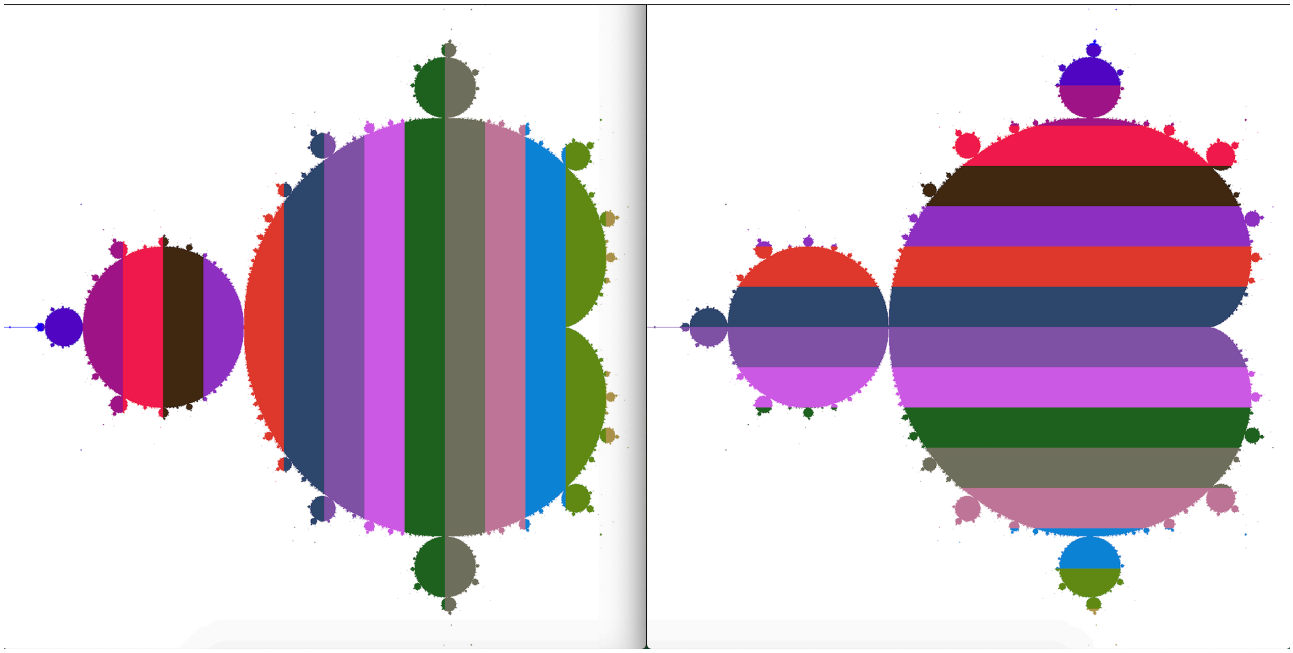
Parallel Computing - SS20
Assignment 2
Report

Task 1 - Mandelbrot using OpenMP

The serial Mandelbrot example was implemented into two parallel variants using OpenMP in C++.

