CMSE 822 Homework 3

Tong Li

1) a.

For an intel16 node, the clock speed of the processor is 2.4 GHz. The L1 cache is 32 KB for data and 32 KB for instructions. The L2 cache is 256 KB and L3 cache is 35840 KB.

Assuming the processor is capable of one floating point operation per clock cycle, the theoretical peak performance of the processor is 2.4 GFlop/sec.

b.

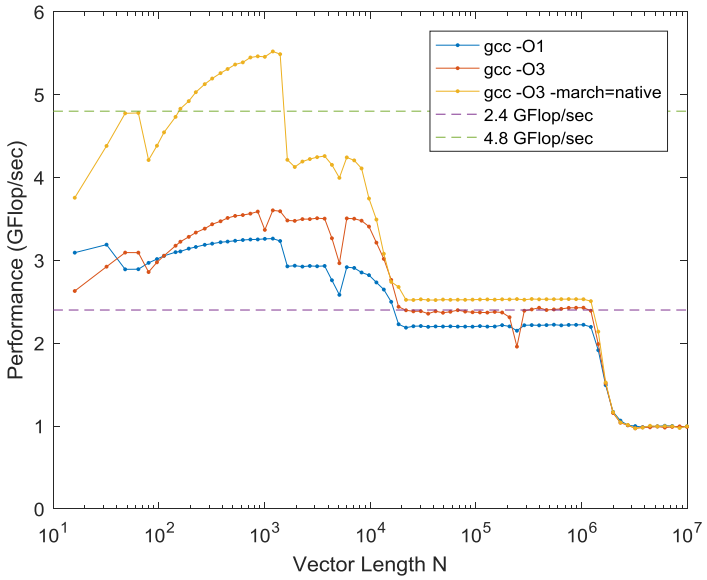
For SSE, the vector size is 128 bits (16 bytes, or 2 double variables). Therefore, the theoretical peak performance with SSE will be 2 *2.4 GFlop/sec = 4.8 GFlop/sec.

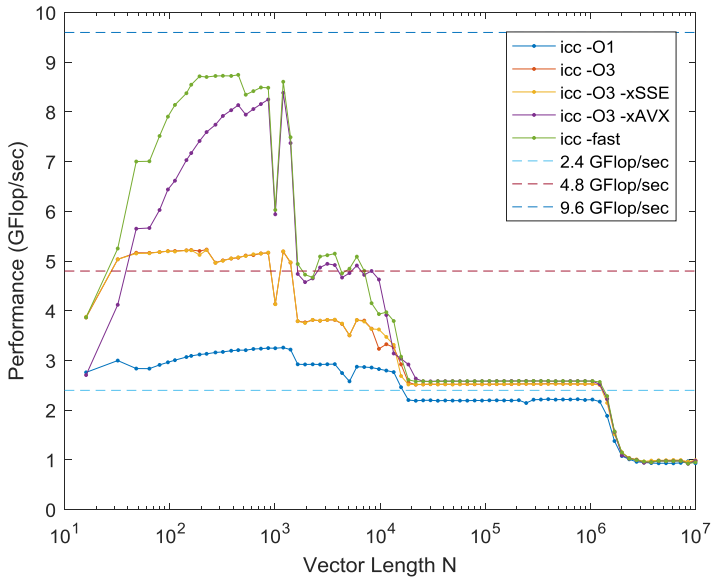
For AVX, the vector size is 256 bits (32 bytes, or 4 double variables). Therefore, the theoretical peak performance with AVX will be 4*2.4 GFlop/sec = 9.6 GFlop/sec.

With FMA, multiple and addition are fused. So only with FMA (without vectorization) the theoretical peak performance will be 2*2.4 GFlop/sec = 4.8 GFlop/sec. With FMA and SSE, the theoretical peak performance will be 2*2*2.4 GFlop/sec = 9.6 GFlop/sec. With FMA and AVX, the theoretical peak performance will be 2*4*2.4 GFlop/sec = 19.2 GFlop/sec.

c.

