Matrix Multiplication

CS 5220: Homework 1

Group 9

Ze Jin (zj58) Sam Tung (sat83) Patrick Cao (pxc2)

1 Method

We refer to the tuning ideas on the note "Tuning on a single core".

1.1 Loop Order

In the naive implementation, we loop over i, then j, then k.

Alternative loop order might be more cache friendly. For example, using the (i, j, k) order, we have to go across a row of A in the inner loop, which is a non-unit-stride access.

We compare different orders, including $(i, k, j), (j, i, k), (j, k, i)$.

1.2 Blocking

We can partition matrices into small blocks, and apply the block matrix multiplication.

The advantage is that original matrices cannot fit into cache while the little blocks can.

We compare different block sizes, including 2, 4, 8, 16, 32, 64, 128.

1.3 Copy Optimization

We might run into conflict misses associated with cache. One way to solve the problem is to explicitly copy blocks into a contiguous block of local storage before multiplying them.
Another side benefit of copy optimization is that we can use it to gracefully deal with fringe blocks.
Therefore, we add copy optimization to one-layer and two-layer blocking.
In one-layer blocking with copy optimization, we compare different block sizes, including 2, 4, 8, 16, 32, 64, 128.
In two-layer blocking with copy optimization, we compare different block size pairs, including (64,2), (64,4), (64,8), (64,16), (64,32), (128, 64), (256,64), (512,64).

1.4 Compiler Flag

It is worthwhile playing with the flags that control the compiler optimizations, since modern compilers do some types of optimizations much better than people do.
We compile the code with option `-funroll-loops` to unroll loops (basic loop unrolling is automatic with -O3).

2 Result

We run jobs on the totient cluster, and get the results depicted in several plots.

2.1 Loop Order

See Figure 1, 2, 3 and 4 below for loop order.
The plot shows that looping performs the best with loop order (j, k, i).

2.2 Blocking

See Figure 5 and 6 below for one-layer blocking.

The plot shows that one-layer blocking performs the best with block size $= 64, 128, 256$.

2.3 Blocking and Copy Optimization

See Figure 7 and 8 below for one-layer blocking with copy optimization.

The plot shows that one-layer blocking with copy optimization performs the best with block size $= 32, 64$.

See Figure 9 and 10 below for two-layer blocking with copy optimization.

The plot shows that two-layer blocking with copy optimization performs the best with block size pair $= (64, 32), (128, 64), (256, 64), (512, 64)$.

2.4 Compiler Flag

See Figure 11 below for unrolling loops in compiler flags.

The plot shows that unrolling loops in compiler flags alone does not make much difference.

Reference

(1) David Bindel, Tuning on a single core, Applications of Parallel Computers (CS 5220), Fall 2011.