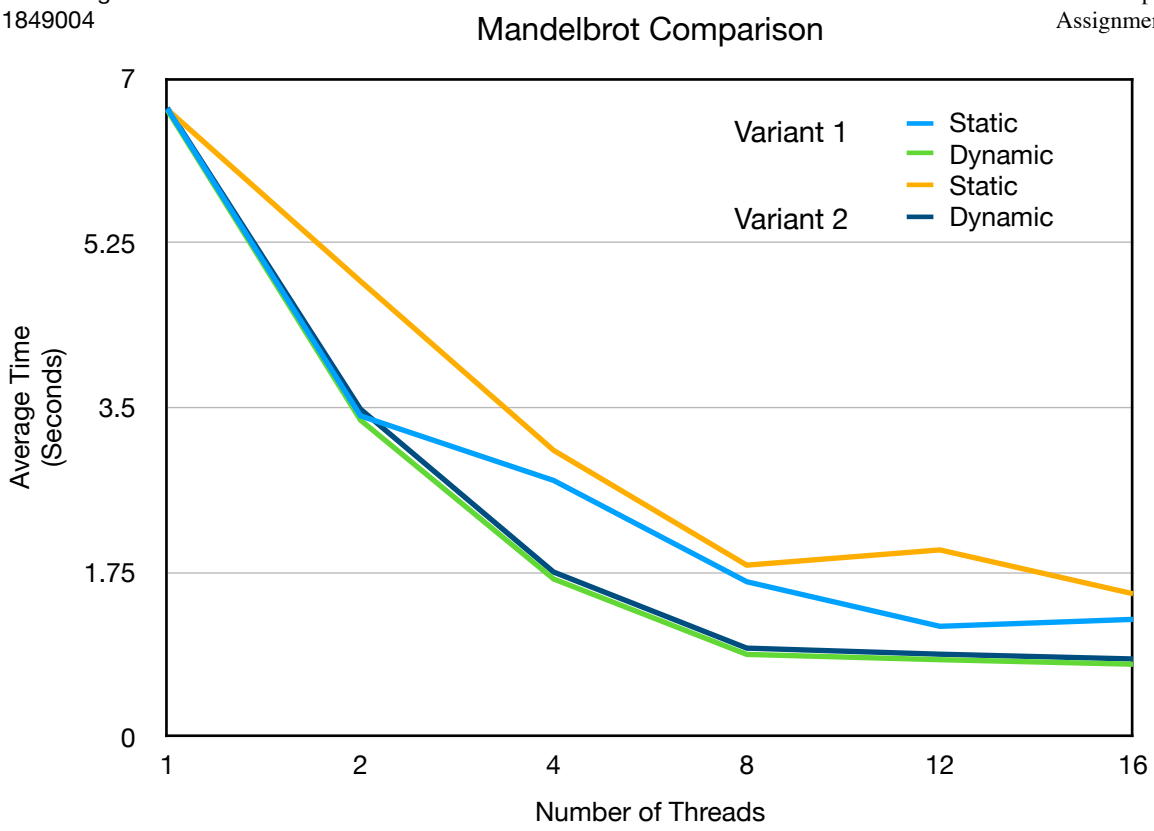


Variant 1 Static

Variant 2 Static

The “#pragma omp parallel for” from OpenMP was used to create a specified number of threads (from 1 to 16 as supported by Cora) and to make a worksharing parallel model by placing on the outer for loop (in case of Variant 2) and inner for loop (in case of Variant 1) while taking care of the memory model of OpenMP with explicit declaration of private and shared variables among the threads so that no pixel loss (race condition) happens in the Mandelbrot algorithm and a significant speedup from sequential code could be achieved.

Each colour band in the mandelbrot.ppm represents the contribution of each thread in the total work done in the parallel program. The performance of each Variant was further tested with two of the most common schedule clauses for “#pragma omp parallel for” - schedule(static) and schedule(dynamic). The execution times graphs for both the variants with different number of threads (from 1 to 16) along with two scheduling clauses have been recorded in detail in the Raw Data file.



The above performance graph gives an overview of the speedups obtained from the both the variants along with different Number of Threads settings.

Using the maximum number of threads of concurrent threads available =16 in Cora and with a dynamic schedule, both the parallel variants were able to obtain a speedup of 8-9 from the sequential mandelbrot code which is a significant improvement. The static scheduled versions obtained a speedup of 5.

Also, it was observed that the static scheduled version of Variant 2 was about 25% faster than the static scheduled version of variant 1 while the dynamic versions had almost the same performance as it can be seen in the green and black lines in the graph. This is because the static schedule naively alots each thread an equal chunk of iterations of the parallel for loop whereas in the dynamic version, as soon a thread becomes idle it takes up the next available loop iteration for a faster execution.

