# CS 575, Project 4: Vectorization

Peter Ferrero, Ph.D. Student, Oregon State University

May 21, 2018

Project 5 was ran on a Windows Desktop with an AMD FX(tm)-8320 Eight-Core 3.50 GHz Processor with 16.0 GB of installed RAM. The Intel compiler was used to run each experiment. The goal of this experiment is to explore the affect of vectorization on SIMD data level parallelization. An array of random numbers between 0 and 1000 was set up for each run and the square of each element of the array was calculated and assigned to a new output array. This allowed for the loop to be vectorized by the compiler. Two versions of this operation were performed. The first with vectorization turned off, and the second with it enabled. Output reports were generated and examined in both cases to ensure vectorization was properly controlled. The time for calculating the square root of the array was reported in MegaRoots/sec. Each experiment was repeated 20 times to examine the consistency of the data. The peak performance results for the cases with and without vectorization are summarized in Tables 1 and 2 below.

Table 1: The peak performance calculating the square root of an array without vectorization for various numbers of threads and array sizes. Performance values are given in MegaRoots/Sec.

| Array Size/Threads | 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| 1000 | 349.5 | 349.5 | 349.5 | 466.0 | 524.3 |
| 4000 | 310.7 | 508.4 | 798.9 | 671.1 | 798.9 |
| 16000 | 327.4 | 593.9 | 1157.1 | 849.5 | 1082.4 |
| 64000 | 331.8 | 659.5 | 1209.2 | 888.9 | 1182.5 |
| 256000 | 332.0 | 663.2 | 1220.2 | 914.6 | 1201.1 |
| 1024000 | 330.6 | 661.4 | 1226.4 | 899.8 | 1194.7 |
| 4096000 | 307.0 | 592.2 | 1146.1 | 869.3 | 1155.5 |
| 16384000 | 304.2 | 608.6 | 1147.0 | 906.3 | 1152.8 |
| 65536000 | 316.0 | 609.2 | 1144.5 | 1014.1 | 1150.1 |

Interestingly, the performance appears to peak at 4 threads for both the vectorized and non-vectorized cases. This is perhaps due to false-sharing of the cache lines between threads. However, there is about a six times performance gain between the vectorized and non-vectorized experiments for the best case scenario. The optimal performance occurred for array sizes containing about

Table 2: The peak performance calculating the square root of an array with vectorization for various numbers of threads and array sizes. Performance values are given in MegaRoots/Sec.

| Array Size/Threads | 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| 1000 | 1048.6 | 1048.6 | 524.3 | 524.3 | 53.1 |
| 4000 | 1864.1 | 2097.2 | 2097.2 | 2097.2 | 2097.2 |
| 16000 | 2033.6 | 3195.7 | 5592.4 | 4194.3 | 4194.3 |
| 64000 | 2130.4 | 4006.5 | 4971.0 | 7064.1 | 9256.4 |
| 256000 | 2246.3 | 4836.7 | 8521.8 | 7780.7 | 4836.7 |
| 1024000 | 1774.8 | 2774.2 | 3520.5 | 3633.6 | 3848.5 |
| 4096000 | 1082.7 | 1872.9 | 3084.4 | 2854.3 | 2786.2 |
| 16384000 | 1128.4 | 1825.7 | 2261.7 | 2221.3 | 2213.8 |
| 65536000 | 1121.6 | 1782.7 | 2259.9 | 2226.0 | 2225.7 |

256000 entries. Arrays larger than this experienced a performance hit as the execution time began to be dominated by cache line reloads. This trend is more clearly illustrated in Figures 1 and 2. The performance appears to peak for 4 threads for arrays containing 256000 entries, but it is observed that arrays containing 64000 entries exhibit superior performance with increasing number of threads. As mentioned above, the trends may be explained by a combination of false-sharing between the threads (especially as the number of threads becomes larger than 4). Furthermore, the vectorized case is highly sensitive to the array size with performance falling rapidly with larger arrays. This due to violating the cache line's temporal locality policy. Temporal locality is violated because each array entry is only used once during the calculation instead of repeatedly. It is of note that the non-vectorized experiment seems to be relatively insensitive to temporal locality. Since large amounts of data are required for this experiment, further performance for the vectorized case may be gained by prefetching future entries of the array. This would mitigate the time that the processor spends idling waiting for the next set of array entries to be fetched from DRAM and/or lower cache levels on cache misses. Furthermore, ensuring that no cache line contains work assigned to two separate threads would eliminate thread false-sharing that may be occurring. This would give a performance boost since the cache line would need to be reloaded a fewer number of times.

