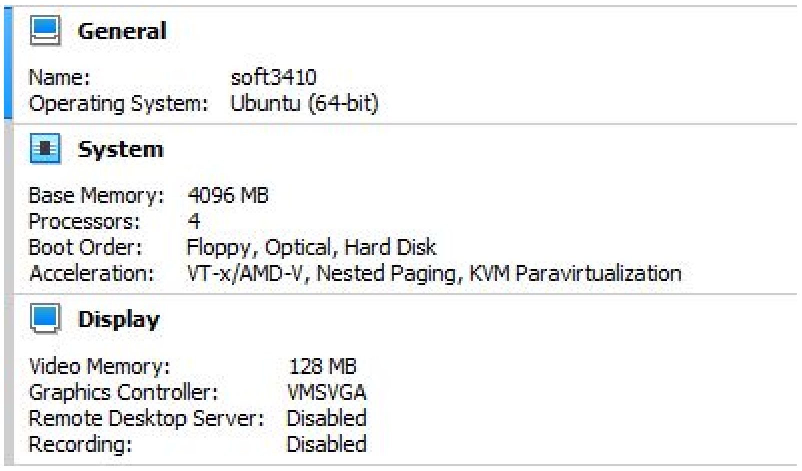
SOFT3410 Assignment 2 Report

System Specifications:

I am running this code on an Ubuntu VM (VirtualBox), the system specifications for the VM are seen below.



General

Name: soft3410
Operating System: Ubuntu (64-bit)

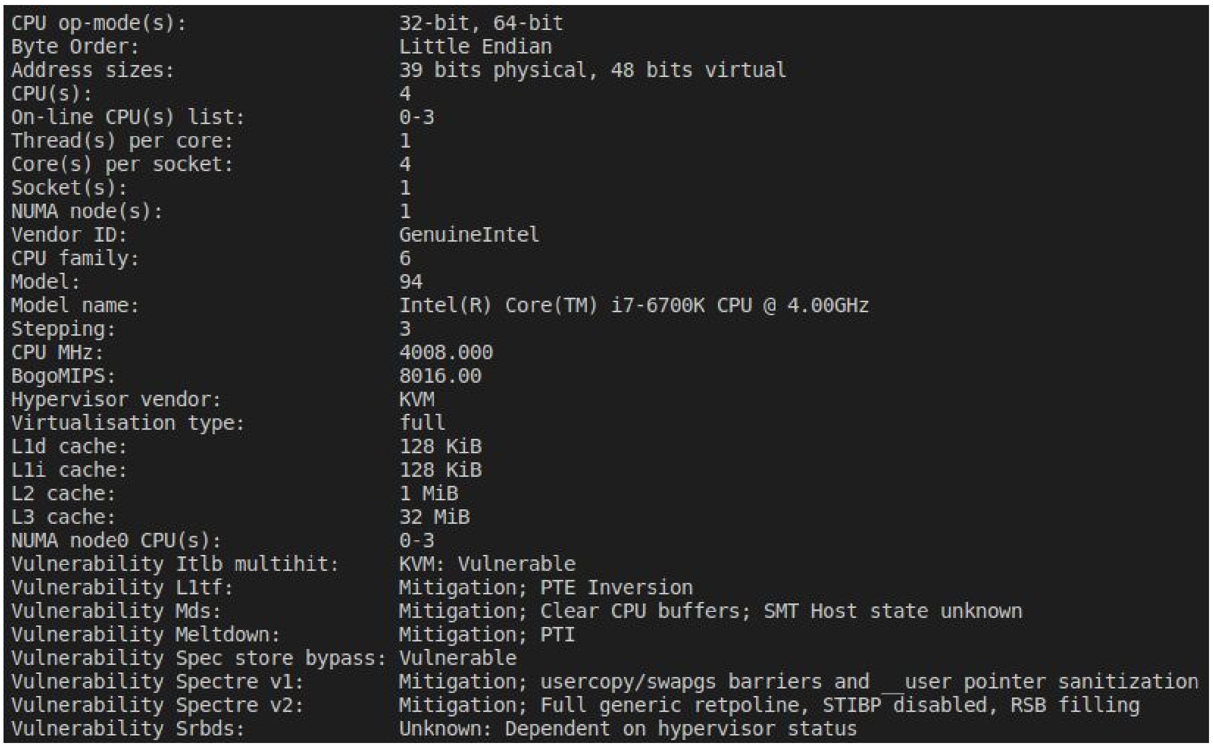
System

Base Memory: 4096 MB
Processors: 4
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display

