HW2 Performance Analysis

Pthread-

Correctness argument: The loop in the sequential program, which used to iterate over rows for normalization is divided in the parallel program and a cluster of rows were allocated to each thread. There was no data dependency between two iterations of this loop and hence no need of any synchronisation. The reason behind avoiding the use of synchronisation was that it was reducing the performance of the program.

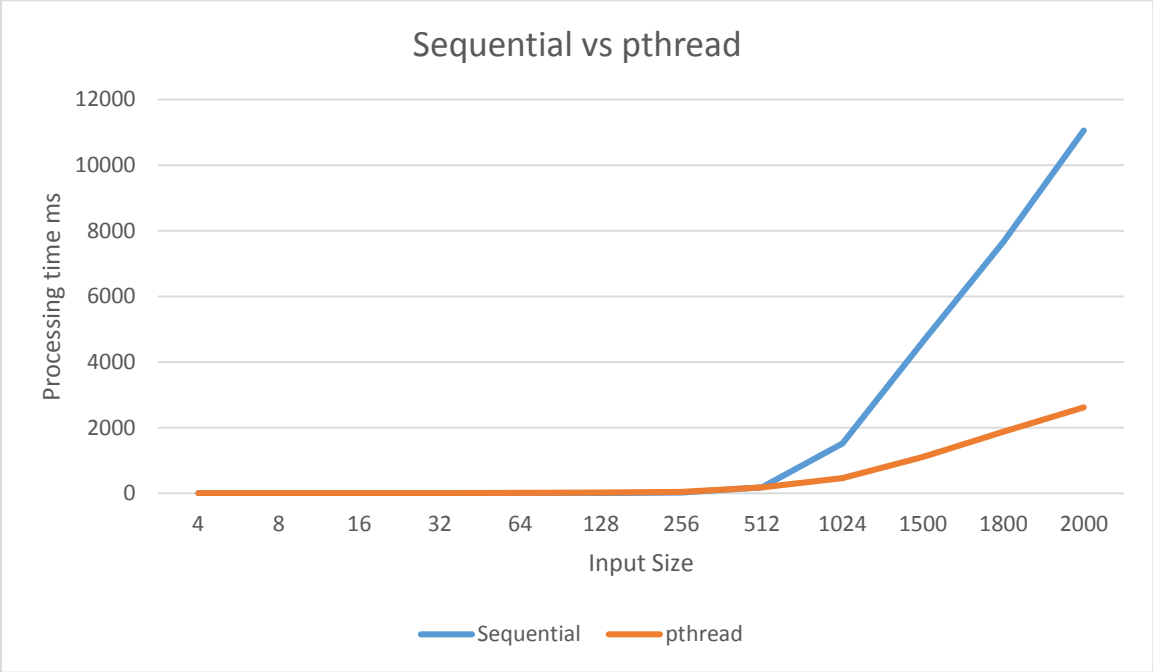
Different Versions tried:

1. The first version of the program used to create a new thread for each iteration of the innermost loop, which iterates over the columns. The problem with this approach was there were too many thread creations and joins, and the code ended up performing even worse than the sequential code due to this overhead. Hence that code was discarded, as the parallelization was on the incorrect loop degrading performance.

2. The second version contained a shared "data" array (used to pass argument to the function executed in a thread) between each thread, which required synchronisation, and was reducing the performance. Separate data arrays were created for each thread, which eliminated this performance.

3. The third and final version is reasonably efficient as even though the threads are created and destroyed n times, it is much cheaper than the first version (threads were being created n^2 times). Also the use of synchronisation was making the other three threads wait for long, in the second version, and the speedup was not that good. The current version provides a **Super Linear Speedup**, and hence it is quite efficient.

Performance analysis:



Following table shows the comparison between the parallel code using pthread and the sequential code, for different problem size. For bigger problem sizes, the parallel program works much faster than the sequential code. Please note that 4 threads (cores) were used for this analysis in parallel program.

