Claude Shyaka
Cameron Whipple
CSE539 – Concepts in Multicore Computing
March 28, 2018

Project 3: Performance Engineering Matrix Multiplication

The goal of this project was for us to apply the Performance Engineering process, as discussed in class, to a real-world application. For the sake of this project, the real-world application to be optimized was floating point matrix multiplication.

For the experiments we performed, we kept the size of the matrix static at 2048 x 2048 and altered the number of processor cores. We chose 2048 for matrix size as it was sufficiently large to force interactions with the cache hierarchy and underlying memory sub-system. The number of processor cores, or workers, was varied from experiment to experiment but were within the range of 1 to 16.

From analysis of the application code, we noticed that span of the algorithm dominated and that the cache was underutilized. Our first optimization was aimed at decreasing the dominance of the spawns and improve cache utilization. We accomplished this by tiling, or coarsening, the work load completed on a given worker. A threshold value was defined that resulted in sequential execution rather than parallel. This reduced the span of the algorithm and allowed more efficient use of cache hierarchy. The results of our optimization can be seen in Table 1.

course_value	proc_count	matrix_size	schedule_time	working_time	idle_time	elapsed_time
1	16	2048	0.784068	620.71789	37.429405	18.928991
2	16	2048	0.457722	97.055354	17.895065	3.320704
4	16	2048	0.529652	23.060553	8.462912	0.930575
8	16	2048	0.231931	11.51605	5.879771	0.514582
16	16	2048	0.500934	11.507401	6.490923	0.54085
32	16	2048	0.803216	27.106667	9.32793	1.077931
128	16	2048	0.708274	34.498744	10.108202	1.322424

Table 1: Run of the divide and conquer with coarsened base case optimization implemented.

Next, we noticed significant number of cache misses within the application. To solve this, we performed an experiment to validate that Blocked Morton-Z would provide benefit. As result of experiment, we implemented Blocked Morton-Z in combination with the coarsening optimization previously discussed. The Blocked Morton-Z format provided us around 0.15 second improvement. The Blocked Morton-Z also resulted in the coarsening threshold increasing to 16 as a result of the improved cache utilization and latency hiding from pre-fetcher.

After the Blocked Morton-Z, we noticed a significant number of remote DRAM accesses were taking place when going from 8 to 16 processors cores. We developed an experiment to verify our prediction that memory pages were being allocated on single processor. We enabled the NUMA memory interleaving policy for all configurations and re-ran our experiments. The results showed significant decrease in number of remote DRAM accesses. However; the results also showed that enabling the interleaving policy for configurations with 8 or less processors reduced performance as expected. As such, we implemented a hybrid approach within the application where interleaving is only utilized with more than 8 processors. As a result, we saw an improvement in both the execution time and the number of remote DRAM accesses.

Due to time constraints, we were not able to complete all the optimizations that we would have liked. We started a Strassen based implementation after research showed that it provided a better asymptotic bound than the divide and conquer. However; we were not able to make significant process due to time. We also looked into the possibility of implementating matrix multiplication in pthreads in order to have more control over memory pages and thread assignment. This possibility was considered after seeing the number of remote DRAM accesses still taking place with hybrid policy incorporated.

In summary, we applied three optimizations to the matrix multiplication application. First, we tiled/coarsened the base case to reduce span and improve cache utilization. Secondly, we implemented Blocked Morton-Z in order to enlist pre-fetcher assistance and further improve cache utilization. Third, we incorporated a hybrid NUMA policy to reduce remote DRAM accesses when more than 8 cores are utilized. As a result of these optimizations, the execution time went from 18.92 seconds to 0.39 seconds for a 2048 x 2048 matrix.

Team Dynamic

There are two of us in the group for the projects. One of us is a full-time student while the other works full time in addition to coursework. For this project, we broke up the experiments and trade studies we performed on the divide and conquer approach evenly. We then split up implementation of the Blocked Morton-Z, Memory Interleaving Policy, and Strassen between the two of us. We utilized shared drive in order to ensure that experiment results were accessible from all members. The report and analysis were a team effort between both members.

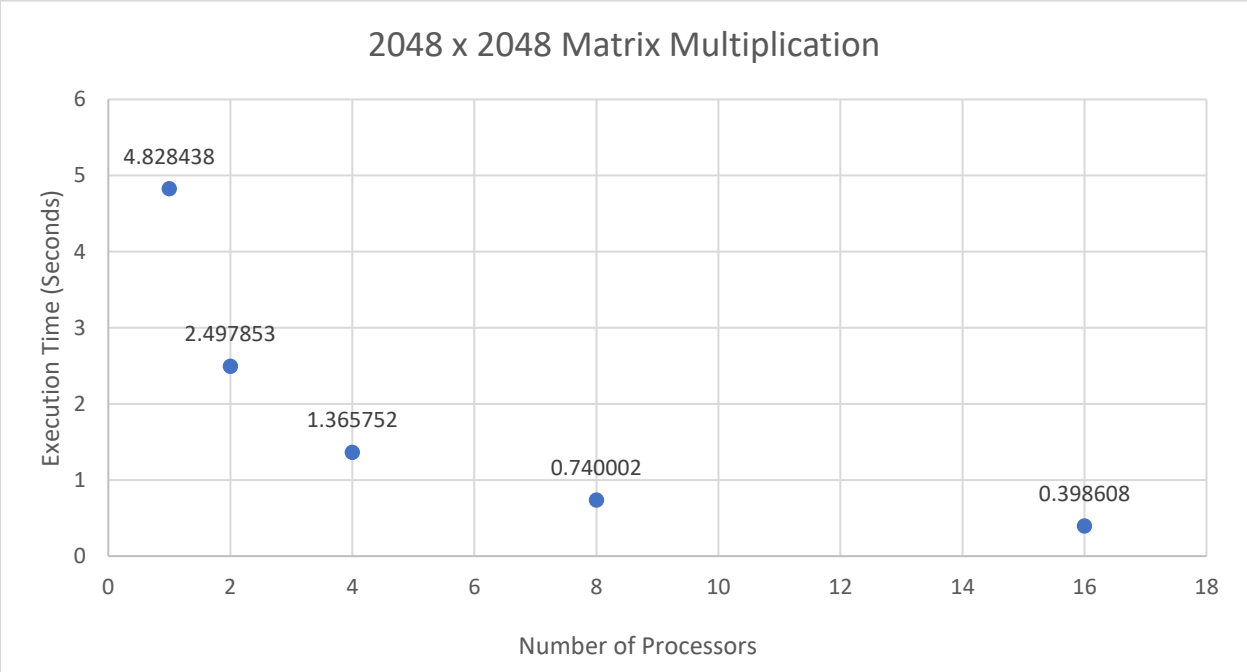

Figure 1: Running time for Blocked Morton-Z and hybrid memory allocation policy