ECE454, Fall 2015

<u>Homework 2: Memory Performance</u>

Team Name: Team Herbert!!!

<u>Summary of "Rotate (current working version)"</u>

Optimizations	**Speed up** (on UG Lab Machine only)
-Loop Interchange (ij to ji) -Loop Tiling/Blocking (16x) -Function Inlining -Loop Invariant Code Motion -Multiplication to Addition expression	Max: 2.9 **<u>Avg: 2.8</u>**

Since we know that when the data size get very large (n=2048, 4096), we cannot fit all data into the cache and cache misses are expected. But if we use tiling to reduce the data amount that need to be calculate each time then they can be fit into the cache or number of cache misses will reduce. After numbers of trial and errors, firstly we found that exchanging the i and j in the loops give us some speed up. This is due to trade-offs for read speed from the source and write speed to the destination. Originally the read is row-wise and after the switch it becomes column-wise, and opposite for the write, thus the column-wise operation will have more miss in the cache. Here, we suspected that write speed probably took less cycles than read speed. When deciding tile size, we found that block size of 32 * 32 gives us the best performance when dimension of the matrix becomes large. Given that UG machine's cache line size is 64 byte (=16 words), we use block width 32 and height of 32 will have good overall performance in both small and large data. And larger block size can make less branch prediction. Using loop unrolling (16 times) also increased performance, because it reduces loop overhead time and increase the instruction-level parallelism. Lastly, our optimization involved loop invariant code motion which inlined the RIDX function inside the most inner loop. Since compiler (-O2) will not look inside the procedure calls, we therefore inlined the RIDX function and did manual optimization. After that, moving multiplication parts out of the most inner loop(local variable vs. memory access) and transforming to additions has shown much improvement in performance due to reduced CPU cycles. (multiplication vs. addition)