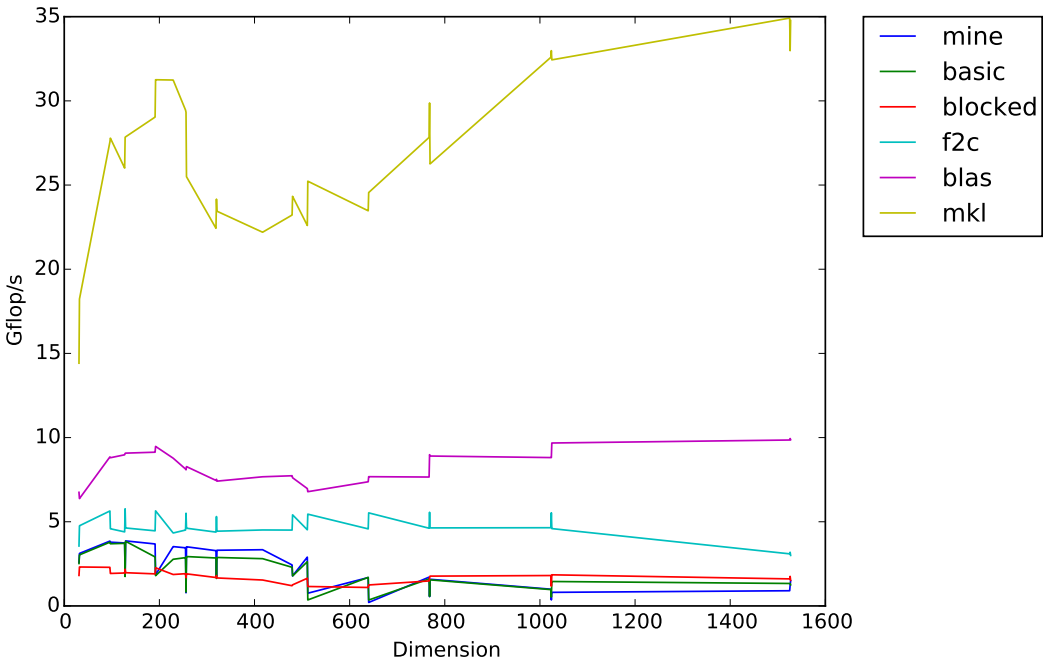

Figure 1: loop order: (i, j, k)

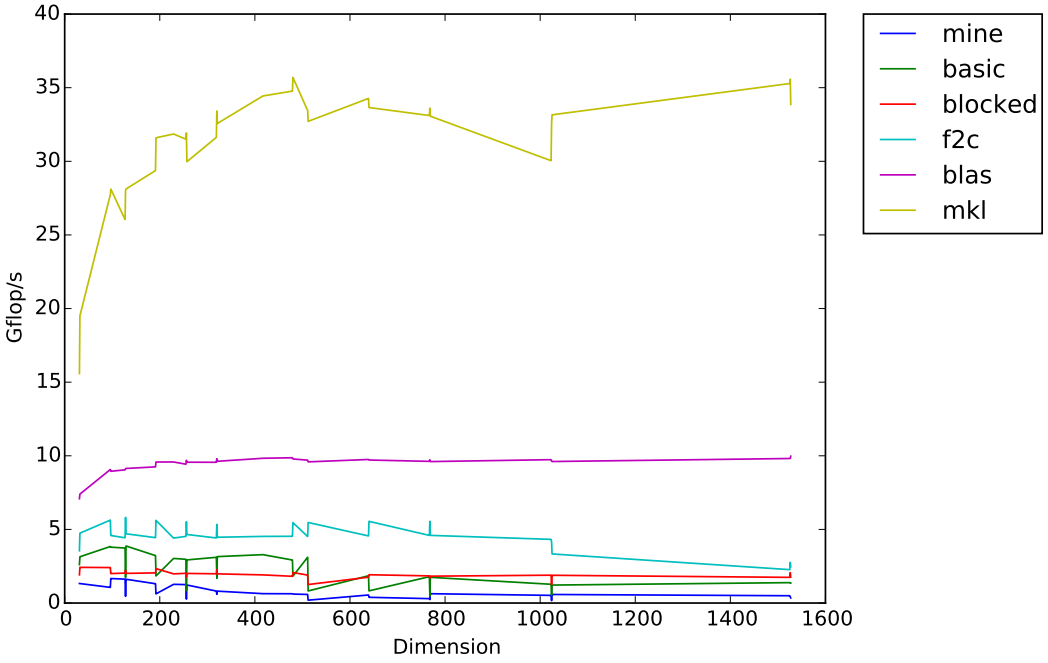

Figure 2: loop order: (i, k, j)