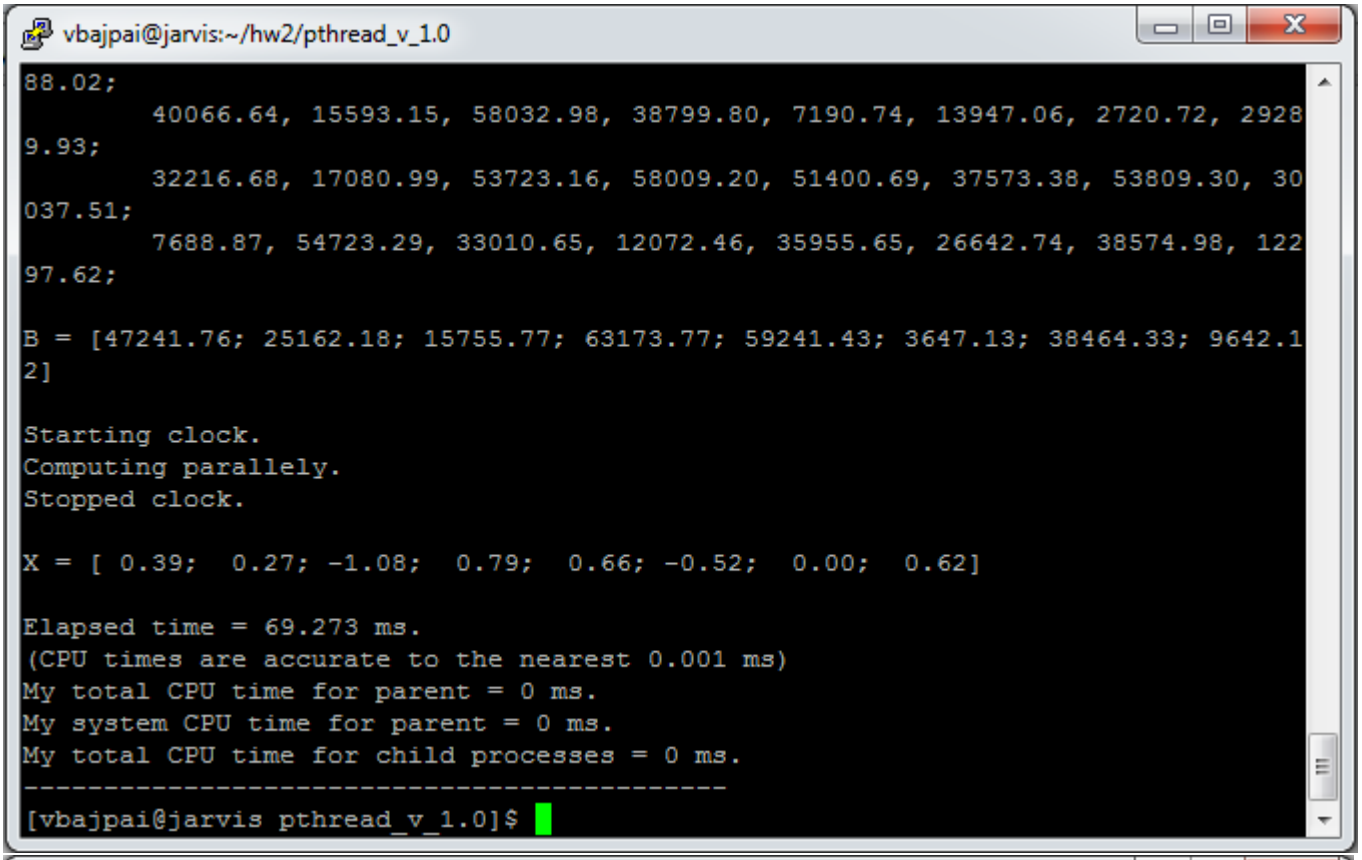
Matrix Dimension (N)	Sequential Processing time	Pthread Processing time
4	0.012	0.671
8	0.013	1.177
16	0.032	2.245
32	0.161	4.533
64	1.098	8.922
128	4.205	18.358
256	25.585	43.951
512	174.357	178.306
1024	1515.24	456.971
1500	4612.51	1105.59
1800	7657.37	1879.14
2000	11051	2614.54

