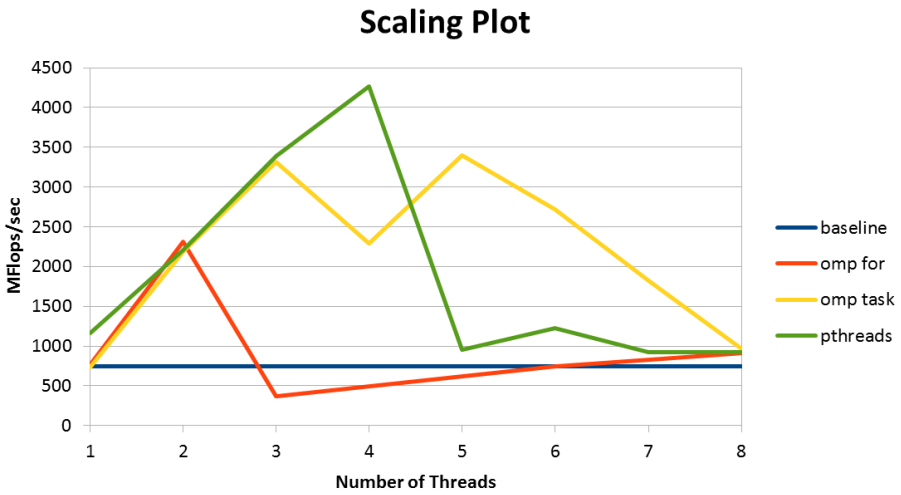
Vy-An Phan
26159888
cs195-15-aii

Assignment 2

3.2 Matrix Multiply

a) Plot: omp for – omp task – pthreads



b) The peak occurred in pthreads and was about 6 times better than the scalar baseline. However, scaling is not linear: after about 3-4 threads, we get a huge dropoff in speed. This is because previously, the computations all fit in the L2 cache, getting an advantage from spatial locality. However, having too many threads will cause them to overwrite each other's caches and fight one another over limited memory space. Having to run back and forth between L3 and main memory slows things down significantly.

c) I'm getting around 4.2 Gflops at best, which is much less than the peak performance of the hive machines. Peak performance assumes every single ALU is active at once and all available memory is in use, but the program I am running is not the only one the hive machine is dealing with, meaning we don't have that luxury. To improve performance, we can divvy up the matrices into blocks that we know for sure will fit into one level of cache to minimize the amount of memory fetching we have to do.

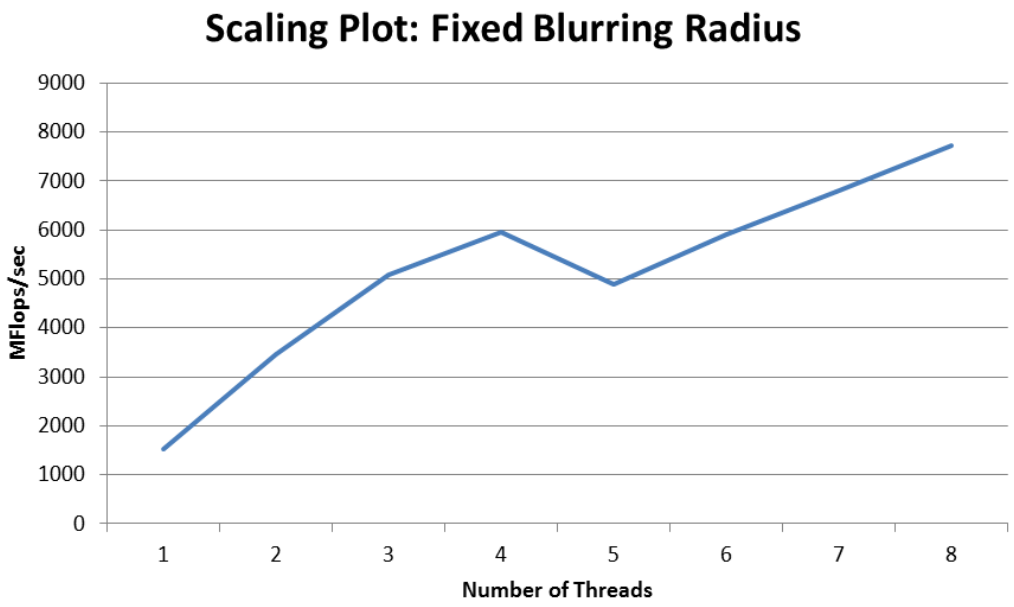
d) I found placing the OpenMP markers to be more intuitive, even though there were more commands I had to learn. Creating and joining pthreads just worked like magic, which, while simple, bothered me.

