ME 766: High Performance Scientific Computing

Assignment 02

Aaron John Sabu

1 Introduction

This assignment implements matrix operations for an $N \times N$ matrix - random initialization, matrix multiplication, and conversion to the upper-triangular form - using shared (OpenMP) and distributed (MPI) parallel computing. The following section provides insight into the work done for the same.

2 Timing Study Results

2.1 OpenMP

We run the OpenMP code over specific values of N and over a specific set of number of threads.

	1	2	3	4	5	6	8	10	12	15	16
120	0.000	0.000	0.250	0.000	0.000	0.250	0.250	0.250	0.500	0.000	0.000
240	0.000	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.250
480	1.500	1.000	1.250	1.000	1.000	1.250	1.000	1.500	4.750	1.250	3.000
720	5.000	4.000	3.250	3.000	2.750	2.750	2.750	2.500	2.750	3.250	3.250
960	11.750	8.500	6.500	5.750	5.500	5.000	5.000	4.750	4.750	5.500	10.000
1200	22.000	15.250	11.750	10.250	9.750	9.000	8.000	8.000	9.250	11.500	37.750
1500	47.000	30.250	23.000	19.250	17.000	15.750	14.500	13.750	13.500	12.500	14.750
1800	73.750	48.000	36.250	30.750	27.000	24.750	22.500	21.750	21.750	21.250	22.500
2100	126.250	70.000	58.000	47.000	43.000	38.500	34.500	31.500	31.000	27.500	27.500
2400	179.500	96.250	66.500	52.250	44.000	49.250	45.500	42.250	41.000	39.000	48.000
3600	645.750	367.250	264.250	213.500	182.000	164.000	139.500	121.250	114.000	106.750	103.000
4800	2023.500	1094.000	761.000	594.750	494.500	433.250	354.250	305.500	273.250	254.750	242.500
9600	18091.500	9190.250	6391.750	4793.000	3913.500	2885.000	2209.250	1826.750	2897.250	1622.250	1589.000

2.2 MPI

We run the MPI code over specific values of N and over a specific set of number of threads.

	2	3	4	5	6	7	9	11	13	16	17
120	0.032	0.018	0.015	0.014	0.013	0.014	0.012	0.014	0.014	0.015	0.014
240	0.177	0.100	0.085	0.081	0.075	0.075	0.070	0.059	0.066	0.064	0.057
480	1.229	0.641	0.463	0.412	0.382	0.363	0.340	0.329	0.327	0.325	0.315
720	4.071	2.047	1.447	1.267	1.155	1.081	0.989	0.948	0.917	0.890	0.885
960	9.467	4.697	3.336	2.918	2.648	2.470	2.259	2.149	2.083	2.008	1.988
1200	17.809	9.068	6.331	5.499	5.087	4.769	4.422	4.200	4.012	3.720	3.791
1500	36.980	18.563	13.018	11.557	11.216	10.689	9.746	9.197	8.693	8.130	7.942
1800	61.365	30.996	22.166	20.807	18.847	17.788	15.955	14.889	14.138	13.284	13.058
2100	101.668	52.088	42.122	37.445	34.585	31.763	28.046	25.815	24.295	22.445	22.021
2400	144.156	75.325	53.625	46.842	42.407	39.491	35.746	33.656	32.193	30.613	30.136
3600	514.783	264.492	185.256	159.513	144.481	134.400	121.677	114.444	109.483	104.185	102.886
4800	1648.998	830.010	574.693	480.518	428.212	387.045	341.527	313.608	294.914	275.003	270.529
9600	14299.618	7489.097	5242.604	4393.968	3836.987	3514.742	3058.494	2671.629	2493.785	2309.545	2269.529

