MODULE CODE: CM3103

STUDENT NUMBER: C1545036

PARALLEL PROGRAMMING WITH OpenMP

Hardware and Software Environment:

CPU :	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Number of Cores:	4
Number of Logical Cores :	8
Operating System:	macOS Mojave version 10.14.1
GCC compiler version:	gcc-8 (Homebrew GCC 8.2.0) 8.2.0

Program was compiled and ran on my personal laptop for OpenMP Parallelisation.

Timing Experiments:

The baseline measurements would be collected for the unchanged blur.c program by running the compiled program 10 times and getting the average time for the 10 runs.Once the baseline is recorded, values for number of threads and chunksize is declared in blurOMP.c and compiled and run the same way, 10 runs and getting the averages.

Numbers of threads tested: 4,8,12,16
For each threadcount, chunksizes tested: 10,50,100,500,10000,50000.

The reasons for picking these specific chunksizes are after testing the original blurOMP program with random values there were only slight variations and to more clearly see the differences in runtimes I chose sets of low, intermediate and relatively large chunksizes to possibly more clearly visualize how it would impact the run times.

Timing Results:

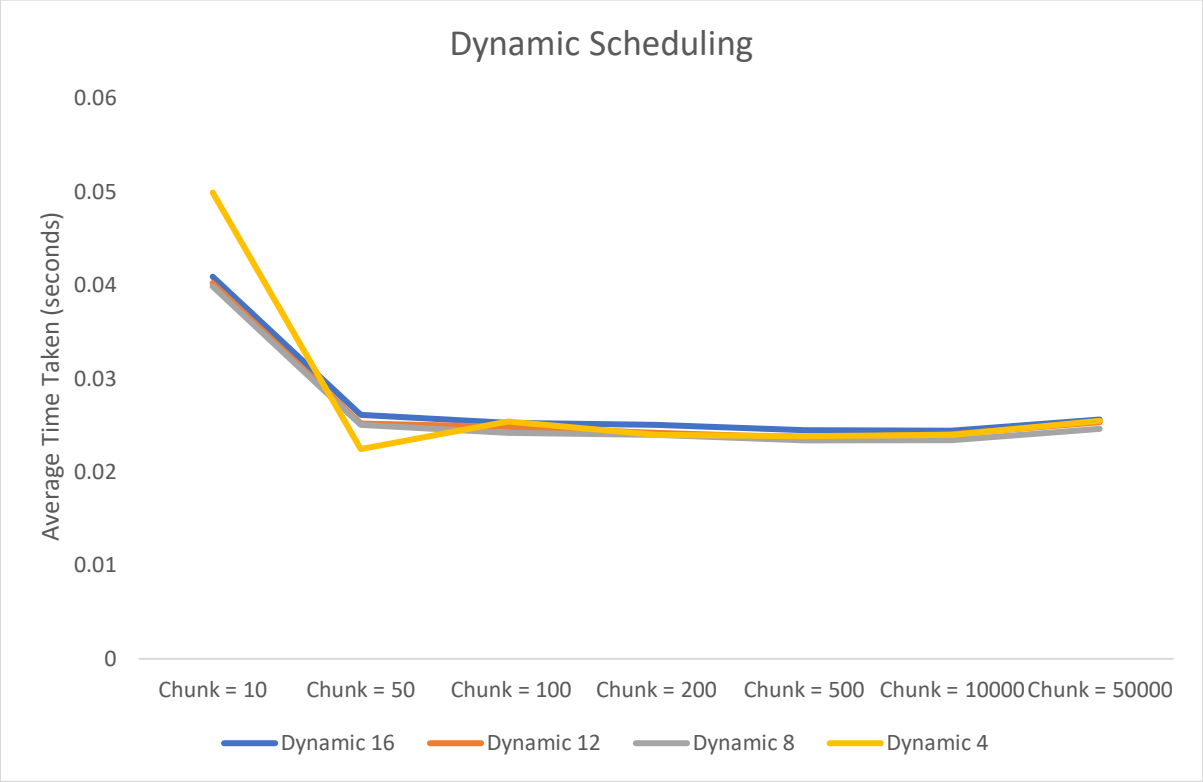

Figure 1. Graph of Average Runtime for Dynamic Scheduling in OpenMP

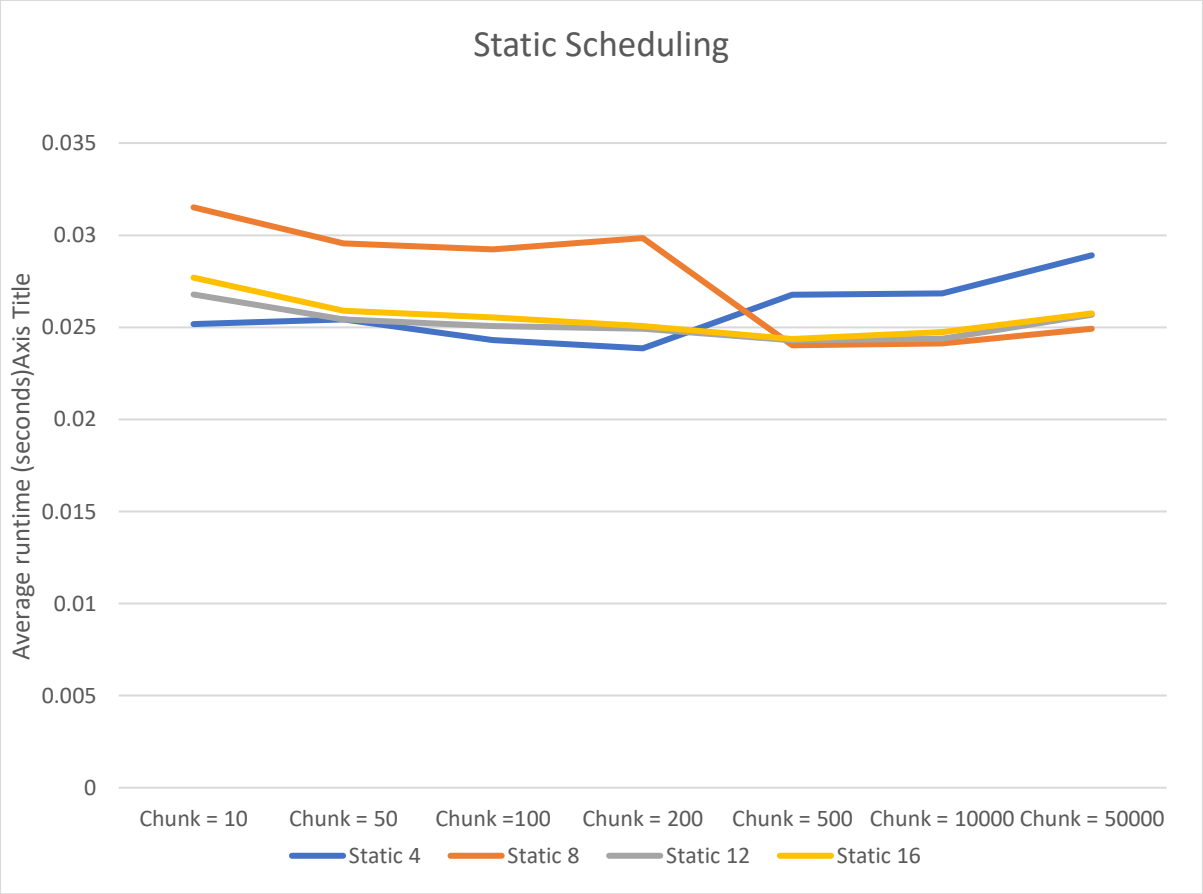

Figure 2. Graph of Average runtime for Static Scheduling in OpenMP.

Average Runtime for Non-parallelised code: 0.044143 seconds
With a standard Deviation of : 0.002216

Discussion:

From the values collected and visualized in the graphs, it is obvious that the Non-parallelised code runs slower than when it is parallelized regardless of Threadcount and Chunksizes. However, an anomaly occurred when conducting the Dynamic scheduling test for a threadcount of 4 and with a chunksize of 10, where it exhibited an average runtime much higher than that of the non-parallelised code.

Comparing how Thread counts and chunk sizes affect runtimes in Dynamic scheduling methods, they all show the same trend regardless of threadsize, chunk size is a determining factor in how fast the program runs. From my initial testing phase I chose to run a relatively small chunk size of 10 that showed it did hinder performance almost doubling in the time taken to run. However as the chunk sizes increase, the gain in performance is minimal when regardless