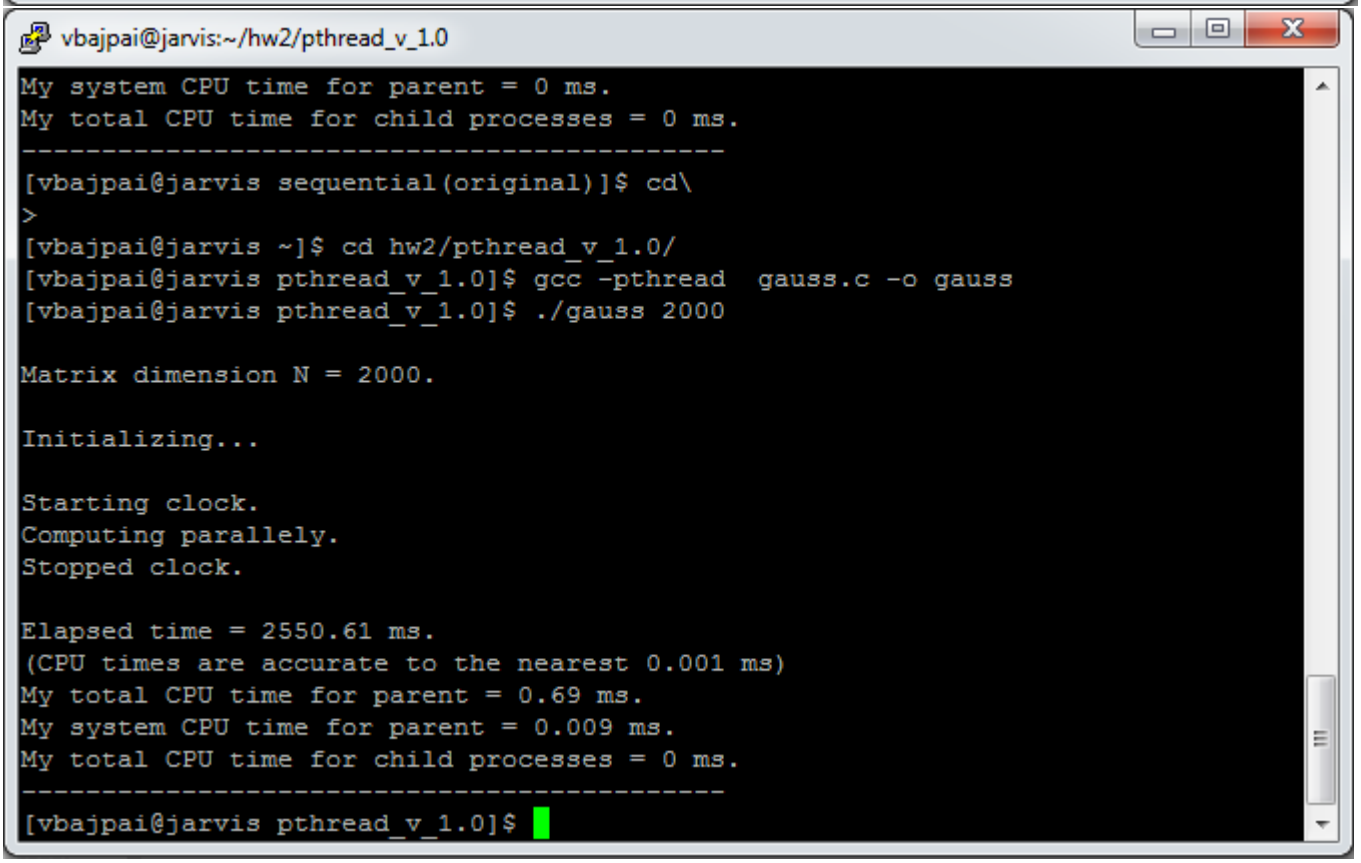
vbajpai@jarvis:~/hw2/sequential(original)

```
436.16;
        53178.46, 53123.15, 15524.02, 54038.47, 50407.95, 54516.72, 42997.06, 66
46.10;
        38263.33, 36107.78, 26287.80, 56861.48, 28811.47, 44355.20, 40803.26, 42
516.12;
        43014.98, 18674.44, 37085.13, 49518.34, 5964.28, 24111.51, 15606.73, 533
31.20;

B = [22927.08; 63649.12; 1439.24; 17865.71; 45251.82; 46543.45; 418.09; 57003.77
]

Starting clock.
Computing Serially.
Stopped clock.

X = [ 3.20; -2.84;  2.90; -1.70; -0.74;  2.06;  0.27; -1.88]

Elapsed time = 0.012 ms.
(CPU times are accurate to the nearest 0.001 ms)
My total CPU time for parent = 0 ms.
My system CPU time for parent = 0 ms.
My total CPU time for child processes = 0 ms.
----------------------------------------
[vbajpai@jarvis sequential(original)]$
```

