
0.450000
0.440000
Run time in seconds
2
4
5
NPES

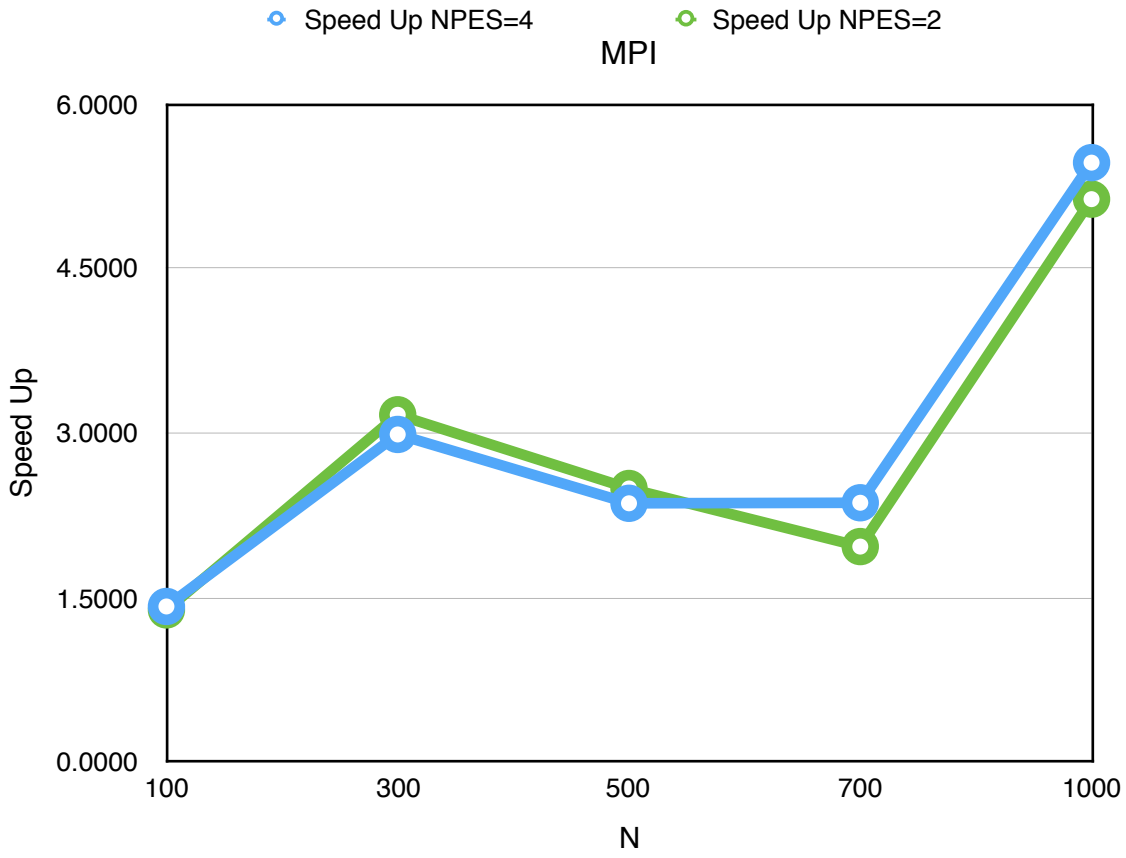

Average Run time
N = 1000
5.200000
5.075000
4.950000
4.825000
4.700000
Average Run time in sec
2
4
5
8
NPES


Average Run time
N = 1500
23.000000
17.250000
11.500000
5.750000
0.000000
Avg. Run time in sec
2
3
4
5
6
NPES


Average Run time
N = 2400
83.000000
81.500000
80.000000
78.500000
77.000000
Avg. Run time in sec
2
3
4
5
6
NPES

Explanation of the Results

- The results for the MPI code for fix matrix size and varying number of processors seems to be random for which I don't have any plausible explanation.
- For fix number of processes and varying matrix size, the speed up is increasing and then decreasing. But overall is increasing.



- As in the case of OpenMP, the speed-up in MPI crosses 1 from below possibly because of the overheads in communication with other processes.

Machine and OS Specifications

- Model Name: MacBook Pro
- Model Identifier: MacBookPro9,2
- Processor Name: Intel Core i5
- Processor Speed: 2.5 GHz
- Number of Processors: 1
- Total Number of Cores: 2
- L2 Cache (per Core): 256 KB
- L3 Cache: 3 MB
- Memory: 4 GB
- Boot ROM Version: MBP91.00D3.B0C
- SMC Version (system): 2.2f44
- Serial Number (system): C1MNCKVVDTY3
- Hardware UUID: 99B0E574-294B-5234-8BB0-A01DE50E7971
- Sudden Motion Sensor:
  - State: Enabled

Learnings

- OpenMP performs very well if shared memory is used. It is very convenient for converting from serial to parallel without much additions in the code. The reverse process is even more easier, we just have to remove the *-fopenmp* flag while compiling
- MPI is used for distributed memory systems. The conversion from serial to parallel is not convenient and requires significant effort.