Optimization of Matrix Multiplication

Kumudha K.N. (12588), MSc(Engg), CSA

October 7, 2015

1 Optimizations

1.1 Code Optimizations

The following code optimizations were performed to increase the performance of the matrix multiplication code -
1. **Loop interchange:** Incase of a matrix multiplication code, we access the matrix A rowise and the matrix B columnwise. Since the matrices are stored in the row major order, this method of accessing will result in poor performance. Hence, we interchange the j and the k loops.
2. **Tiling:** The next performance bottle neck will be the memory access which are not found in the cache. Hence, we tile the code inorder to improve locality and increase reuse. We tile all the three loops i,j and k.
In order to find the optimal tile size for each of the loops, we ran the code starting from tile size of 16 upto 2048, with all the different combinations for i,j and k and found the optimal tile size to be $128 * 8 * 128$.
3. **Auto-Vectorization:** Atomatic vectorizer, is a component of icc which tries to automatically use SIMD.
4. **Parallelization:** The code for matrix multiplication has inherent data parallelism that can be used. We parallelise the code on the outer most loop (i loop), so that we perform maximum number of operations before synchronization.
We used OpenMP in order to parallelize the code.

1.2 Compiler flags

The following compiler Flags were used -

1. -DTIME : To print the time taken by matrix multiplication
2. -O3 : Used when loops use floating point calculations
3. -openmp : Enables the parallelizer to generate multi-threaded code
4. -o : to specify the output filename

5. -fp-model precise : Enables value-safe optimizations on floating-point data
6. -xhost : used if CPU supports only AVX
7. -ipo : Enable interprocedural optimization
8. -axCORE_AVX2: Used since the Xeon processor supports Advanced Vector Extensions 2, uses AVX if not present
9. -fma : when specified, the compiler may generate FMA instructions for combining multiply and add operations where applicable

Command used : $icc\ -DTIME\ -O3\ -openmp\ -o\ mout\ mmm-opt.c\ -fp-model\ precise\ -xhost\ -ipo\ -axCORE\_AVX2\ -fma$

2 Performance

We performed the optimizations as specified in Section 1, and got the execution time of matrix multiplication to 2.966402 seconds for and array size of $M = N = K = 4096$ and Number of cores used is 4. Whereas, the original code took 857.990507 seconds.

2.1 Processor Details

Performance was measured with a system with the below configuration -

Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz

32KB L1 iCache, 32KB L2 dCache private

256KB L2 private

20MB L3 shared

2.2 Evaluation

Table 1: Performance on Increased problem size

Optimization	Execution time (in seconds)
No optimization	857.990507
Loop Interchange	50.064958
Loop Interchange + tiling(128*8*128)	11.390123
Loop Interchange + tiling + Parallelization(4 threads)	2.966402

Further more analysis was performed on the optimized code. The results are presented in the next section.

2.3 Speedup

The baseline we consider is the original matriox multiplication code, with no optimizations.
1. Speed up w.r.t vectorization on optimized code :

no-vectorization - 2.966402 s vectorization - 10.955006 s

speedup - $3.69X$

2. Speed up w.r.t parallelism is shown in Table 2. Base reference is the optimized code without parallelism. We can see a speedup of $3.83X$ when we use 4 threads (4 cores).

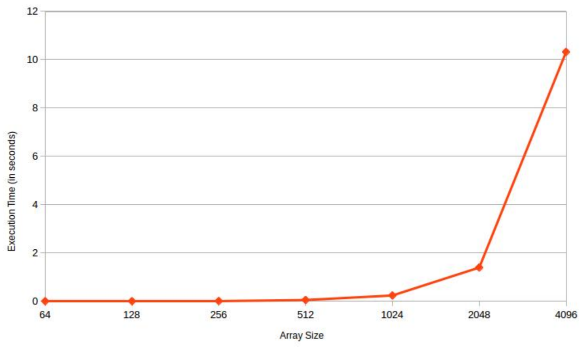
2.4 Increased Problem Size(Weak Scaling)

We maintained the number of cores as 1 and increased the problem size starting from $M, N, K = 64$ to $M, N, K = 4096$ in powers of 2.
The execution time measured is shown in $Table1$, and the performance graph is shown in $Figure1$

Table 2: Performance on Increased problem size

Array Size	Execution time (in seconds)
64	0.00009
128	0.000735
256	0.006393
512	0.049869
1024	0.238103
2048	1.395238
4096	10.322358

Figure 1: Matrix Multiplication (Sequential)



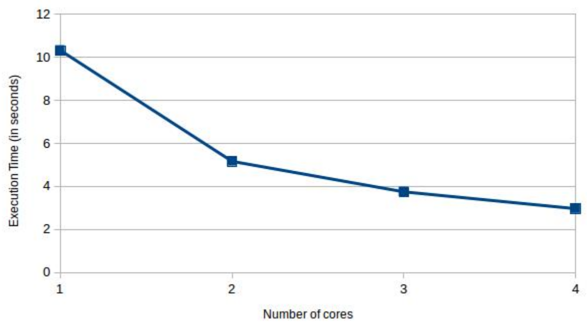
2.5 Execution on Multiple cores(Strong Scaling)

We used the problem size $M = N = K = 4096$ and measured the performance on $1, 2, 3$ and 4 cores. The execution time for matrix multiplication is shown in $Table2$ and the graph is shown in $Figure2$.

Table 3: Performance on Multiple cores

Number of Cores	Execution time (in seconds)	Speedup
1	10.322358	1.1X
2	5.166183	2.2X
3	3.748585	3.03X
4	2.966402	3.83X

Figure 2: Matrix Multiplication (Multiple Cores)



3 References

1. ICC Optimization Options - https://software.intel.com/en-us/articles/step-by-step-optimizing-with-intel-c-compiler