vbajpai@jarvis:~/hw2/pthread_v_1.0

```
>
[vbajpai@jarvis ~]$ cd hw2/sequential\(original\)/
[vbajpai@jarvis sequential(original)]$
[vbajpai@jarvis sequential(original)]$
[vbajpai@jarvis sequential(original)]$ gcc gauss.c -o gauss
[vbajpai@jarvis sequential(original)]$ ./gauss 2000

Matrix dimension N = 2000.

Initializing...

Starting clock.
Computing Serially.
Stopped clock.

Elapsed time = 11634.4 ms.
(CPU times are accurate to the nearest 0.001 ms)
My total CPU time for parent = 1.163 ms.
My system CPU time for parent = 0 ms.
My total CPU time for child processes = 0 ms.
----------------------------------------
[vbajpai@jarvis sequential(original)]$ cd\
>
[vbajpai@jarvis ~]$ cd hw2/pthread_v_1.0/
```


vbajpai@jarvis:~/hw2/pthread_v_1.0
88.02;
40066.64, 15593.15, 58032.98, 38799.80, 7190.74, 13947.06, 2720.72, 2928
9.93;
32216.68, 17080.99, 53723.16, 58009.20, 51400.69, 37573.38, 53809.30, 30
037.51;
7688.87, 54723.29, 33010.65, 12072.46, 35955.65, 26642.74, 38574.98, 122
97.62;
B = [47241.76; 25162.18; 15755.77; 63173.77; 59241.43; 3647.13; 38464.33; 9642.1
2]
Starting clock.
Computing parallely.
Stopped clock.
X = [0.39; 0.27; -1.08; 0.79; 0.66; -0.52; 0.00; 0.62]
Elapsed time = 69.273 ms.
(CPU times are accurate to the nearest 0.001 ms)
My total CPU time for parent = 0 ms.
My system CPU time for parent = 0 ms.
My total CPU time for child processes = 0 ms.
[vbajpai@jarvis pthread_v_1.0]$


vbajpai@jarvis:~/hw2/pthread_v_1.0
My system CPU time for parent = 0 ms.
My total CPU time for child processes = 0 ms.
[vbajpai@jarvis sequential(original)]$ cd\
>
[vbajpai@jarvis ~]$ cd hw2/pthread_v_1.0/
[vbajpai@jarvis pthread_v_1.0]$ gcc -pthread gauss.c -o gauss
[vbajpai@jarvis pthread_v_1.0]$ ./gauss 2000
Matrix dimension N = 2000.
Initializing...
Starting clock.
Computing parallely.
Stopped clock.
Elapsed time = 2550.61 ms.
(CPU times are accurate to the nearest 0.001 ms)
My total CPU time for parent = 0.69 ms.
My system CPU time for parent = 0.009 ms.
My total CPU time for child processes = 0 ms.
[vbajpai@jarvis pthread_v_1.0]$

Possible Improvements:

1. Repetitive creation of threads: The threads are being created and destroyed n times. This is reducing the performance a lot. It can be eliminated if we can parallelize the outer loop somehow. Currently, there is a dependency of every iteration of the loop on the previous iteration.

2. Cache coherence for data in different cores: Even though the cores on working on separate sections of the arrays, it is possible that some of these may have multiple copies in caches of different cores, and if one core modifies a part of array, the system may consider that as a cache coherence issue, and may try to update the data in memory again and again. This could be a potential reason for decrease in performance. We may add some pad to this data so that the cache contains only data which is needed for that core, along with some padded data. For this modification, we need to know the exact size of the cache.

