**Homework 1. Matrix Multiplication**

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

**Report for Stage 1**

**•Optimizations used or attempted**

We tried the first three methods mentioned in the "Tuning on a single core" notes, namely blocking with different sizes, changing the order of the loops and copying blocks into separate storage.

Also for small and not divisable by 64 matrix sizes, we use the basic multiplication methods because the blocked methods seems not have good performance on those sizes.

Also, we add restrict to the pointers.

**•Reasons for using the aforementioned Optimizations**

1) Blocking:

Blocking can enable us to work on smaller parts of the matrix, which can make better use of the cache. Since different size of blocks will change the behavior of the cache, we tried several sizes of blocks. Also, given that the size of L2 cache of the processor is 256KB, we should try the block sizes around 104. Also, we should avoid choosing number that is power of 2, which may cause conflict missing on cache. Currently, we find that choosing 80 as the block size gives as good result.

2) Changing the order of the loops

The changing of the loop order may affect the efficiency of the visit to the cache by changing the order of bytes in the memory. So we can have a lower cache missing rate.

3) Copying the data to a separate storage space.

This can let us work on smaller parts of the matrix so that we can try to keep the whole block inside the cache. Also we can avoid some of the conflict missing of cache when the original size of matrix is some multiple of a power of 2.

4) Not using blocked multiplication when the size of the original matrix is small.

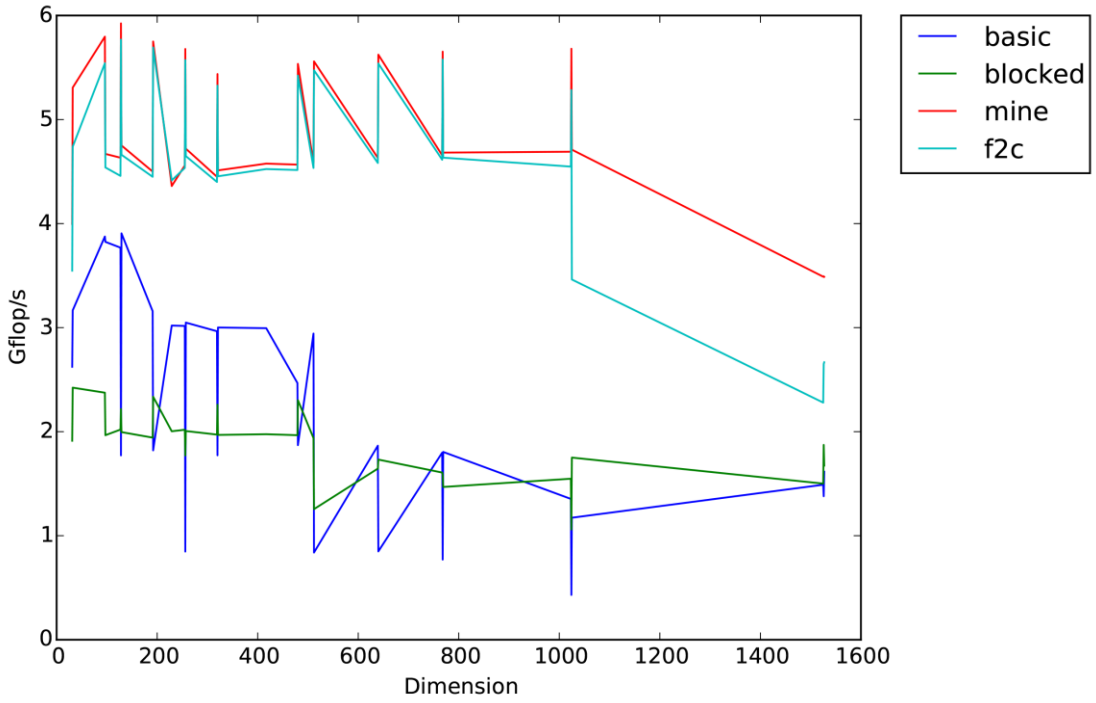
When the size of the original matrix is small, blocking and copying doesn't save so much time. So blocking is only used for matrices larger than n=1200.

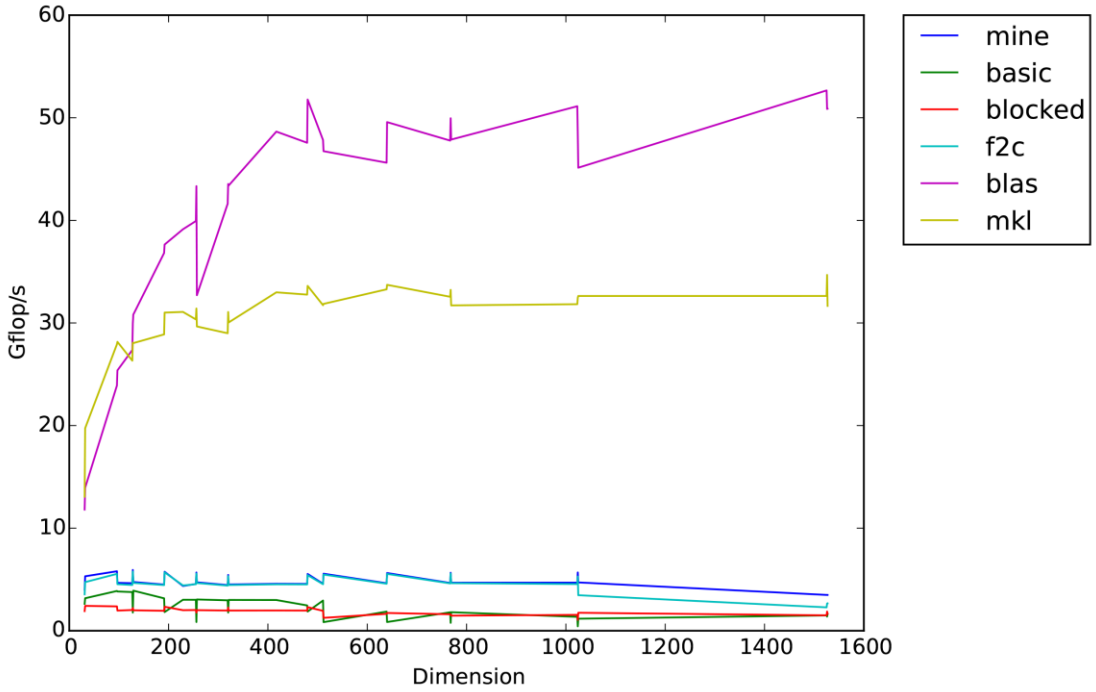
5) Restricted pointers

Adding restrict to pointers promises the compiler there will be no aliasing between different pointers. Thus, the compiler does not need to check for aliased pointers, resulting in faster code.

•Results

The following graphs show the performance of our code in comparison to the other approaches.






From the results, our code beats the f2c implementation. We have reached the performance of f2c code for small sizes, and did better for larger sizes. Also we are much better than the original blocked and basic version of the code.

•What didn't and did work

Loop ordering didn’t work so far. Restricting the pointers was the most critical change for increasing the performance on small size because it reduced the amount of checks for aliasing so the code ran faster. Blocking and copying helps improve the performance at larger size.