Homework 1. Matrix Multiplication Report for Stage 2

Yuan Huang(yh638) & Enrique Rojas Villalba(elr96) & Ravi Patel (rgp62)

1. Summary of the stage 1

In stage 1, we tried using the following methods:

1) Blocking
2) Changing the order of the loops
3) Copying the data to a separate storage space.
4) Only using blocked multiplication for large original matrices
5) Restricted pointers

Among the methods, just adding restrict pointers to the basic implementation can improve performance to the f2c baseline, while blocking and copying can beat f2c for large matrices.

For stage 2, we tried several methods to further improve performance.

2. Transpose Matrix when copying

Arrays in C are row-major but when we perform a matrix multiplication, we traverse the matrix A column first. Transposing A enables us to traverse it row first, so our code can make better use of the cache and allow for the compiler to vectorize the calculations more easily.

3. Adding other hints for compiler

As adding restrict worked well in stage 1, we tried to add other hints such as the " __assume_aligned" and "#pragma vector always" to help the compiler to find opportunities for optimization. These keywords, however, did not improve performance.

4. Adjusting the block size

After changing the way of copying, we tried again to determine the most effective block size. We find that BLOCK_SIZE=32 gives the best results in our experiments. This is because the size of L1 cache of the Xeon Phi is 32KB and using BLOCK_SIZE=32 results in 2*32*32*8 Bytes=16KB of cache usage. This

maximizes L1 cache usage, resulting in better performance.

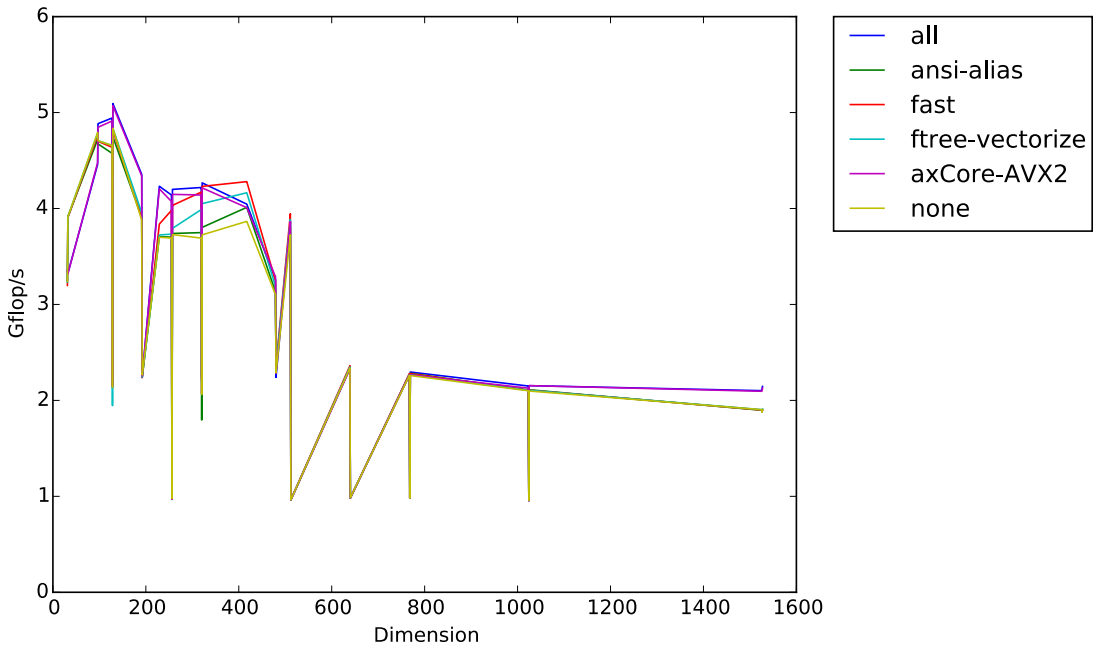
5. Vectorization

We also tried to use SSE vector instructions in xmmintrin.h in our code dgemm_mine_vec1.c. However, this reduced performance. The vector instructions require additional checks on the boundary cases resulting in slower performance.

We generated a n=5 vectorization report with the Intel compiler. From the report, the compiler successfully vectorized several sections of the code. Speedups up to 8 were reported.

6. Flags

To analyze the effects of certain flags, we compared the performance of -axCORE-AVX2 -ftree-vectorize -fast and –ansi-alias on the basic implementation.