Video Memory: 128 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

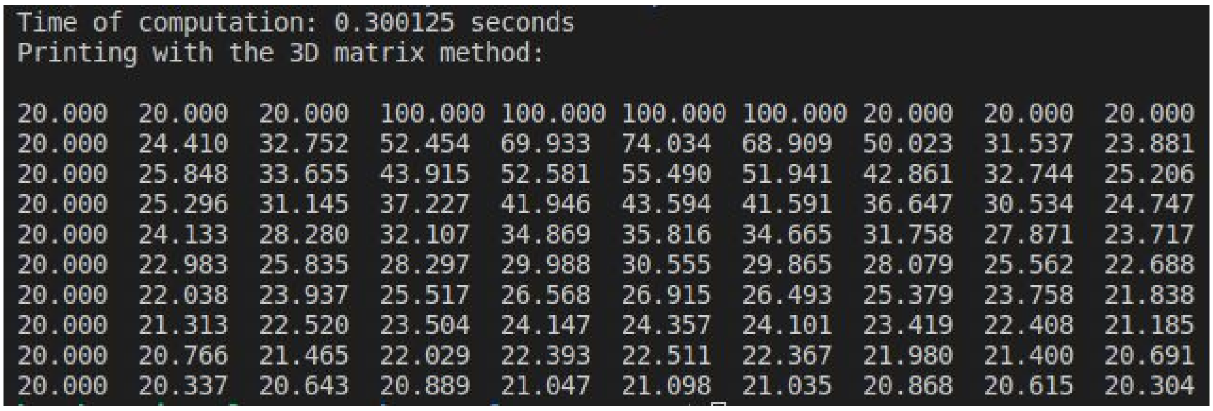
Below is the output for calling lscpu in the VM.



```
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          4
On-line CPU(s) list:             0-3
Thread(s) per core:              1
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           94
Model name:                      Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
Stepping:                        3
CPU MHz:                         4008.000
BogoMIPS:                        8016.00
Hypervisor vendor:               KVM
Virtualisation type:             full
L1d cache:                       128 KiB
L1i cache:                       128 KiB
L2 cache:                        1 MiB
L3 cache:                        32 MiB
NUMA node0 CPU(s):               0-3
Vulnerability Itlb multihit:     KVM: Vulnerable
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT Host state unknown
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, STIBP disabled, RSB filling
Vulnerability Srbds:             Unknown: Dependent on hypervisor status
```

Part 1: Heat Distribution Calculation

Q1:

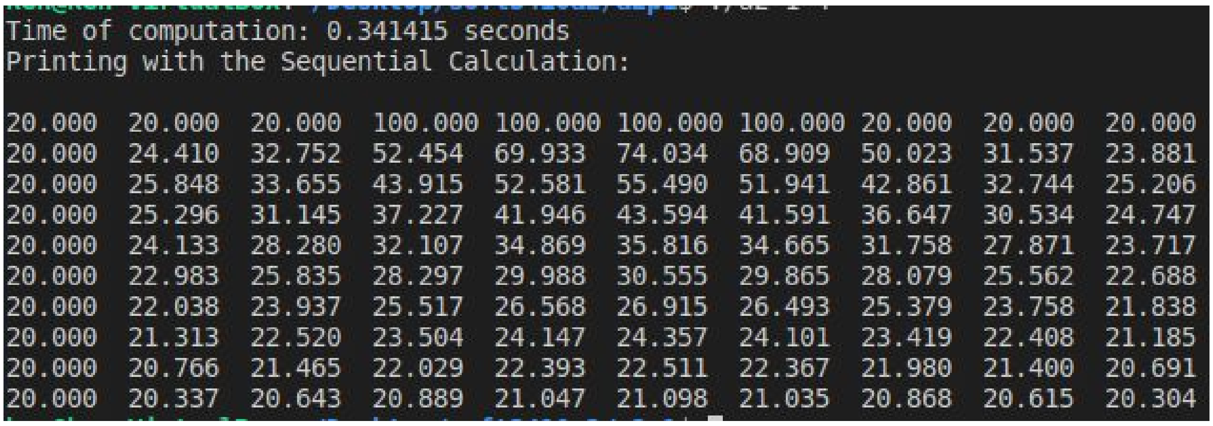


```
Time of computation: 0.300125 seconds
Printing with the 3D matrix method:

20.000  20.000  20.000  100.000 100.000 100.000 100.000 20.000  20.000  20.000
20.000  24.410  32.752  52.454  69.933  74.034  68.909  50.023  31.537  23.881
20.000  25.848  33.655  43.915  52.581  55.490  51.941  42.861  32.744  25.206
20.000  25.296  31.145  37.227  41.946  43.594  41.591  36.647  30.534  24.747
20.000  24.133  28.280  32.107  34.869  35.816  34.665  31.758  27.871  23.717
20.000  22.983  25.835  28.297  29.988  30.555  29.865  28.079  25.562  22.688
20.000  22.038  23.937  25.517  26.568  26.915  26.493  25.379  23.758  21.838
20.000  21.313  22.520  23.504  24.147  24.357  24.101  23.419  22.408  21.185
20.000  20.766  21.465  22.029  22.393  22.511  22.367  21.980  21.400  20.691
20.000  20.337  20.643  20.889  21.047  21.098  21.035  20.868  20.615  20.304
```