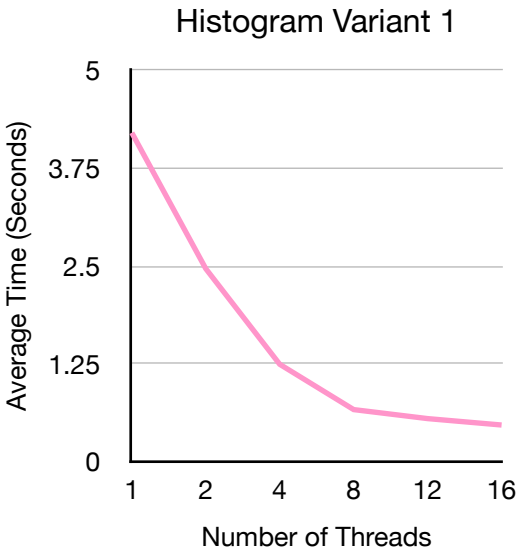
Task 2 - Parallel Histogram using OpenMP

In this task the sequential histogram code was modified to create the parallel versions using OpenMP with c++ to obtain faster execution times. I experimented with various OpenMP versions and the way how the executed version of histogram obtained significant speedups was by extending the histogram and worker structs in a way that each thread execution writes its data to a separate histogram vector and in the end all the histogram data vectors from all the parallel executions are combined to obtained the final histogram. This not only avoids any race conditions but also make the execution much faster than the sequential histogram program.

1. Using SPMD style:

"#pragma omp parallel" was used to create a number of specified threads (1 to 16 on Cora) which all run the same block of code which was written in such a way that each thread calls the worker method with only a specifically divided chunk of data from repeats_to_do and there is no overlap of data within the threads and the random number generated from every repeats_to_do is placed in its correct bucket in the histogram using the parallel threads.

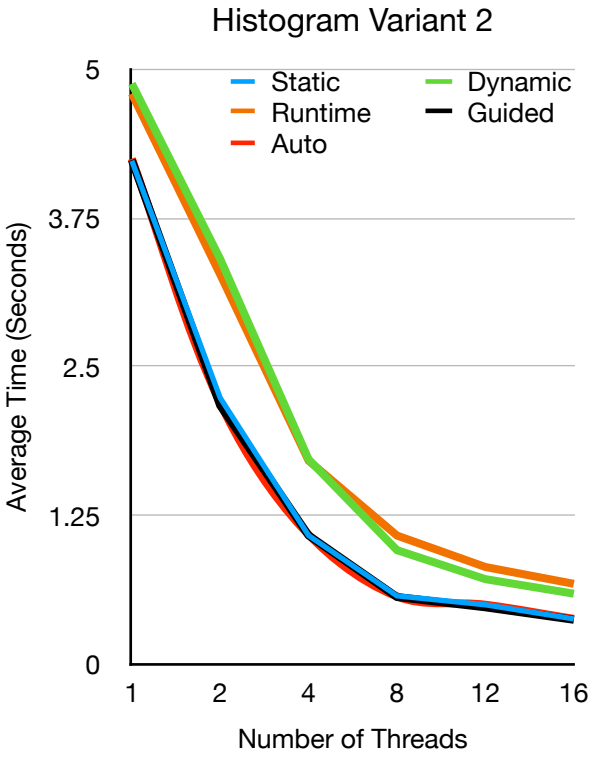
With the maximum of 16 threads supported by Cora, this variant of parallel histogram program was able to obtain a speed up of 9-10 from its sequential version which is a significant improvement.