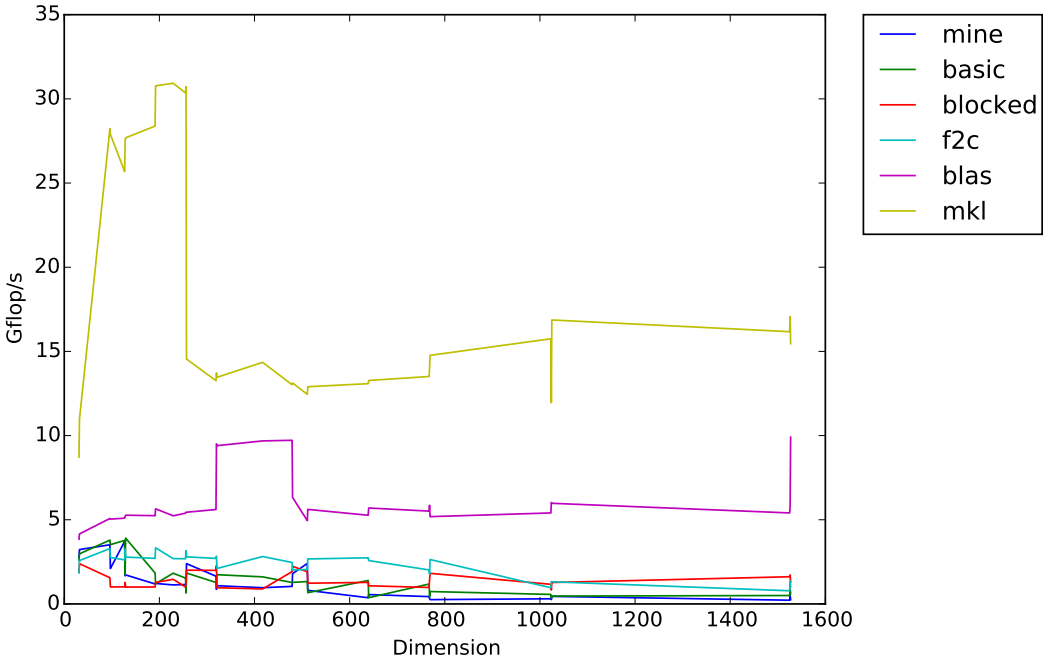

Figure 3: loop order: (j, i, k)

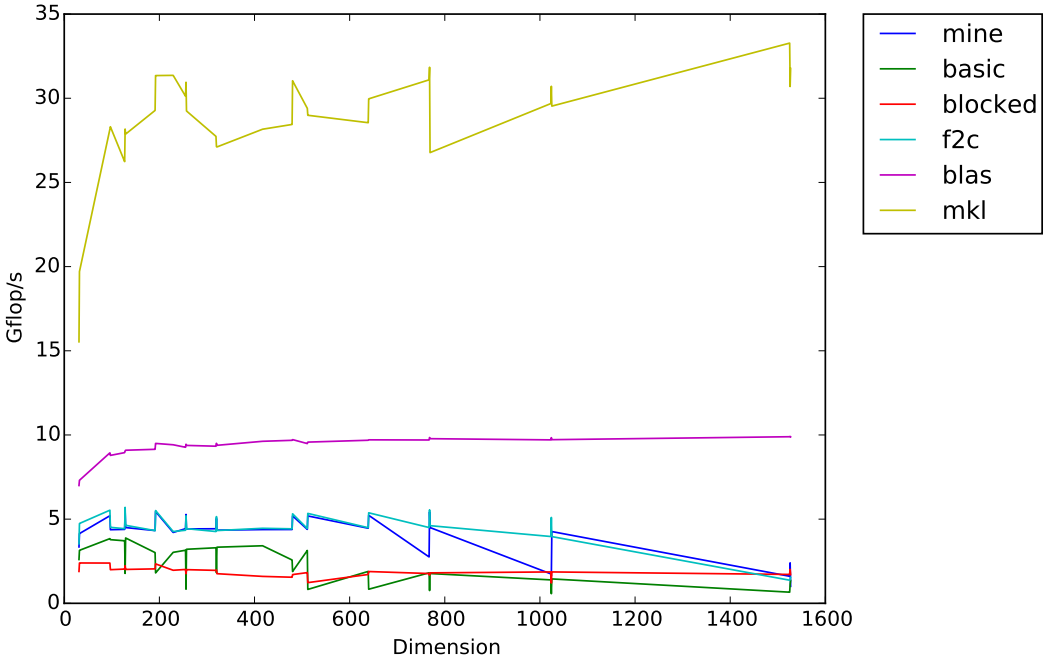

Figure 4: loop order: (j, k, i)

