**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. Not using blocked multiplication when the size of the original matrix is small
5. Restricted pointers

Among the methods, adding restrict pointers can do as well as the f2c baseline and the blocking and copying can beat the f2c baseline when the size of input is large.

We are trying to do the stage 2 on the blocking and copying with the restrict pointers.

1. **Transpose on Matrix when copying**

The array in C is row based but when we make a matrix multiplication, we need to visit one of the matrixes through column. So when we are copying the matrix, we do a transpose on matrix A. This enables us to go through a line when we do the multiplication so that we can make better use of the cache and make the code easier for the compiler to vectorize the calculations.

1. **Adding other hints for compiler**

As adding restrict works 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. But it doesn’t help us to improve the performance..

1. **Adjusting the block size**

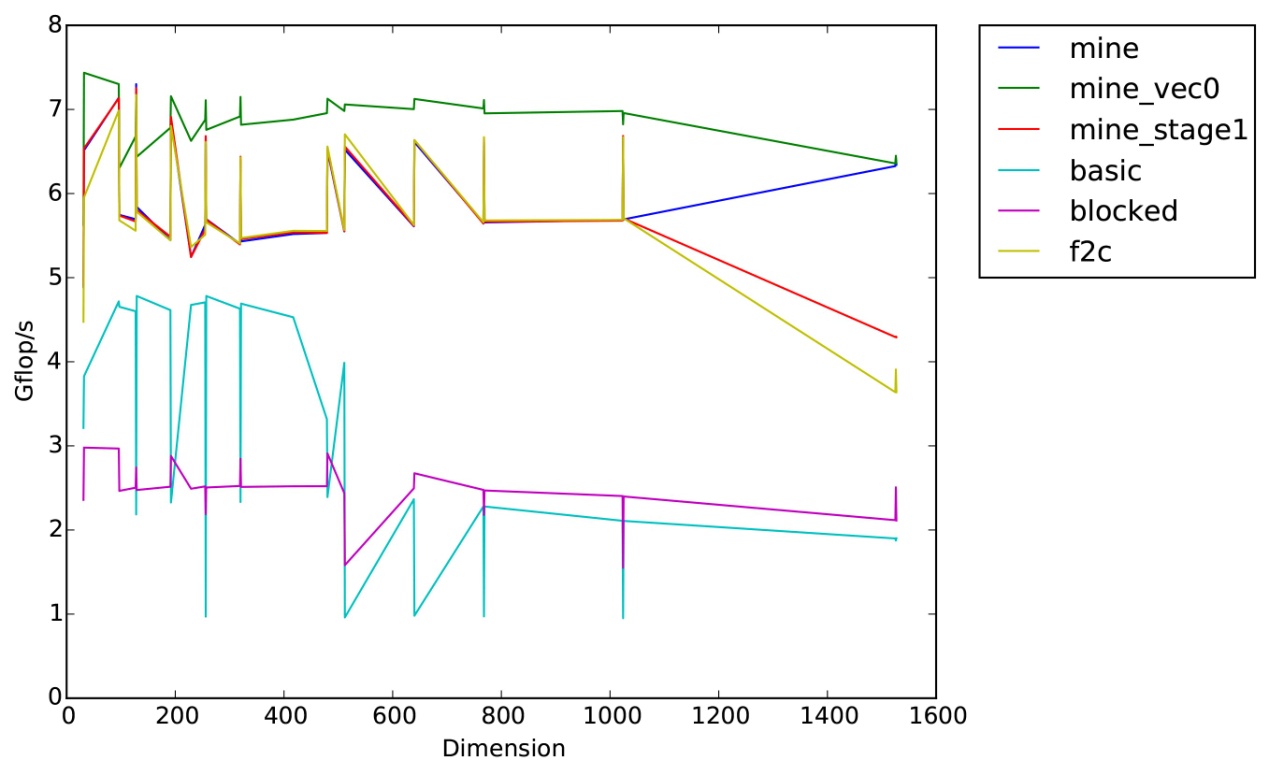
After changing the way of copying, we tried again to determine the best size of blocking. We find that using BLOCK\_SIZE=32 is the best in our experiment. That’s because the size of L1 cache of the Xeon Phi is 32K and the BLOCK\_SIZE=32 takes 2\*32\*32\*8 Bytes=16KB. So it can just fit into the L1 cache and this makes the performance better.

1. **Using Vector Instructions in xmmintrin.h**

We also tried to use vector instructions in xmmintrin.h in our code dgemm\_mine\_vec1.c. But this makes the performance go down. I think that’s because when using the vector instructions, we have to do some checks on the boundary situation and this makes the code goes slow.

1. **flags**
2. **Result**
3. without using flags

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



The mine\_vec0 code is the code that always uses blocking and transposed copying.

The “mine” code is the one which starts do blocking after a START\_BLOCK\_SIZE which is 1200(the same as the number we used in stage\_1) and use naive multiplication with restrict pointers.

We can see that our code mine\_vec0 using the transpose of matrix A can do much better than other simple baselines (not containing the blas and mkl baseline). So the blocking, copying and transposing performs well even in the case that the size of input is small.

1. using compiling flags

