CPE 315 Lab #5 Report – Cache Optimization

In this lab I manually optimized the matmul (matrix multiply) code to exploit cache locality and record the results. The results and discussion are detailed below.

Results

Original Code	-O0	-O1	-O2	-O3
Runtime	202.34877439	166.076212079	159.971429077	161.545491659
Cache Misses	1,084,140,262	1,079,980,413	1,080,257,277	1,081,763,365
Cache References	22,770,521,309	3,383,834,986	3,382,122,281	3,382,311,965

Table 1: Comparing original code performance at various gcc optimization levels

With Best Compiler Optimization	Original (-O2)	Column-major (-O1)	Tiled 16x16 (-O3)	Tiled 32x32 (-O2)
Runtime	159.97142908	14.123902489	13.929938737	12.740925229
Cache Misses	1,080,257,277	5,722,087	11,378,444	6,578,498
Cache References	3,382,122,281	3,384,556,693	3,381,367,003	3,381,796,674

Table 2: Comparing the best compiler optimization runs of the original and manually optiimzed code

Discussion

Comparing the results obtained measuring the performance of the original code at different optiization levels, it is clear that cache references contributes to the runtime somewhat. This conclusion can be made since the cache misses at each optimization level has a consistent count of cache misses. For example, when the number of cache references decreased in an order of magnitude from -O0 to -O1, the runtime dropped sharply by about 40 seconds.

Comparing the results from running the manually optimized code, it is clear that cache misses are very strongly correlated to runtime while being slightly dependent on the number of cache references. It is quite clear from that the penalty of the cache misses is a significant contribution to runtime compared to the amount of time it takes to refrence the cache billions of times. For example, the original code running at O2 had three orders of magnitude more cache misses and that accounted for a 145 second increase in runtime. While cache misses play a big role in the runtime, it can still be offset by the number of cache references. For example, the number of cache misses of the Column-major run was half of that of the Tiled 16x16 run, but the runtime was slightly faster in the tiled run due to the number of cache references.

While the compiler optimization is quite useful at speeding up code using tried-and true techniques such as loop unrolling or code restructuring, it cannot be compared to manually optimizing the code where the developer can change the execution order and much more which breaks many floating point operations like in matmul.