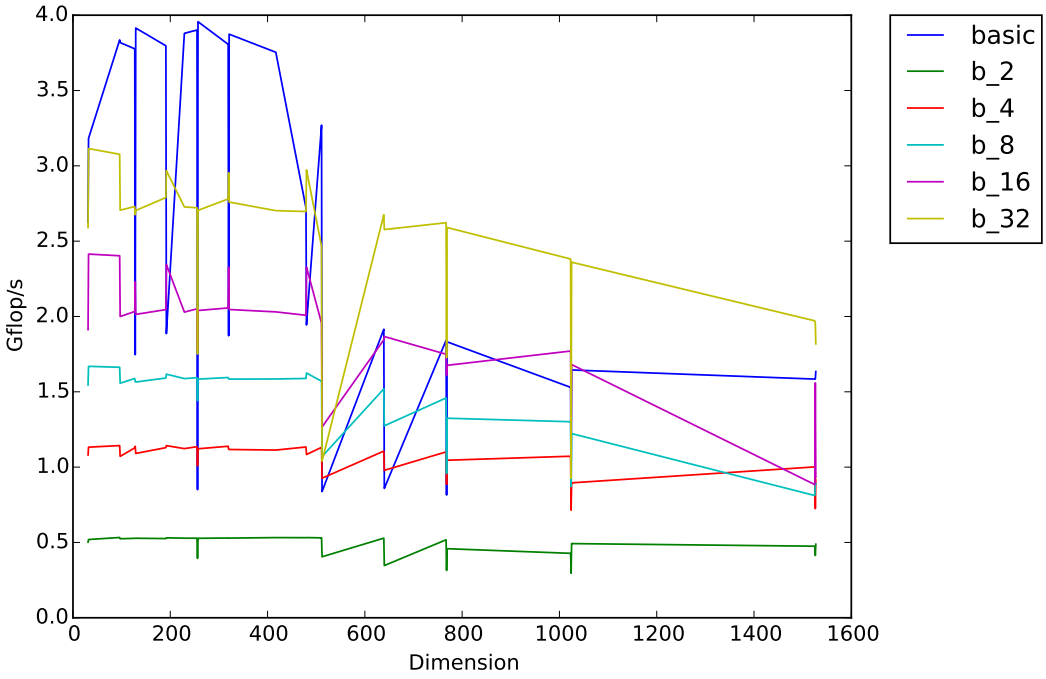

Figure 5: one layer of blocking

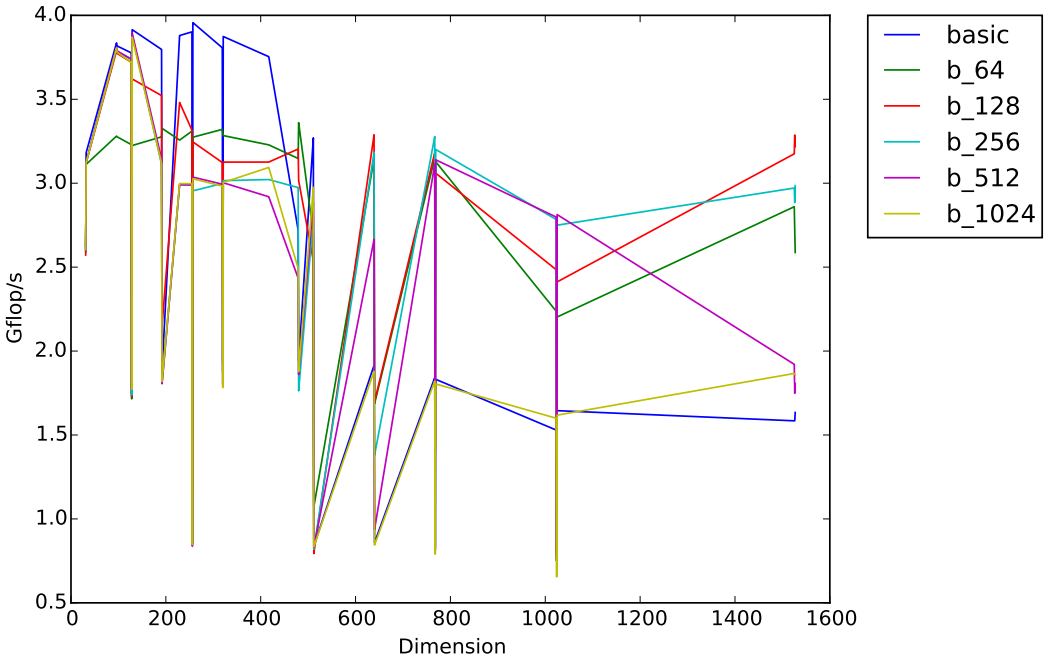

Figure 6: one layer of blocking

