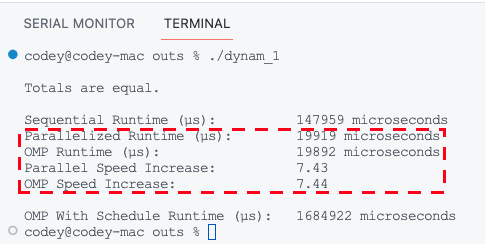
SIT315 M2.S3P: Multithreading with OpenMP Compiler Directives

Codey Funston | S222250824

No Group

Activity 1 – Parallel Vector Addition - OMP

Since I implemented my pthreads version of the program in a way such that threads take a chunk of the data and then continually request more chunks until the data is fully processed meant that the performance was like OMP. This is because OMP performs this way by default ie using the *omp paralell for* clause *schedule(dynamic)*. The results are under “OMP Runtime (μs):” or can be compared by viewing the speed increase factors for “Parallel” (pthreads) and “OMP”. As you can see, for this execution of the program, the two parallel implementation methods produce nearly identical runtimes:



Activity 2 – Parallel Vector Addition Part 2 – OMP++

The main scheduling options with OMP are dynamic or static with a chunk size specification. With static scheduling, threads are assigned chunks of a parallel part of the program with round-robin scheduling. This means that if the chunk size is very large not all threads may be used. With dynamic scheduling it is like how I used pthreads as mentioned in activity 1 above. Below are the outputs for different scheduling types, with executable names as *<static/dynamic>\_<chunk>*:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

The main observation I had was that when the chunk size was larger with dynamic scheduling there was a speed up. This agrees with the observation in the previous VectorAdd project with pthreads where dynamically grabbing chunks was more effective when the chunk size was slightly larger since then the proportion of the time spent locking and getting the new chunk was less compared to the overall operation.