

SIT315 M2.S3P: Multithreading with OpenMP

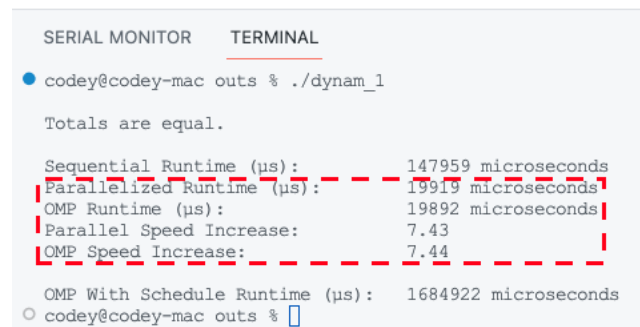
Compiler Directives

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 parallel 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:



```
SERIAL MONITOR  TERMINAL
● codey@codey-mac outs % ./dynam_1

Totals are equal.

Sequential Runtime (μs):      147959 microseconds
Parallelized Runtime (μs):    19919 microseconds
OMP Runtime (μs):             19892 microseconds
Parallel Speed Increase:      7.43
OMP Speed Increase:           7.44

OMP With Schedule Runtime (μs): 1684922 microseconds
○ codey@codey-mac outs %
```

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>`:

```

codey@codey-mac outs % for file in *; do
echo "--- $file ---"
./"$file"
done
--- dynam_1 ---

Totals are equal.

Sequential Runtime (µs):      148062 microseconds
Parallelized Runtime (µs):    19240 microseconds
OMP Runtime (µs):            20190 microseconds
Parallel Speed Increase:      7.7
OMP Speed Increase:           7.33

OMP With Schedule Runtime (µs): 1716371 microseconds
--- dynam_20 ---

Totals are equal.

Sequential Runtime (µs):      129734 microseconds
Parallelized Runtime (µs):    19351 microseconds
OMP Runtime (µs):            19504 microseconds
Parallel Speed Increase:      6.7
OMP Speed Increase:           6.65

OMP With Schedule Runtime (µs): 104503 microseconds
--- dynam_4 ---

Totals are equal.

Sequential Runtime (µs):      128659 microseconds
Parallelized Runtime (µs):    19139 microseconds
OMP Runtime (µs):            19838 microseconds
Parallel Speed Increase:      6.72
OMP Speed Increase:           6.49

OMP With Schedule Runtime (µs): 443812 microseconds
--- dynam_8 ---

Totals are equal.

Sequential Runtime (µs):      129677 microseconds
Parallelized Runtime (µs):    19335 microseconds
OMP Runtime (µs):            19867 microseconds
Parallel Speed Increase:      6.71
OMP Speed Increase:           6.53

OMP With Schedule Runtime (µs): 232744 microseconds

```

```

--- stat_1 ---

Totals are equal.

Sequential Runtime (µs):      129744 microseconds
Parallelized Runtime (µs):    19722 microseconds
OMP Runtime (µs):            19779 microseconds
Parallel Speed Increase:      6.58
OMP Speed Increase:           6.56

OMP With Schedule Runtime (µs): 26346 microseconds
--- stat_20 ---

Totals are equal.

Sequential Runtime (µs):      128216 microseconds
Parallelized Runtime (µs):    20108 microseconds
OMP Runtime (µs):            20052 microseconds
Parallel Speed Increase:      6.38
OMP Speed Increase:           6.39

OMP With Schedule Runtime (µs): 28218 microseconds
--- stat_4 ---

Totals are equal.

Sequential Runtime (µs):      128921 microseconds
Parallelized Runtime (µs):    19109 microseconds
OMP Runtime (µs):            19650 microseconds
Parallel Speed Increase:      6.75
OMP Speed Increase:           6.56

OMP With Schedule Runtime (µs): 28036 microseconds
--- stat_8 ---

Totals are equal.

Sequential Runtime (µs):      130263 microseconds
Parallelized Runtime (µs):    18790 microseconds
OMP Runtime (µs):            19811 microseconds
Parallel Speed Increase:      6.93
OMP Speed Increase:           6.58

OMP With Schedule Runtime (µs): 26105 microseconds
codey@codey-mac outs %

```

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.