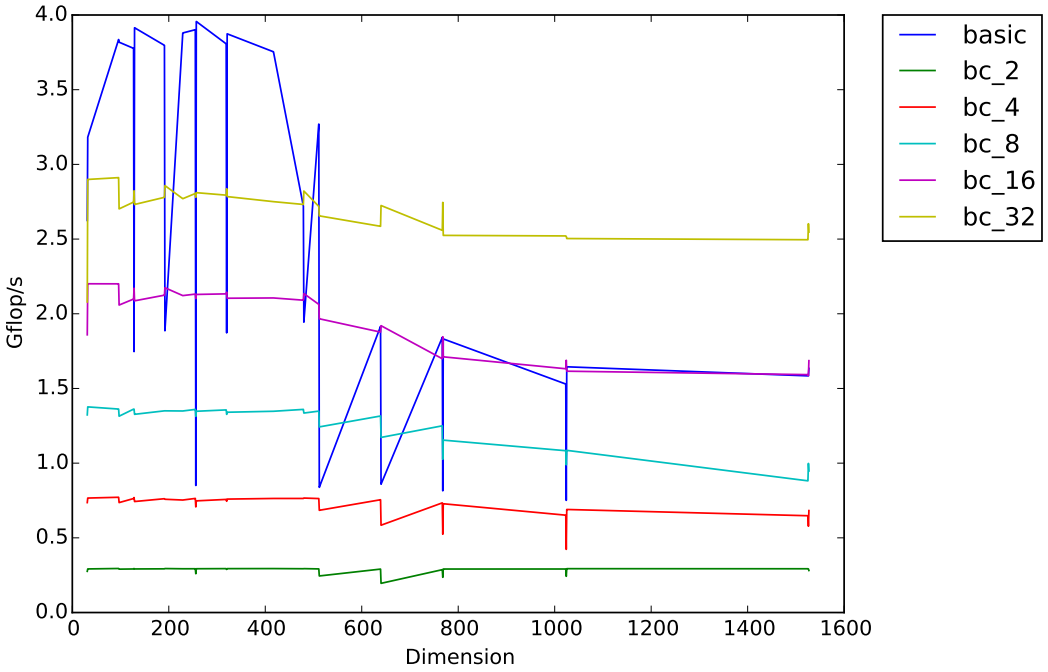

Figure 7: one layer of blocking with copy optimization

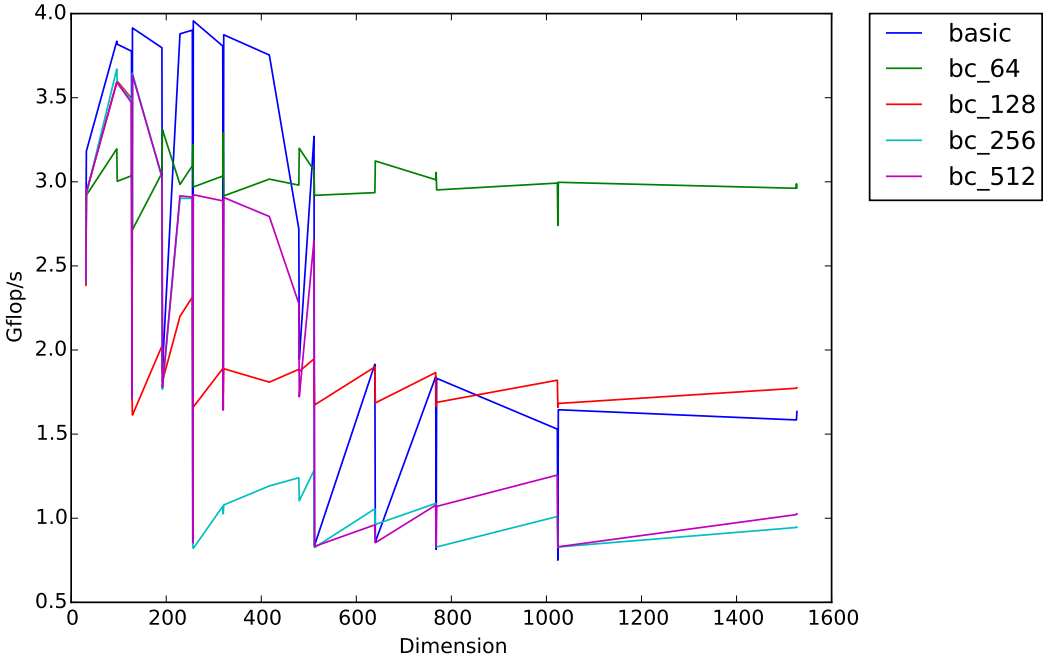

Figure 8: one layer of