3 Plots

3.1 OpenMP

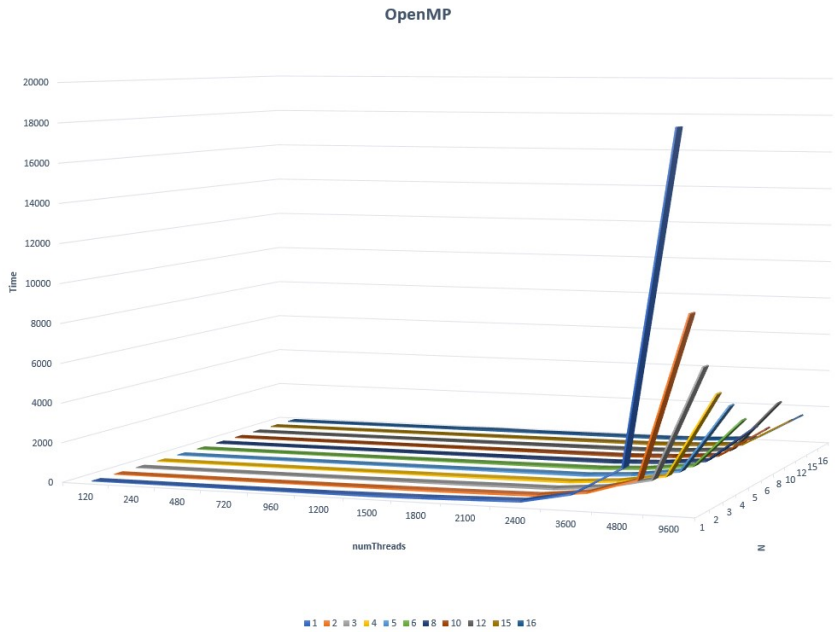

Figure 1: OpenMP

3.2 Timing Study

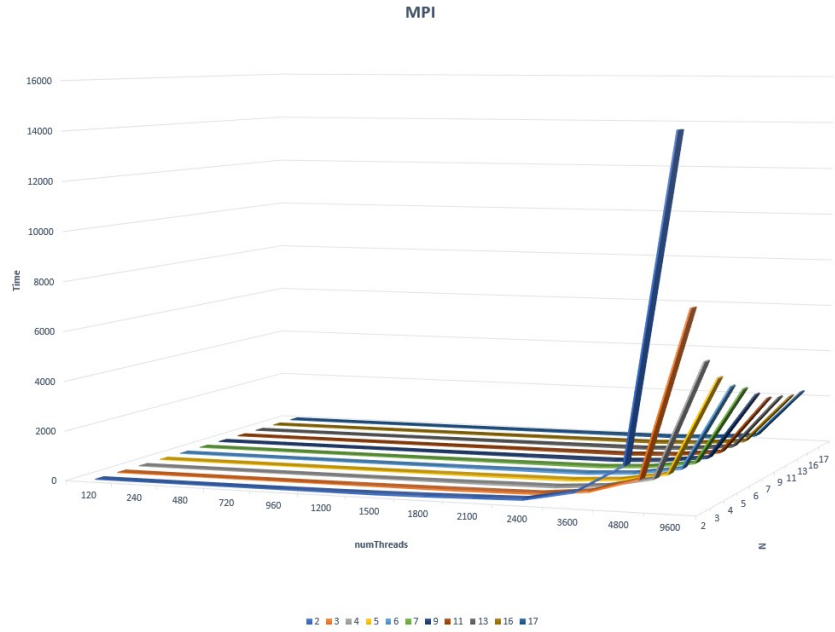

Figure 2: MPI

3.3 Comparison of $N = 9600$

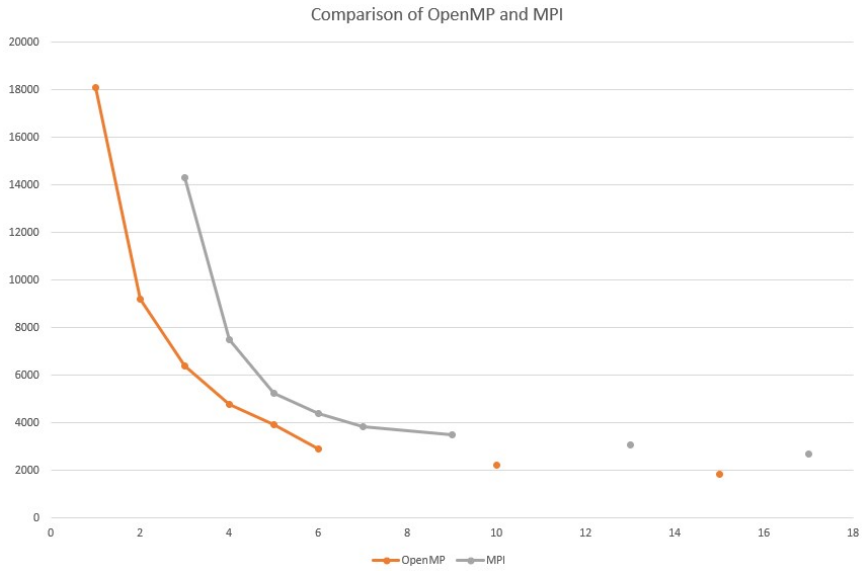

Figure 3: Comparison of $N = 9600$

4 Observations

We observe that, at a given number of threads, the time taken for MPI is more than that taken for OpenMP due to the tranfer of information from the master to the slaves and vice versa. However, it is observed that MPI at a given number of threads performs better than OpenMP with one fewer than the given number of threads. This comparison arises since computation for MPI is only performed on one fewer than the given number of threads and this is because of the superiority of inter-process capabilities offered by MPI in comparison to OpenMP.

5 Configurational Detail

The final codes were run on the supercomputer (PARAM Sanganak) for which the author is very highly grateful to the instructor and C-DAC. Testing was performed on the laptop of the author which has the following characteristics:

- **CPU**
  - Specification: Intel Core i5-10300H CPU @ 2.50GHz
  - Cores: 4
  - Threads: 8
- **RAM**
  - Size: 8192 MBytes
- **Operating System**
  - Main OS: Windows 10 Home Single Language 64-bit
  - Windows Subsystem for Linux: Debian