Q2:

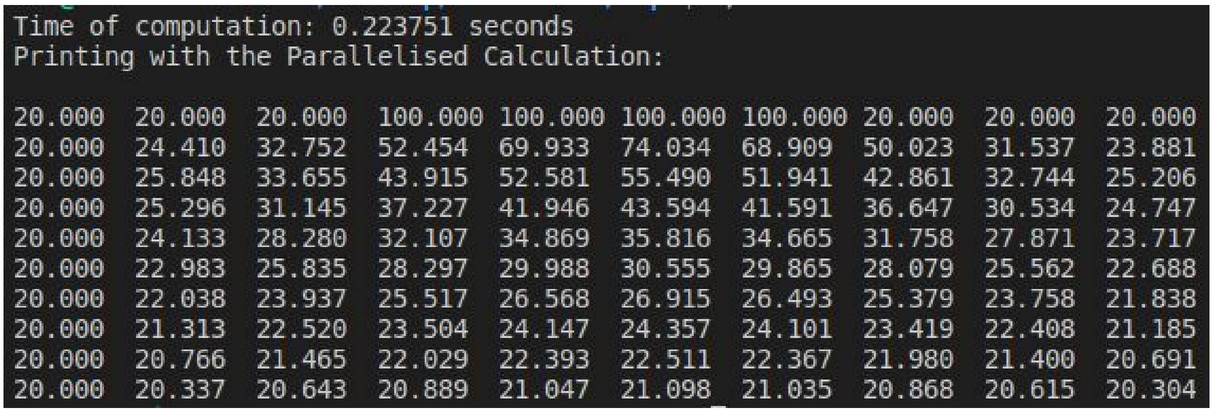
Output of the Sequential Calculation:



```
Time of computation: 0.341415 seconds
Printing with the Sequential Calculation:

20.000  20.000  20.000  100.000 100.000 100.000 100.000 20.000  20.000  20.000
20.000  24.410  32.752  52.454  69.933  74.034  68.909  50.023  31.537  23.881
20.000  25.848  33.655  43.915  52.581  55.490  51.941  42.861  32.744  25.206
20.000  25.296  31.145  37.227  41.946  43.594  41.591  36.647  30.534  24.747
20.000  24.133  28.280  32.107  34.869  35.816  34.665  31.758  27.871  23.717
20.000  22.983  25.835  28.297  29.988  30.555  29.865  28.079  25.562  22.688
20.000  22.038  23.937  25.517  26.568  26.915  26.493  25.379  23.758  21.838
20.000  21.313  22.520  23.504  24.147  24.357  24.101  23.419  22.408  21.185
20.000  20.766  21.465  22.029  22.393  22.511  22.367  21.980  21.400  20.691
20.000  20.337  20.643  20.889  21.047  21.098  21.035  20.868  20.615  20.304
```