All of this seems to suggest that when vectorizing large computations, paying attention to memory hierarchy is essential. The rate at which the memory supplies the processors with data ultimately determines the maximum performance achievable. Thus, in order for vectorization to be viable, care must be taken to prefetch data and ensure that false-sharing does not occur within the thread team. If these criteria are satisfied, then vectorization will deliver significant performance gains compared to non-vectorized code.

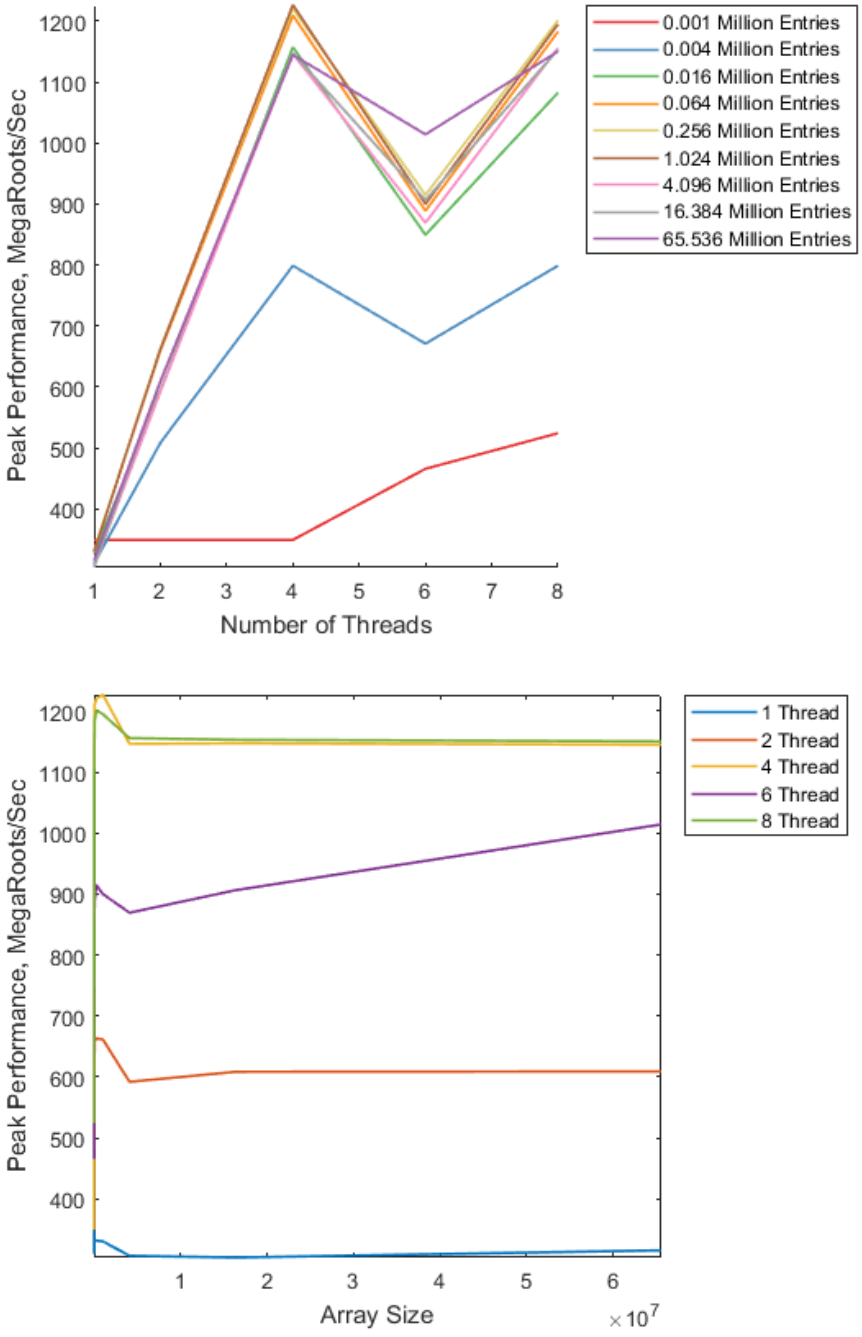


Figure 1: The peak performance of calculating the square root of an array given as a function of number of threads (top) and array size (bottom) for the non-vectorized case. The performance is highly sensitive to false-sharing.