The figures of performance of the vector triad kernel at different vector lengths N are shown below. GCC compiler (version 4.4.5) is used for the first figure, while Intel compiler (version 13.0.1.117) is used for the second one. The compile options are given in the legend. The theoretical peak performance estimations given in part b are shown by dashed lines on both figures.





d.

Comparing the two figures given in part c, we can see that with similar options Intel compiler optimizes better than GCC compiler. More CPU's features for optimization should be used by Intel compiler.

For GCC, the peak performances of O1 and O3 options are above the theoretical peak performance with no optimization (2.4 GFlop/sec), but below the theoretical estimation for FMA or SSE (4.8 GFlop/sec). Thus, GCC's O1 and O3 options may enable FMA or SSE. The other possibility is that Turbo Boost technology increases the frequency of CPU, making the performance better than 2.4 GFlop/sec. As O3's performance is better than O1's, the most probable explanation is that Turbo Boost makes O1's peak performance better than 2.4 GFlop/sec and that Turbo Boost + "FMA or SSE" makes O3's performance better than O1's.

For GCC's "-O3 -march=native" option, the peak performance is above 4.8 GFlop/sec, but below the theoretical estimations for AVX or FMA+SSE (9.6 GFlop/sec). Thus, "-O3 -march=native" may enable AVX or FMA+SSE. The other possibility is that only FMA or SSE is enabled but Turbo Boost technology makes its performance better than 9.6 GFlop/sec.

Similar analysis can also be done for Intel compiler. The most probable explanation is given in the following table.

Compile option for Intel compiler	Probable explanation for its performance
-O1	Turbo Boost
-O3	Turbo Boost + "FMA or SSE"
-O3 -xSSE	Turbo Boost + SSE
-O3 -xAVX	AVX
-fast	AVX or FMA + SSE (other optimizations may also be implemented)

e.

For small vector length N, the performance does not reach the peak. The main reason should be that for small N instructions which are not floating-point operations (e.g. for loop, call dummy function) take much time.

When N increases to 1000 ~ 2000, there is a sudden drop of performance. At N = 1000 ~ 2000 the memory consumption of the three arrays in vector_triad kernel is about 23 ~ 47 KB, while the L1 data cache is 32 KB. So this drop is caused by the exhaustion of L1 data cache.

When N increases to 10000, there is also a sudden drop of performance. At N =10000 the memory consumption of the three arrays is about 234 KB, while the L2 cache is 256 KB. So this drop is caused by the exhaustion of L2 cache.

When N increases to 10^6 to 2×10^6, there is also a drop of performance. At N = 10^6 to 2×10^6 the memory consumption of the three arrays is about 23 ~ 46 MB, while the L3 cache is 35 MB. So this drop is caused by the exhaustion of L3 cache.

2) a.

The memory bandwidth of function copy computed by STREAM for Intel16 at HPCC is 10955.2 MB/sec (10.698 GB/sec).

Here the array size (STREAM_ARRAY_SIZE) is 20000000. Thus, memory requirement of each array is 153 MB, which is larger than 4 times the size of cache available. Also, each kernel is executed 10 times (NTIMES = 10). In this part, the Makefile provided on STREAM website is used, so the compile command is "gcc -O2 stream.c -o stream_c.exe". The GCC version is 4.4.5 (the same as Problem 1).

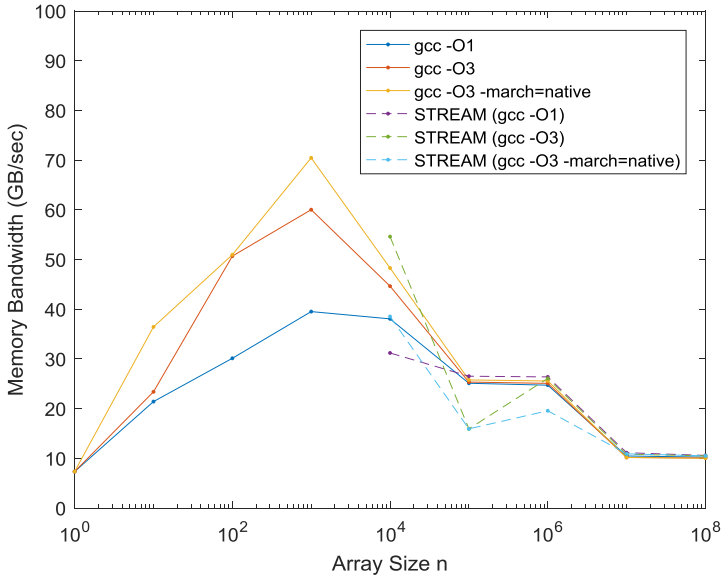
b.