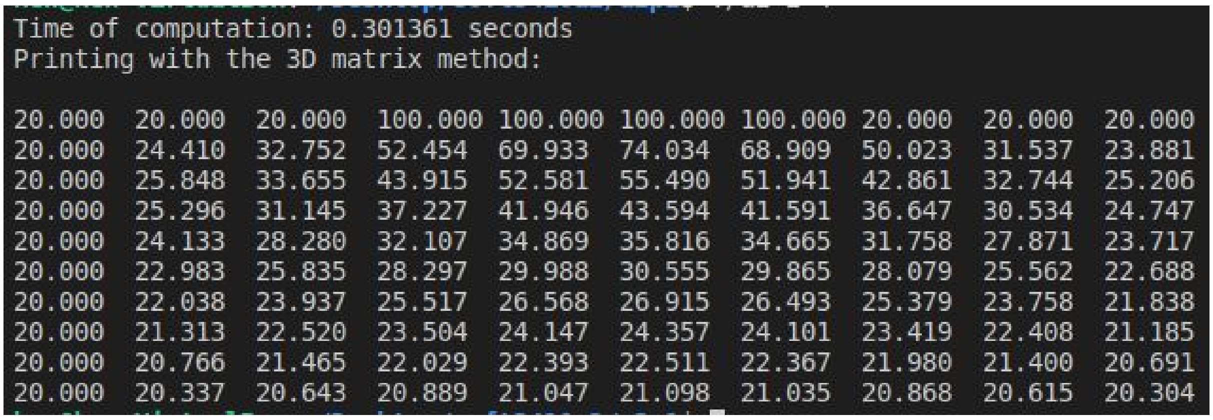
Output of Parallelised Calculation (4 threads):



```
Time of computation: 0.223751 seconds
Printing with the Parallelised Calculation:

20.000  20.000  20.000  100.000 100.000 100.000 100.000 20.000  20.000  20.000
20.000  24.410  32.752  52.454  69.933  74.034  68.909  50.023  31.537  23.881
20.000  25.848  33.655  43.915  52.581  55.490  51.941  42.861  32.744  25.206
20.000  25.296  31.145  37.227  41.946  43.594  41.591  36.647  30.534  24.747
20.000  24.133  28.280  32.107  34.869  35.816  34.665  31.758  27.871  23.717
20.000  22.983  25.835  28.297  29.988  30.555  29.865  28.079  25.562  22.688
20.000  22.038  23.937  25.517  26.568  26.915  26.493  25.379  23.758  21.838
20.000  21.313  22.520  23.504  24.147  24.357  24.101  23.419  22.408  21.185
20.000  20.766  21.465  22.029  22.393  22.511  22.367  21.980  21.400  20.691
20.000  20.337  20.643  20.889  21.047  21.098  21.035  20.868  20.615  20.304
```

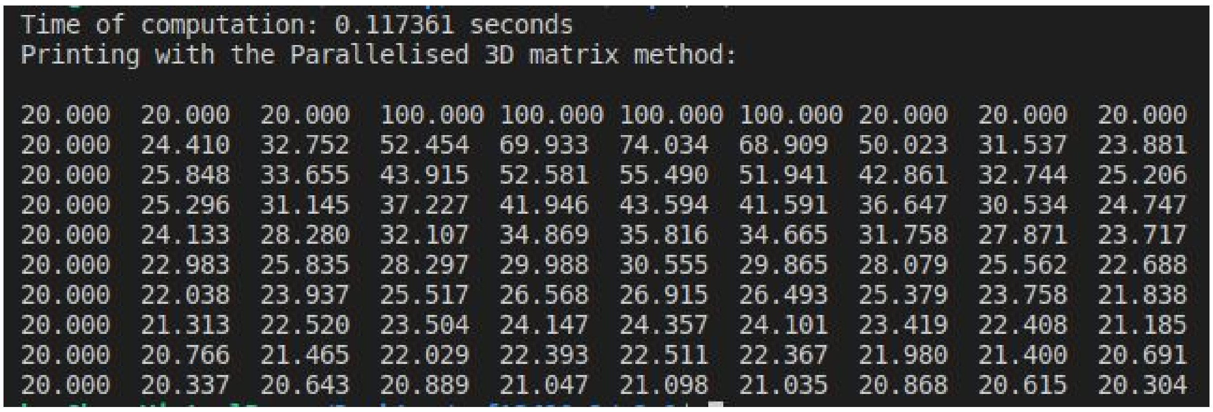
Output of the 3D Matrix version of the calculation:



```
Time of computation: 0.301361 seconds
Printing with the 3D matrix method:

20.000  20.000  20.000  100.000 100.000 100.000 100.000 20.000  20.000  20.000
20.000  24.410  32.752  52.454  69.933  74.034  68.909  50.023  31.537  23.881
20.000  25.848  33.655  43.915  52.581  55.490  51.941  42.861  32.744  25.206
20.000  25.296  31.145  37.227  41.946  43.594  41.591  36.647  30.534  24.747
20.000  24.133  28.280  32.107  34.869  35.816  34.665  31.758  27.871  23.717
20.000  22.983  25.835  28.297  29.988  30.555  29.865  28.079  25.562  22.688
20.000  22.038  23.937  25.517  26.568  26.915  26.493  25.379  23.758  21.838
20.000  21.313  22.520  23.504  24.147  24.357  24.101  23.419  22.408  21.185
20.000  20.766  21.465  22.029  22.393  22.511  22.367  21.980  21.400  20.691
20.000  20.337  20.643  20.889  21.047  21.098  21.035  20.868  20.615  20.304
```