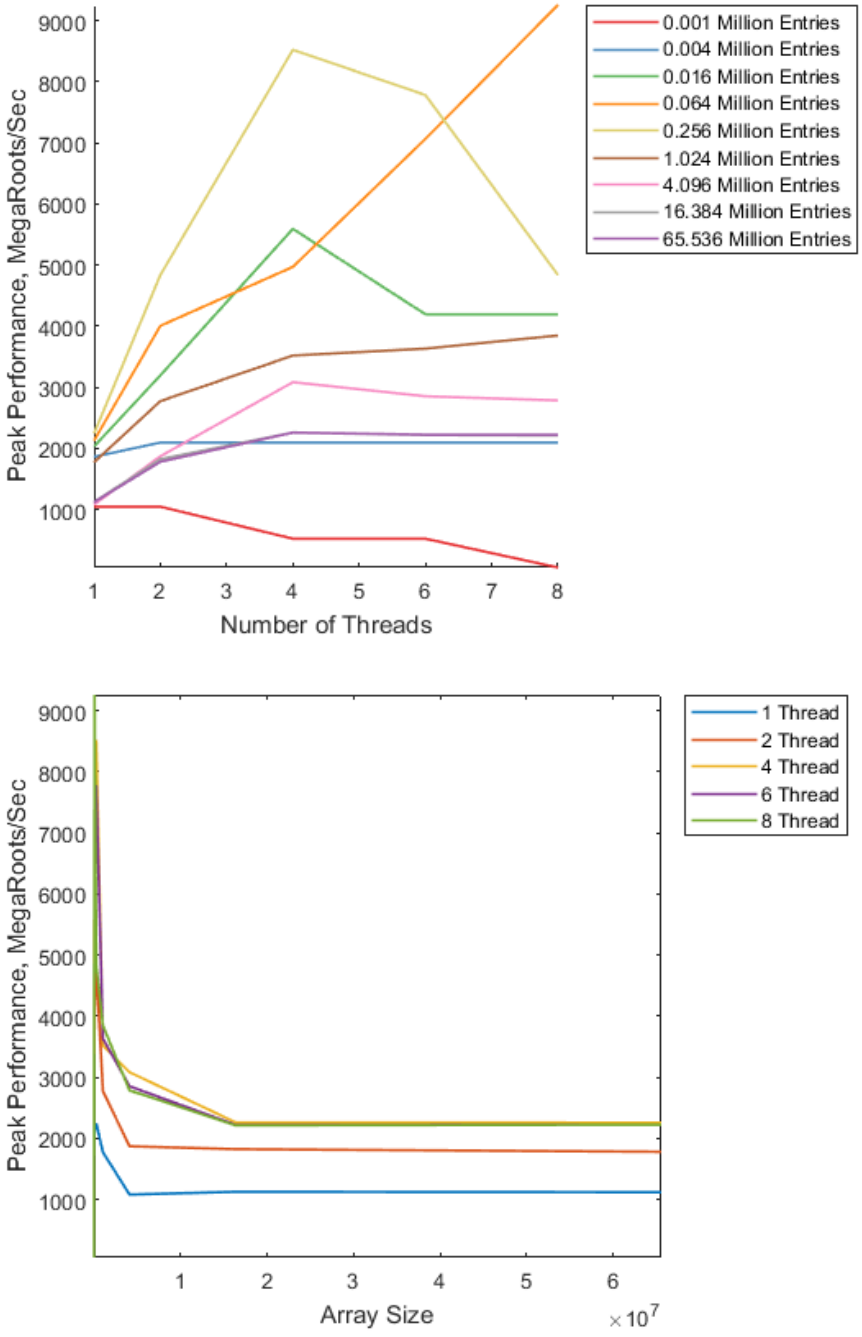


Figure 2: The peak performance of calculating the square root of an array given as a function of number of threads (top) and array size (bottom) for the vectorized case. The performance is highly sensitive to false-sharing and cache temporal locality.