blocking with copy optimization

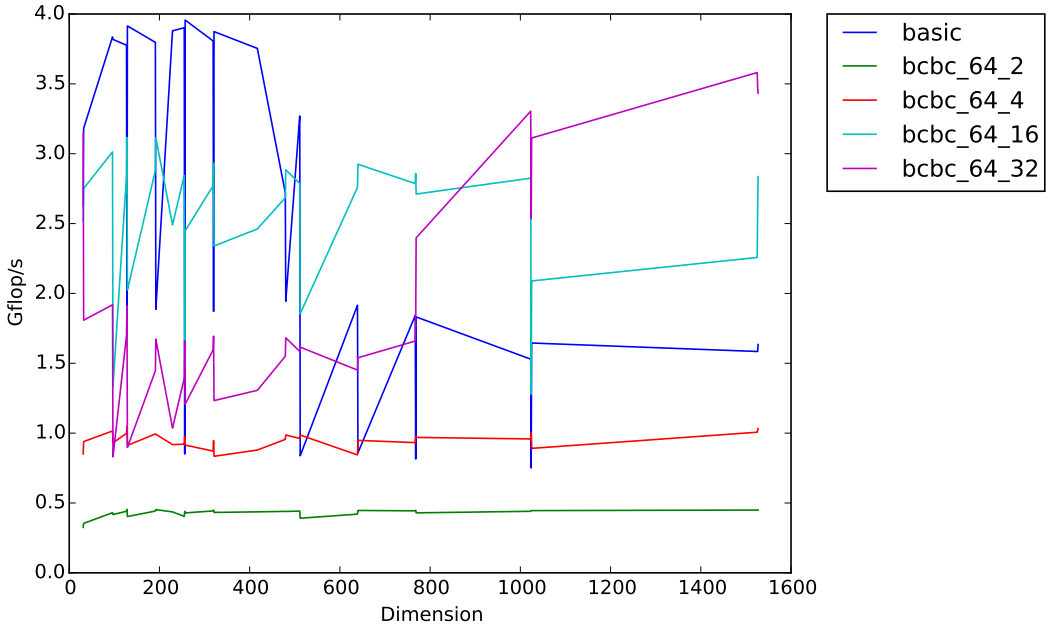

Figure 9: two layers of blocking with copy optimization

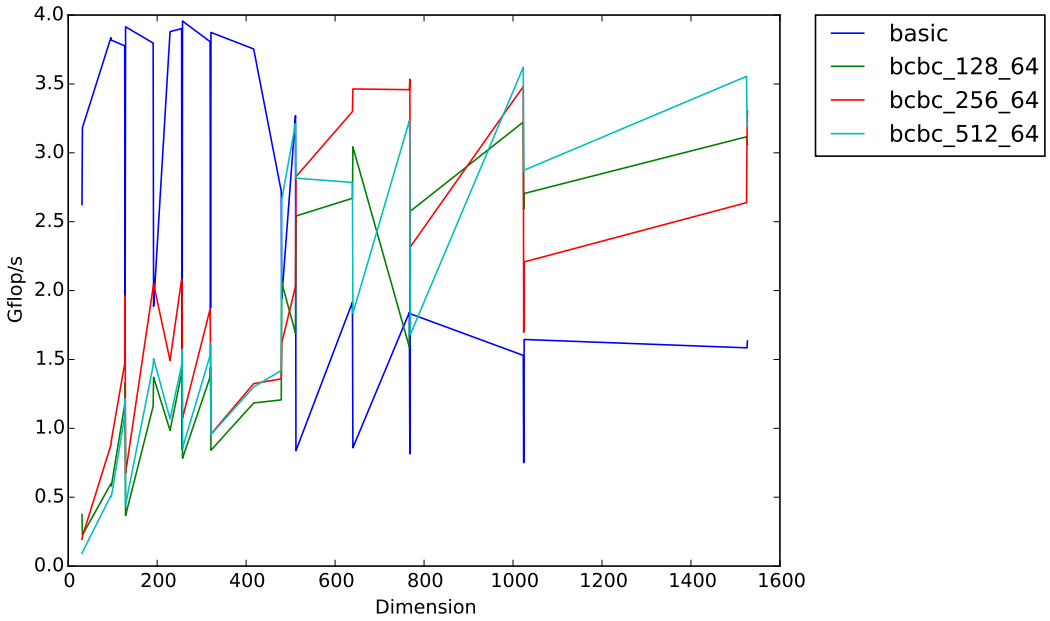