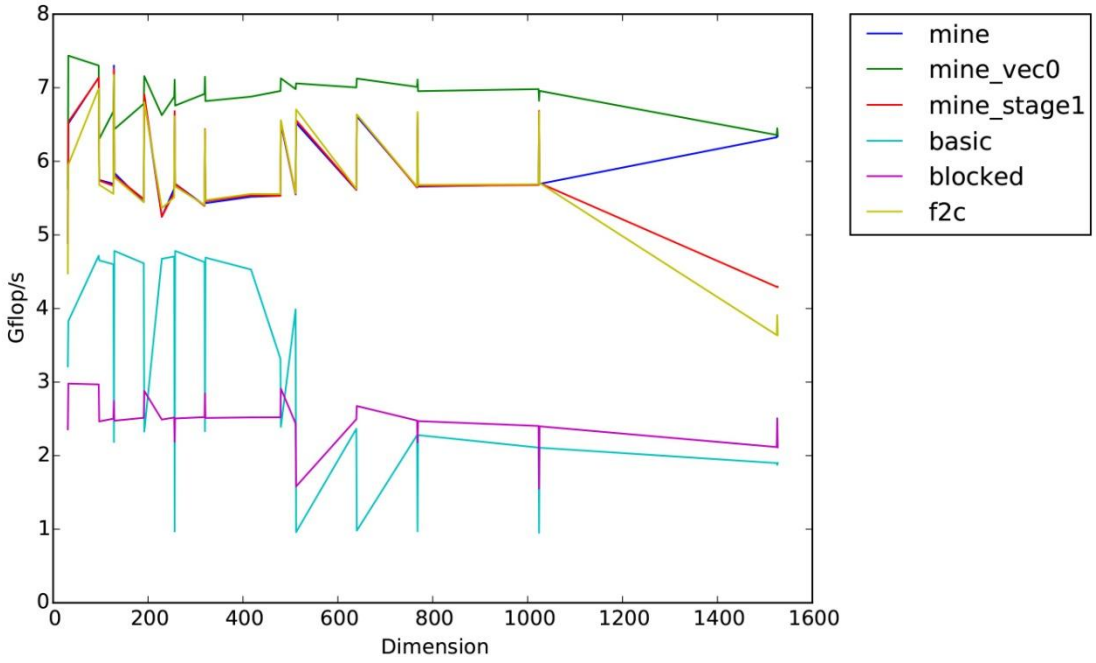


We found that the flags above, especially –fast and axCore-AVX2, offer a modest improvement in performance, especially for small and large sized matrices.

7. Result

1) Without flags

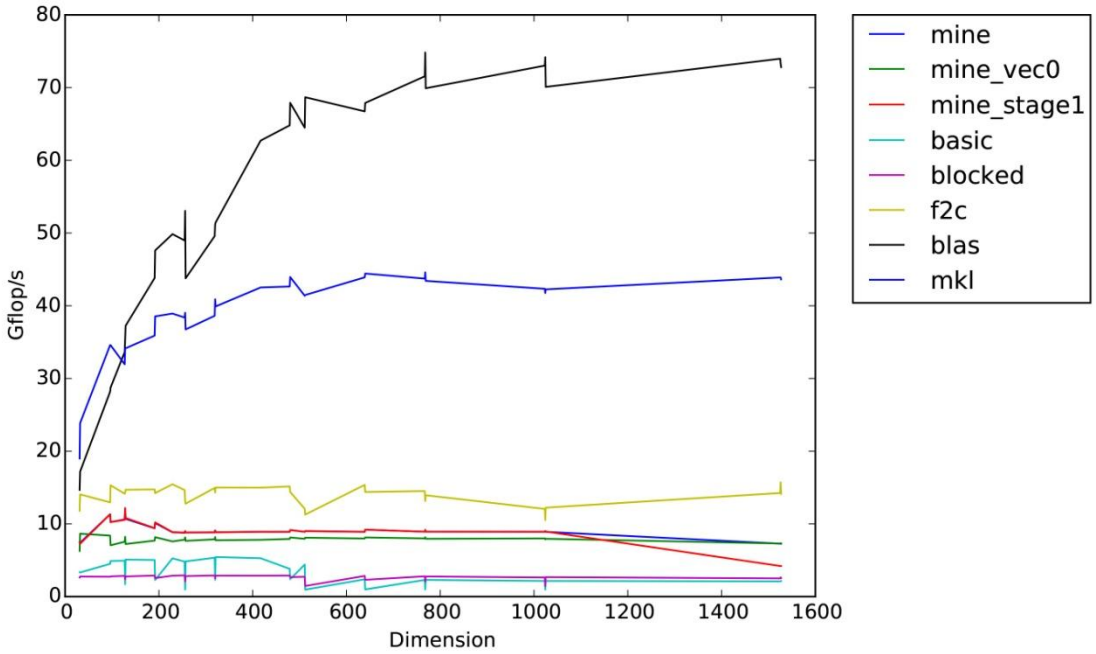
The following result is the result we get before using flags:

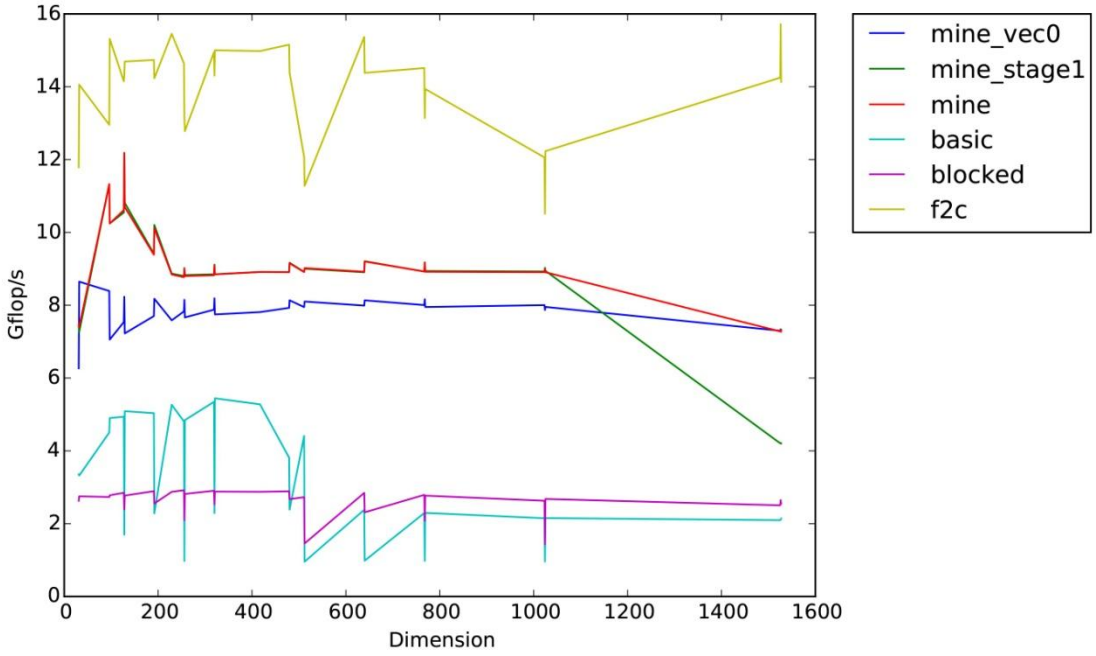


mine_stage1 is our previously submitted code. mine_vec0 code uses blocking and transposed copying. mine uses blocking for matrices smaller than START_BLOCK_SIZE (set to 1200, the same as the number we used in stage 1) and use naive multiplication with restricted pointers.

We can see that our code mine_vec0 using the transpose of matrix A can do much better than other simple baselines (excluding the blas and mkl baseline). So the blocking, copying and transposing performs well even for small inputs.

2) With flags





We compared the effect of the flags on the different implementations. We used -axCORE-AVX2 -ftree-vectorize –fast and –ansi-alias for this analysis. The simple implementations did not see much improvement with the flags. However, our implementations received a substantial boast with the flags. Interestingly, the flags gave the largest gains to the Fortran code such that it surpassed our implementations. The Intel compilers appear to be better at optimizing the Fortran code compared to the

C code.

8. Conclusions

After trying several combinations, we found that not all the modifications produce a significant improvement. At the end, the best results were obtained by changing block sizes, copying, transposing and using restrict on the pointers.

Blocking, copying and transposing were found to be very important in maintaining good performance for big matrix sizes (>1200), although blocking was found to degrade performance with smaller matrices.

9. References

"The GotoBLAS/BLIS Approach to Optimizing Matrix-Matrix Multiplication - Step-by-Step." *HowToOptimizeGemm*. Web. 01 Oct. 2015. <http://wiki.cs.utexas.edu/rvdg/HowToOptimizeGemm/>.

Bindel, David. "Tuning on a Single Core." Web. <http://www.cs.cornell.edu/~bindel/class/cs5220-f11/notes/serial-tuning.pdf>.

"Intel® Xeon PhiTM Coprocessor Architecture Overview." *PRACE MIC Summer School* (2013): Web. <http://www.training.prace-ri.eu/uploads/tx_pracetmo/MIC_Intro_Architecture.pdf>.