Output of the Parallelised 3D Matrix version:



```
Time of computation: 0.117361 seconds
Printing with the Parallelised 3D matrix method:

20.000  20.000  20.000  100.000 100.000 100.000 100.000 20.000  20.000  20.000
20.000  24.410  32.752  52.454  69.933  74.034  68.909  50.023  31.537  23.881
20.000  25.848  33.655  43.915  52.581  55.490  51.941  42.861  32.744  25.206
20.000  25.296  31.145  37.227  41.946  43.594  41.591  36.647  30.534  24.747
20.000  24.133  28.280  32.107  34.869  35.816  34.665  31.758  27.871  23.717
20.000  22.983  25.835  28.297  29.988  30.555  29.865  28.079  25.562  22.688
20.000  22.038  23.937  25.517  26.568  26.915  26.493  25.379  23.758  21.838
20.000  21.313  22.520  23.504  24.147  24.357  24.101  23.419  22.408  21.185
20.000  20.766  21.465  22.029  22.393  22.511  22.367  21.980  21.400  20.691
20.000  20.337  20.643  20.889  21.047  21.098  21.035  20.868  20.615  20.304
```

When looking at the output for both versions of the code, it appears that there are no errors between the two versions compared. Meaning that the max error may be 0.

Q3:

For each version of the code, I ran the code 5 times for each thread count. I calculated the average, and tabulated the data to 3 decimal places, because I was printing the results in 3 dp as well.

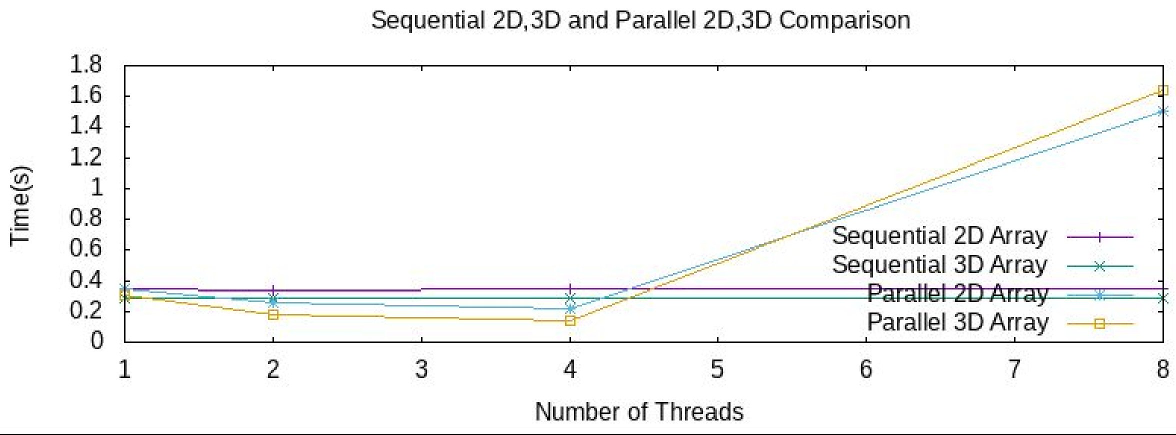
The times were measured using the method **omp_get_wtime()** at the start and end of each function, and calculated by deducting the end time from the start time, like what was done for homework 10**.**

Tabulated Results for the execution time for the four programs.