Figure 10: two layers of blocking with copy optimization

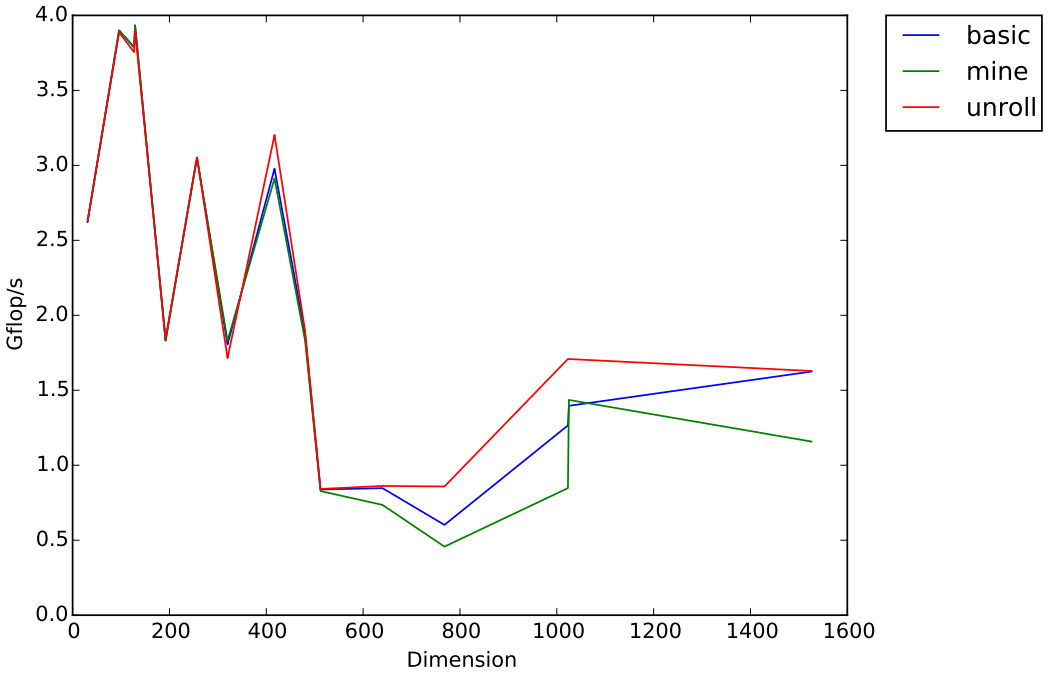

Figure 11: unrolling loops in compiler flags