Speed up:

$$Sp = Ts/Tp$$

For input size N=2000,

Ts = 11051 ms

Tp = 2614.54 ms

Sp =11051/2614.54

Sp = 4.227

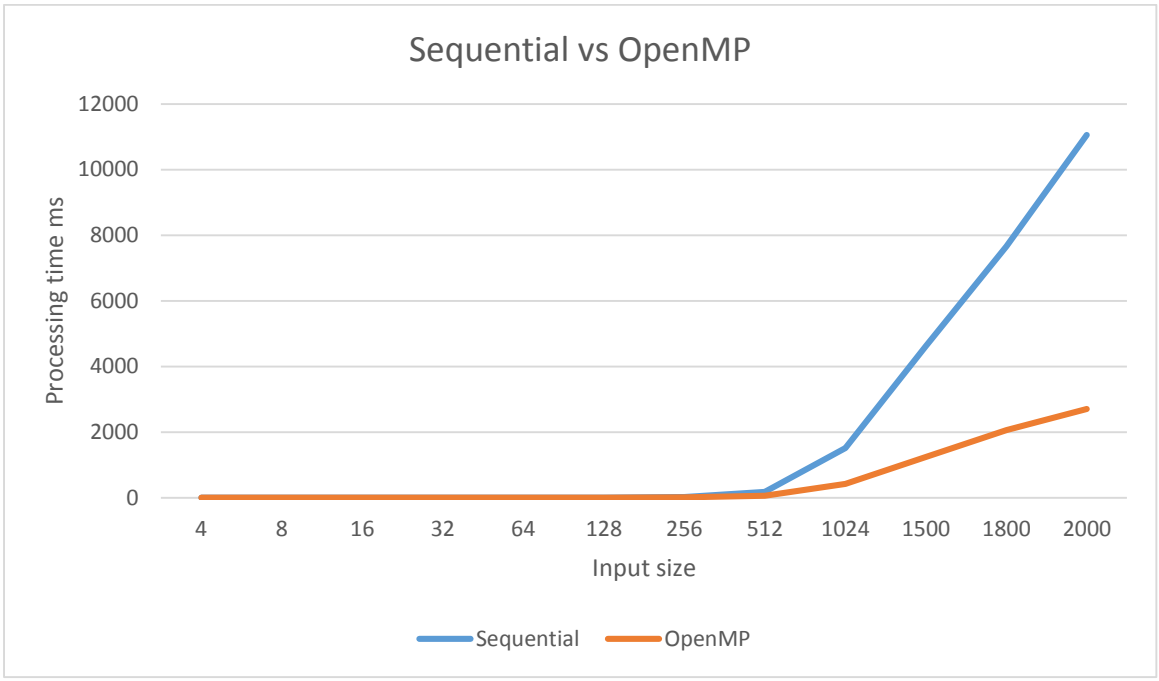
So it's a super linear speedup

OpenMP-

Correctness Argument:

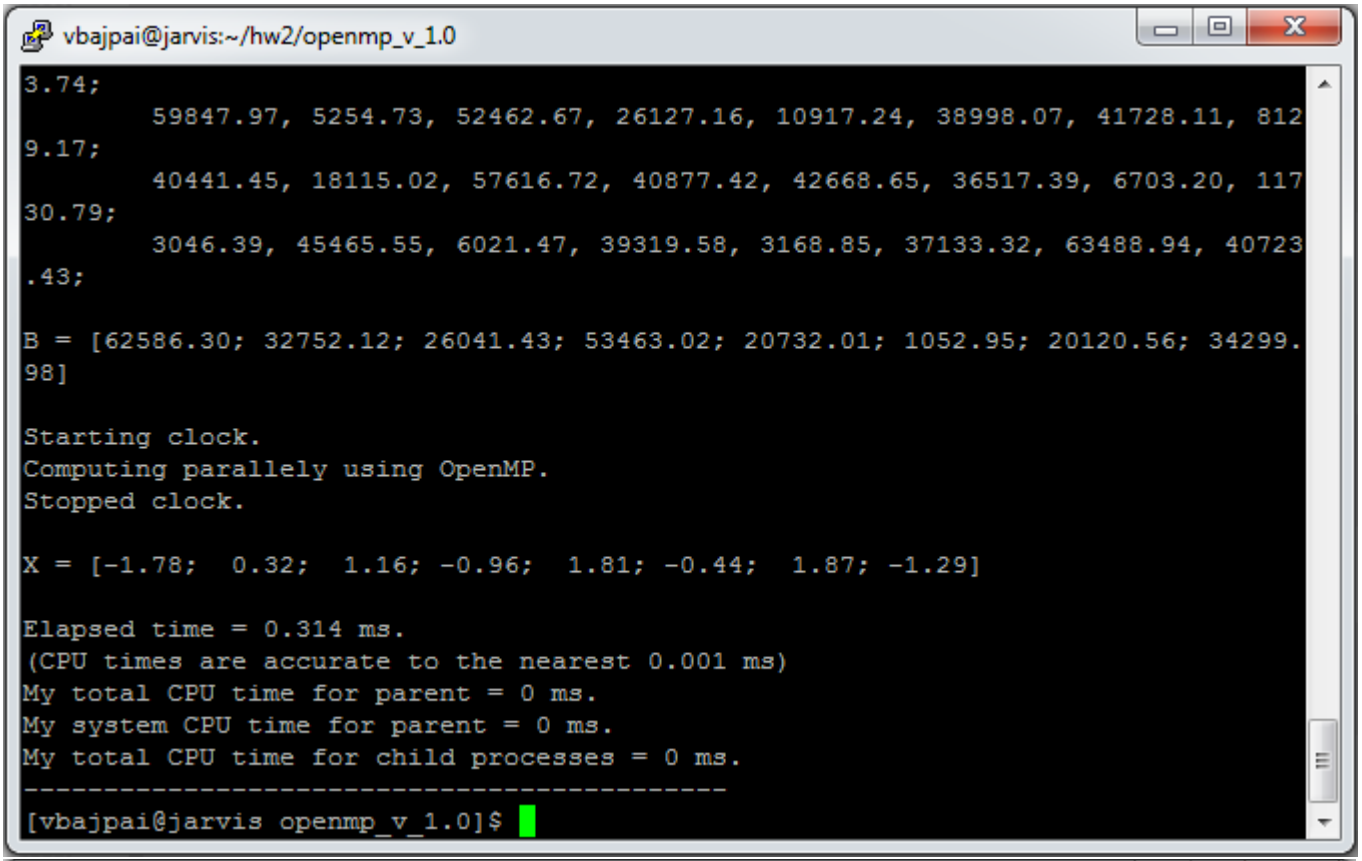
This code has the exact same logic as mentioned above for pthread. The data which is shared between threads has been declared as “shared” and the data which is thread specific is kept as “private”. The total number of threads requested are 4.

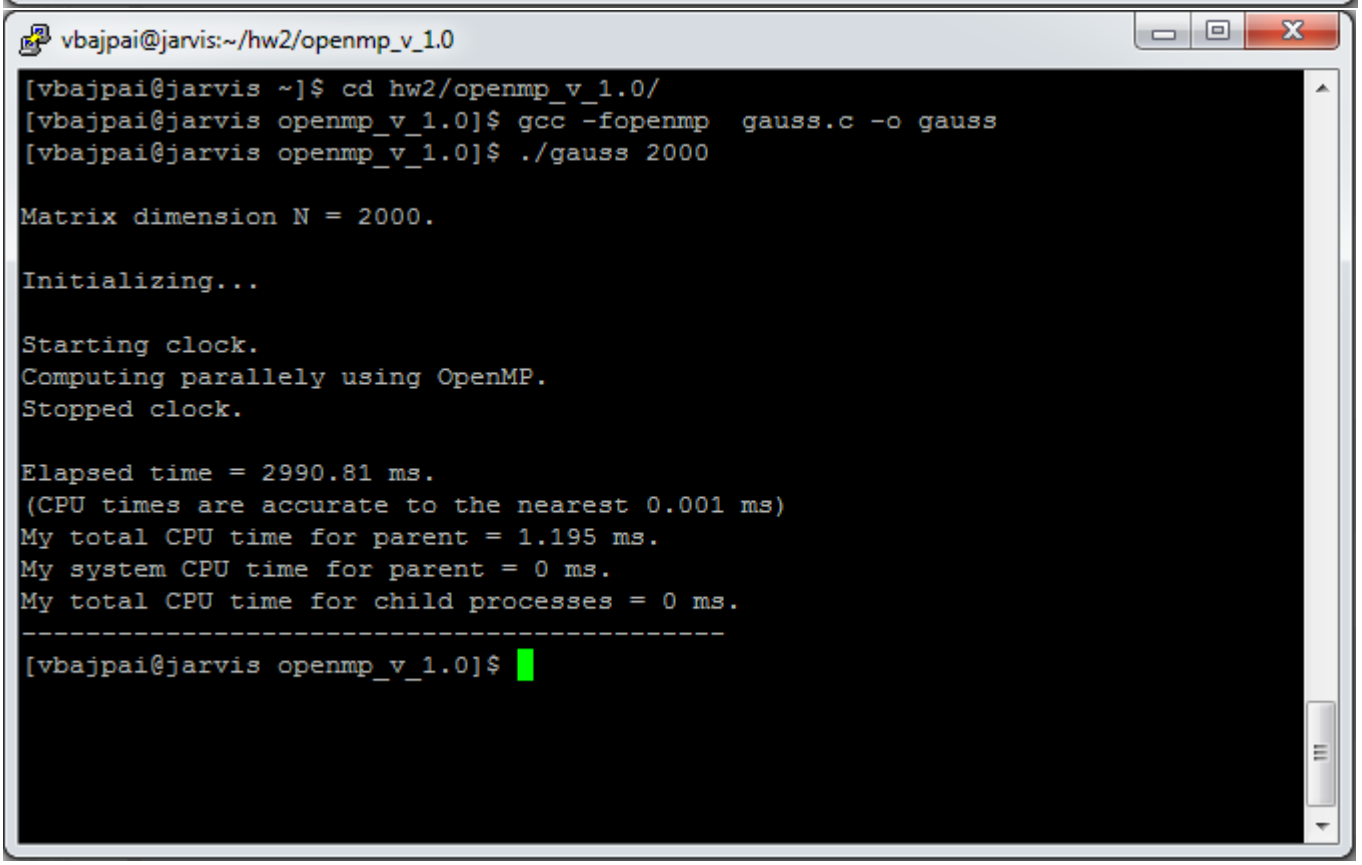
Performance Analysis:



Following table shows the comparison between the parallel code using OpenMP and the sequential code, for different problem size. For bigger problem sizes, the parallel program works much faster than the sequential code. Please note that 4 threads (cores) were used for this analysis in parallel program.

Matrix Dimension (N)	Sequential Processing time	OpenMP Processing time
4	0.012	0.28
8	0.013	0.354
16	0.032	0.37
32	0.161	0.576
64	1.098	1.025
128	4.205	3.294
256	25.585	15.267
512	174.357	61.346
1024	1515.24	421.884
1500	4612.51	1239.89
1800	7657.37	2060.68
2000	11051	2703.49


vbajpai@jarvis:~/hw2/openmp_v_1.0
3.74;
59847.97, 5254.73, 52462.67, 26127.16, 10917.24, 38998.07, 41728.11, 812
9.17;
40441.45, 18115.02, 57616.72, 40877.42, 42668.65, 36517.39, 6703.20, 117
30.79;
3046.39, 45465.55, 6021.47, 39319.58, 3168.85, 37133.32, 63488.94, 40723
.43;
B = [62586.30; 32752.12; 26041.43; 53463.02; 20732.01; 1052.95; 20120.56; 34299.
98]
Starting clock.
Computing parallely using OpenMP.
Stopped clock.
X = [-1.78; 0.32; 1.16; -0.96; 1.81; -0.44; 1.87; -1.29]
Elapsed time = 0.314 ms.
(CPU times are accurate to the nearest 0.001 ms)
My total CPU time for parent = 0 ms.
My system CPU time for parent = 0 ms.
My total CPU time for child processes = 0 ms.
[vbajpai@jarvis openmp_v_1.0]$


vbajpai@jarvis:~/hw2/openmp_v_1.0
[vbajpai@jarvis ~]$ cd hw2/openmp_v_1.0/
[vbajpai@jarvis openmp_v_1.0]$ gcc -fopenmp gauss.c -o gauss
[vbajpai@jarvis openmp_v_1.0]$ ./gauss 2000
Matrix dimension N = 2000.
Initializing...
Starting clock.
Computing parallely using OpenMP.
Stopped clock.
Elapsed time = 2990.81 ms.
(CPU times are accurate to the nearest 0.001 ms)
My total CPU time for parent = 1.195 ms.
My system CPU time for parent = 0 ms.
My total CPU time for child processes = 0 ms.
[vbajpai@jarvis openmp_v_1.0]$

Speed up:

$$Sp = Ts/Tp$$

For input size N=2000,

Ts = 11051 ms

Tp = 2703.49 ms

Sp =11051/2703.49

Sp = 4.08

So it's a super linear speedup