e) OpenMP programs work through threads. Initially, the program begins at a single process, which operates sequentially until it comes to a region that can be parallelized. Then, it splits off into teams of parallel threads, and the subroutines in that region are executed in parallel. When the work there is done, they rejoin to form the master thread.

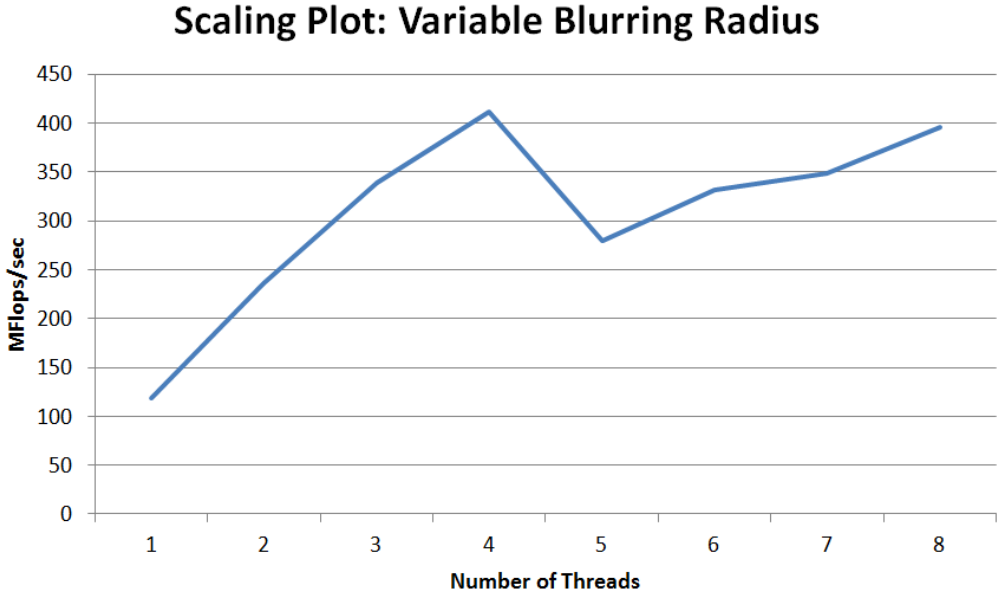
f) First, OpenMP generates an intermediate ssa file and marks off the areas where we have implemented parallelization (the pragma omp parallel stuff). These areas will receive special commands that will tell the processor where to allocate threads for them, and later rejoin them, on execution. The intermediate file goes through another round of processing where sections are translated to do/for constructs, barriers are made explicit, and further optimizations (merging adjacent regions, etc.) are carried out. Finally, all of these constructs are translated once more into multithreaded code. The computer will also set up for storage and initialization in this last step.

4.1 Image Blurring

a) Plot: radius n=1



b) Plot: radius n=10



Both the fixed and variable blurring radii have similar scaling, but the variable blurring definitely does worse. Fixed radii had nearly linear scaling, while variable barely managed to improve past the dip at 5 threads. This is because the program is limited by memory – a large radii requires many more accesses. To improve scaling, we can try to improve data locality and minimize cache misses by partitioning the whole matrix into blocks of data so that they're small enough to end up in the same cache.

4.2 Time Spent on Assignment

a) This assignment took me a few hours, mostly from learning the basic OMP commands and generating my charts.