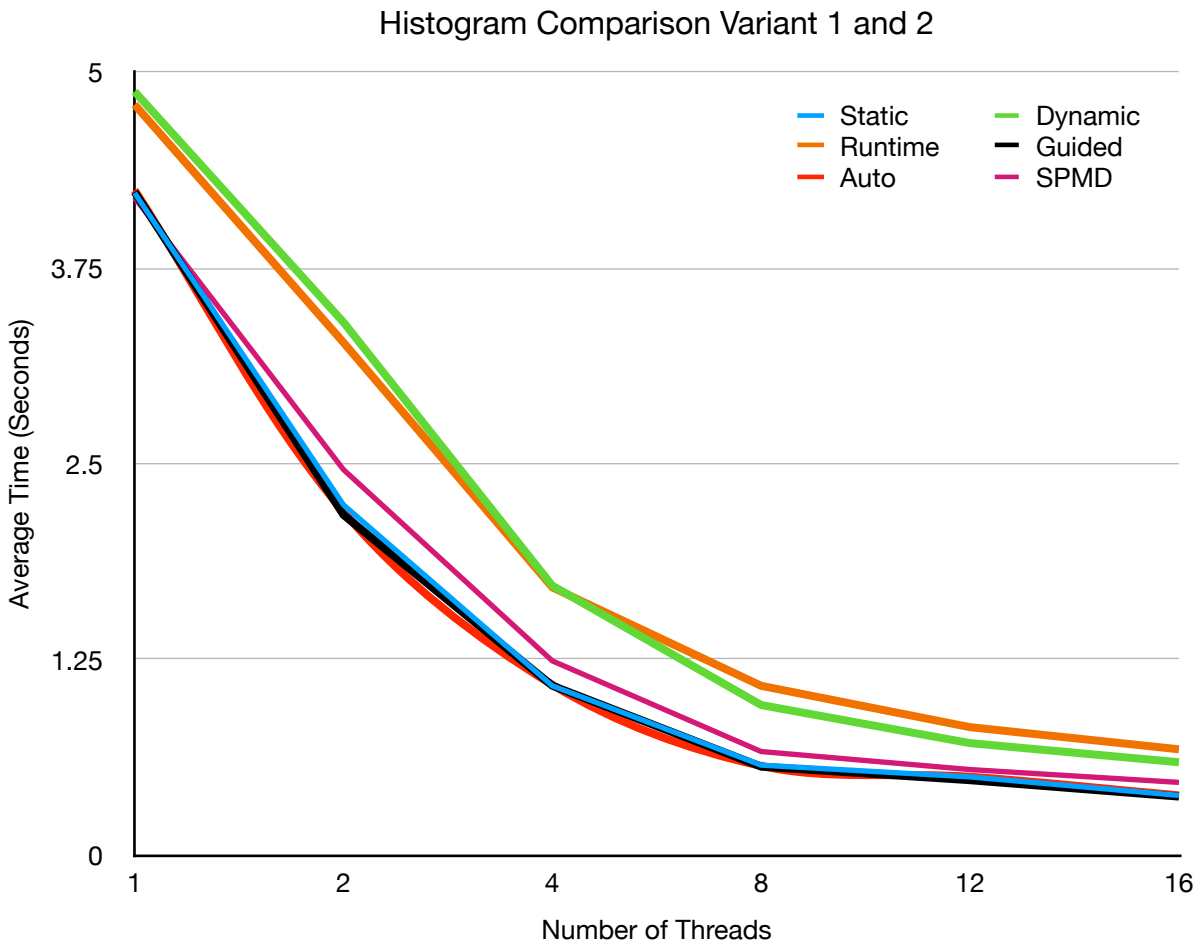


2. Using OpenMP Worksharing

"#pragma omp parallel for" was used to create a number of specified threads (1 to 16 on Cora) which shared the iterations of the for loop executing all the repeats_to_do of the histogram. Every iteration created a random number which was then placed into its correct bucket in parallel using the OpenMP created threads. This for loop divided its work based on various scheduling clauses - static, dynamic, runtime, guided and auto.

With the maximum of 16 threads supported by Cora, this variant of parallel histogram program was able to obtain its fastest speed up of 12 using the guided schedule clause.





The above graph shows an overview of the performance comparisons of all the versions from variant 1, 2a, 2b, 2c, 2d and 2e. It can be observed that as the number of threads increase there are more threads available to perform the work in parallel and save execution time. Also, as the number of threads increase the difference in the execution times become smaller and smaller.

All the experiment recording data comprising of execution times of different versions from both the Tasks along with their graphs can be seen in detail in the Raw Data file.