Threads	Sequential 2D	Sequential 3D	Parallel 2D	Parallel 3D
1	0.343	0.290	0.350	0.302
2	0.340	0.289	0.258	1.179
4	0.346	0.290	0.222	0.138
8	0.342	0.290	1.505	1.638

Note: The sequential versions of the code would naturally not run on more than 1 thread, I decided to tabulate the results for consistency, and to use for comparisons.

Graph of the Execution times for the Four Programs



Looking at the output, the results for the 2 sequential versions of the code were not surprising, as they were sequential the only fluctuations that occurred were due to the inherent fluctuations in runtime when running any program.

The parallelized code appeared to show steady increase in performance as the thread count went from 2 to 4, but drastically decreased at 8 threads. This is most likely due to the fact that the system I am using only has 4 cores, and the increased thread count introduces more overhead, most likely from more context switching and significantly reduces the performance. As my system only allows 1 thread per core.

Q4:

For this test, I changed the scheduling type by recompiling the code with the scheduling type being tested. So, if it was default I would have used **#pragma omp parallel for,** when changing to **Static** I would recompile the code with **#pragma omp parallel for scheduling(static, 1)** and so on.

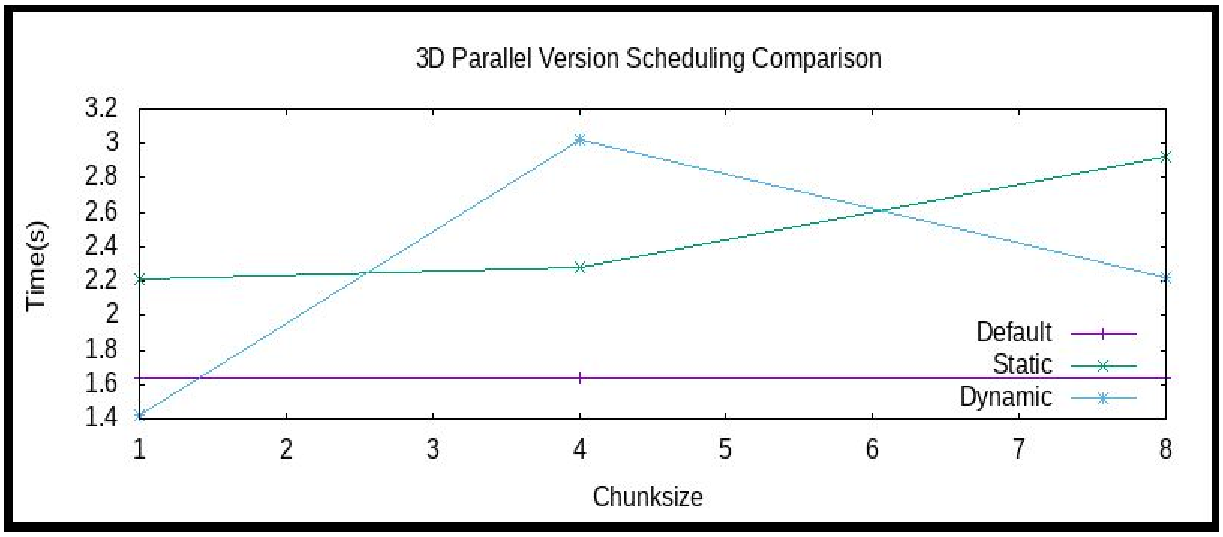
Tabulated Results for the execution time for the Different Scheduling policies and chunk sizes.

Chunksize	Default	Static	Dynamic

1	1.638	2.214	1.420
4	1.638	2.276	3.022
8	1.638	2.920	2.219

Note: I used the results from the 3D parallel version of the code as seen in Q3 for the default values as the chunk size cant be changed for the default policy so for the sake of using it as a comparison I left it as a constant value.

Graph of the Execution times for the 3D Parallel Version, using various scheduling policies and chunk sizes.