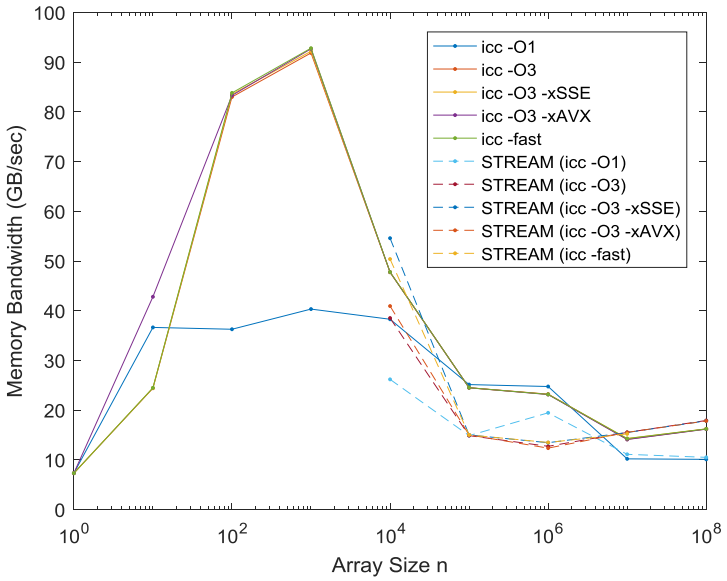
For vector length n = 2000000, the memory bandwidth of my vector copy code is 10.097 GB/sec.

In my vector copy program, function "dummy" which copies C[0] to A[0] is called for each repetition to avoid possible optimization on the repetition loop. To compare with part a, the compile command used for my program is also "gcc -O2". My result is quite close to STREAM's. Cache can impact the bandwidth calculated by my program, and how the compiler optimizes the for loops used in my program can also greatly influence the performance of my program.

c.

The figures of bandwidth calculated by my vector copy kernel at different vector lengths N are given below. The results from STREAM are shown by dashed lines. GCC compiler (version 4.4.5) is used for the first figure, while Intel compiler (version 13.0.1.117) is used for the second one. The compile options are given in the legend. When N < 10000, STREAM gives "inf" (infinite) bandwidth. Thus, the results of STREAM for n < 10000 cannot be given in the two figures.





For n < 10000 my kernel gives finite bandwidths while STREAM gives infinity. It seems that STREAM will give "inf" if the execution time is small, which indicates that cache's impact cannot be neglected for small n. For n ≥ 10000, the results given by my kernel and STREAM are not far away from each other and both decrease as n increases. But for the same n the bandwidths given by my kernel are usually larger than STREAM's.

In addition, comparing the two figures above, we can see that with similar options Intel compiler gives better bandwidths than GCC compiler. So more CPU's features for optimization should be used by Intel compiler. However, when n is large enough different compilers and options give similar memory bandwidths, which indicates that for large n the performance is mainly determined by memory bandwidth (memory bound).

d.

When n is small the memory bandwidth computed by my vector copy kernel is small, because for small n instructions which do not copy (e.g. for loop, call dummy function) take much time.

For n = 10^2 ~ 10^4, the bandwidth computed by my kernel is much higher than it should be. Similar to Problem 1 part 3, when n increases to 10^4 ~ 10^5 there is a sudden drop of bandwidth performance, which should result from the exhaustion of L2 cache. When n increases to 10^6, there is another drop of bandwidth performance, which should result from the exhaustion of L3 cache. So it can be estimated that L2 cache bandwidth is about 70 ~ 90 GB/sec and L3 cache bandwidth is about 20 ~ 30 GB/sec. These estimations should be smaller than the real bandwidth of caches because memory bandwidth still has some impact on the performance when n is small.