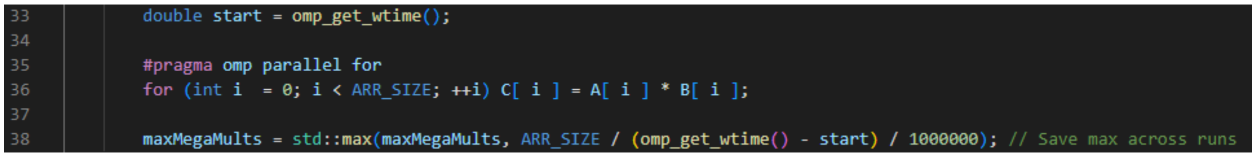
Lyon Kee

keel@oregonstate.edu

Project #0: Simple OpenMP Experiment

Key snippets of code



```
33          double start = omp_get_wtime();
34
35          #pragma omp parallel for
36          for (int i  = 0; i < ARR_SIZE; ++i) C[ i ] = A[ i ] * B[ i ];
37
38          maxMegaMults = std::max(maxMegaMults, ARR_SIZE / (omp_get_wtime() - start) / 1000000); // Save max across runs
```

Line 35: #pragma omp parallel for allows us to parallelize for block to run on a specified number of threads which is set prior to it.

Line 38: We obtain save the peak performance calculated with iterations / time / 1000000 (Mega scaling).

Speedup = (Peak performance for 4 threads)/(Peak performance for 1 thread) = 1022.72/3980.47

= 3.8920427878598245

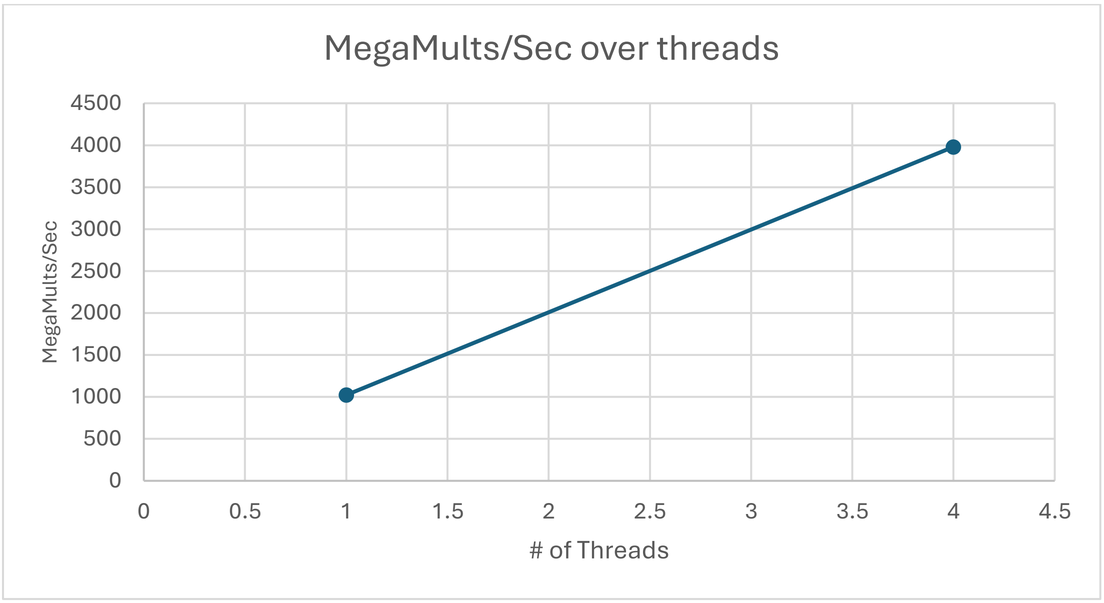
Fp = (4/3)*(1 - (1/S)) = (4/3)*(1 - (1/3.8920427878598245))

= 0.9907540231848666

Tables of data

# of Threads	Peak Performance(Mega-Multiplies per Second)
1	1022.72
4	3980.47

Graphs of data



Conclusion

Things turned out pretty well, I tested with different threads and array sizes and at the extreme levels they are odd. For example, in normal levels of array sizes below 2^20, we obtain a speedup that is slightly above 4, most likely due to floating point precision issues. At array size of 2^25, we obtain a speedup that is slightly below 4 which seems more normal. When running on array sizes that is very small, we would obtain a significantly bad speedup likely due to the overhead cost of parallelizing.

When performing the same task with 16 threads on a computer that has 16 threads, it is observed that there is very minimal speedup as compared to 4 threads, this is likely due to the scheduling issues when running on too many threads.

1. Tell what machine you ran this on

This is my own laptop ran on windows in a bash shell with a 11th Gen Intel® Core™ i7-11800H @2.30GHz.

2. What performance results did you get?

Threads = 1, Peak Performance = 1022.72 MegaMults/Sec

Threads = 4, Peak Performance = 3980.47 MegaMults/Sec

3. What was your 1-thread-to-4-thread speedup?

3.8920427878598245

4. Your 1-thread-to-4-thread speedup should be less than 4.0. Why do you think it is this way?

When creating threads there are overhead costs and when running on more threads there will likely be more thread scheduling which causes each thread to have more downtime, this causing the speedup to be less than 4, but it is observed that my system is able to obtain values that is very close to 4 because there are no significant loads running with the program.

5. What was your Parallel Fraction, Fp? (Hint: it should be less than 1.0, but not much less.)

0.9907540231848666