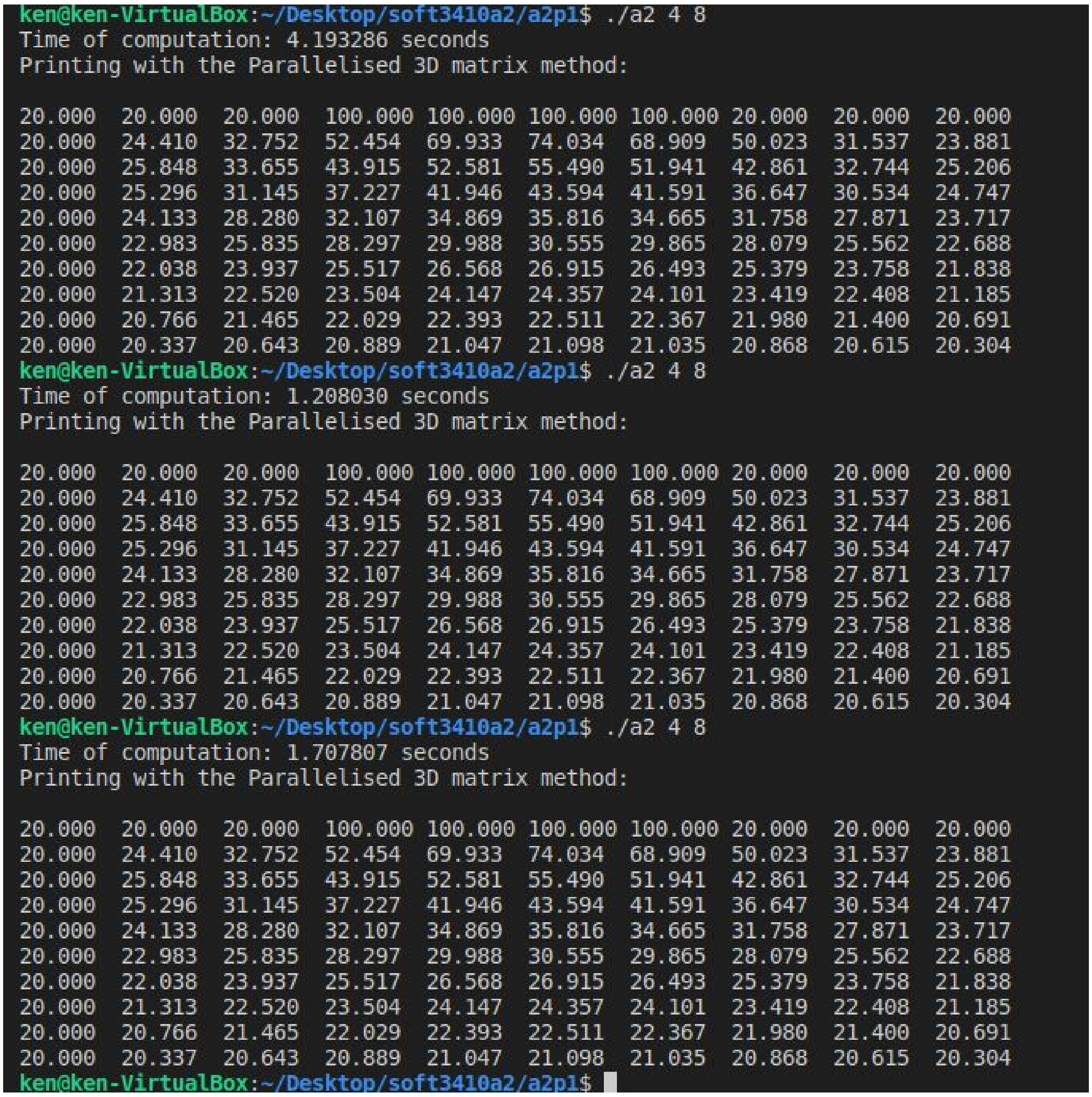
As mentioned in Q3, the most likely reason for 8 threads causing a significant reduction in performance was most likely due to my system only having 4 cores, that allows only 1 thread per core as seen in the screenshot located in the system specifications section. As a result, this may result in increased run time from overheads when using more threads than is supported.

When looking at the results, the static scheduling is running slower as the chunk size increases. This may be due to static splitting the work into some fixed n size of values to work with based on the chunk size, as some threads may take longer than others to run, and this may result in an overall increase in the runtime as a result of the synchronisation part at the end of the entire loop. And increasing the chunk size may result in that one or two threads that had to work longer than the rest, to work even longer which may be the reason for the gradual increase in the runtime as the chunk size goes up.

Although the default may appear significantly faster, when using the results from Q3, when running repeated tests on the default scheduling, it appears that the performance of default may be running similarly to what is shown in the static scheduling policy, as seen in the following page.