of the number of threads, but 8 threads and 500 chunks showed the best overall performance. Differences in the values between the different number of threads are so minimal mostly due to how modern machines are able to run so much faster. Dynamic scheduling mostly maintained the same trend regardless of Thread count and chunk size due to how it only gives work to a thread that needs it, so it will work on the next chunk iterations.

For Static Scheduling, the first thing that strikes out are the values for the 8 threads runtime being significantly higher at lower chunksizes. This could be due to programs running in the background during one of the runs that hindered the runtimes. Once it is passed the 200 chunk size mark it behaves normally like threadsize of 12 and 16, however 4 threads was the only one that jumped significantly higher than the rest after 200 chunks. The reason 8 threads would have taken so long could be due to the time taken to set up the threads and being that the chunk size were so small, they spent more time setting up the threads to do only a small amount of work. Whilst the other thread and chunk size combinations show better compatibility in terms of thread size to the amount of work. 4 Threads proves this point due to its jump in run time after the chunk size of 200 as there are only 4 threads that need to work on these chunks whereas the others had many more threads to split up the work. The fact that static scheduling also only work in a round-robin order will only slow down the run times for 4 threads when the chunk size gets increasingly bigger.

Conclusions & Reflection:

After comparing the results I can conclude that for both scheduling methods, it would be best to use a chunk size that is within an intermediate range of 200-500 for the best results. Any larger chunk size would lead to slow downs.

As for thread count I would recommend using double the amount of physical cores for both scheduling methods. In this case it would be 4 x 2 = 8 threads. Despite Static scheduling showing a high runtime for 8 threads, the fact that the best chunk size lie between 200 and 500 could show a significant drop when paired with 8 threads. Dynamic scheduling performed the best with 8 threads, hence the recommendation.

Some aspects that could be better improved during testing would be a better runtime environment, without any background programs running.