Given that we are working with doubles, each thread would most likely be working with just enough memory that matches the maximum size of the cache line. Resulting in only one cache line being accessed by one thread at a time, resulting in little or no false sharing.

Looking at the dynamic scheduler, because there is no order in which the chunks are being distributed to the threads, unlike static which goes in a circular order making it unlikely for threads to access the same cache line. Dynamic scheduling would request for chunks until there are none left, in I believe a greedy order, this may result in the chance that multiple threads accessing the same cache line. And this may be the reason why the 4 chunks, dynamic schedular's performance being significantly slower, as it may have been a result of false sharing.



```
ken@ken-VirtualBox:~/Desktop/soft3410a2/a2p1$ ./a2 4 8
Time of computation: 4.193286 seconds
Printing with the Parallelised 3D matrix method:

20.000  20.000  20.000  100.000 100.000 100.000 100.000 20.000  20.000  20.000
20.000  24.410  32.752  52.454  69.933  74.034  68.909  50.023  31.537  23.881
20.000  25.848  33.655  43.915  52.581  55.490  51.941  42.861  32.744  25.206
20.000  25.296  31.145  37.227  41.946  43.594  41.591  36.647  30.534  24.747
20.000  24.133  28.280  32.107  34.869  35.816  34.665  31.758  27.871  23.717
20.000  22.983  25.835  28.297  29.988  30.555  29.865  28.079  25.562  22.688
20.000  22.038  23.937  25.517  26.568  26.915  26.493  25.379  23.758  21.838
20.000  21.313  22.520  23.504  24.147  24.357  24.101  23.419  22.408  21.185
20.000  20.766  21.465  22.029  22.393  22.511  22.367  21.980  21.400  20.691
20.000  20.337  20.643  20.889  21.047  21.098  21.035  20.868  20.615  20.304
ken@ken-VirtualBox:~/Desktop/soft3410a2/a2p1$ ./a2 4 8
Time of computation: 1.208030 seconds
Printing with the Parallelised 3D matrix method:

20.000  20.000  20.000  100.000 100.000 100.000 100.000 20.000  20.000  20.000
20.000  24.410  32.752  52.454  69.933  74.034  68.909  50.023  31.537  23.881
20.000  25.848  33.655  43.915  52.581  55.490  51.941  42.861  32.744  25.206
20.000  25.296  31.145  37.227  41.946  43.594  41.591  36.647  30.534  24.747
20.000  24.133  28.280  32.107  34.869  35.816  34.665  31.758  27.871  23.717
20.000  22.983  25.835  28.297  29.988  30.555  29.865  28.079  25.562  22.688
20.000  22.038  23.937  25.517  26.568  26.915  26.493  25.379  23.758  21.838
20.000  21.313  22.520  23.504  24.147  24.357  24.101  23.419  22.408  21.185
20.000  20.766  21.465  22.029  22.393  22.511  22.367  21.980  21.400  20.691
20.000  20.337  20.643  20.889  21.047  21.098  21.035  20.868  20.615  20.304
ken@ken-VirtualBox:~/Desktop/soft3410a2/a2p1$ ./a2 4 8
Time of computation: 1.707807 seconds
Printing with the Parallelised 3D matrix method:

20.000  20.000  20.000  100.000 100.000 100.000 100.000 20.000  20.000  20.000
20.000  24.410  32.752  52.454  69.933  74.034  68.909  50.023  31.537  23.881
20.000  25.848  33.655  43.915  52.581  55.490  51.941  42.861  32.744  25.206
20.000  25.296  31.145  37.227  41.946  43.594  41.591  36.647  30.534  24.747
20.000  24.133  28.280  32.107  34.869  35.816  34.665  31.758  27.871  23.717
20.000  22.983  25.835  28.297  29.988  30.555  29.865  28.079  25.562  22.688
20.000  22.038  23.937  25.517  26.568  26.915  26.493  25.379  23.758  21.838
20.000  21.313  22.520  23.504  24.147  24.357  24.101  23.419  22.408  21.185
20.000  20.766  21.465  22.029  22.393  22.511  22.367  21.980  21.400  20.691
20.000  20.337  20.643  20.889  21.047  21.098  21.035  20.868  20.615  20.304
ken@ken-VirtualBox:~/Desktop/soft3410a2/a2p1$
```