CS540 Assignment 2

Dustin Ingram

November 14, 2011

1 Introduction

For this assignment, various methods of mini- matrix multiplication are compared, including matrix blocking and loop unrolling. The results are compared based on the performance to determine the optimal block size for a particular system.

2 Summary of Results

The results can be organized as follows:

2.1 Blocking, Unrolling, and Scalar Replacement

It was found that blocking the matrix, unrolling the innermost loop, and performing scalar replacement and addition/subtraction interleaving greatly improved performance.

2.2 Loop Ordering

It was found that differences in loop ordering produced results that were essentially the same, thereby offering no improvement in performance.

2.3 Matrix Size Comparisons

As expected, it was found that the best implementation with the highest-performing values for block size performed better than the naïve code.

3 Description of Computing Platform

All tests were run on `float.cs.drexel.edu`. Relevant system architecture information follows.

3.1 System & Kernel Information

```
$ uname -a
Linux float.cs.drexel.edu 2.6.35-28-generic #50-Ubuntu SMP Fri Mar 18 18:42:20
    UTC 2011 x86_64 GNU/Linux
```

3.2 GCC Version Information

```
$ gcc --version
gcc (Ubuntu/Linaro 4.4.4-14ubuntu5) 4.4.5
```

3.3 CPU Information

```
$ cat /proc/cpuinfo
processor       : 15
vendor_id       : GenuineIntel
cpu family      : 6
model           : 44
model name      : Intel(R) Xeon(R) CPU           L5630  @ 2.13GHz
stepping        : 2
cpu MHz         : 1600.000
cache size      : 12288 KB
physical id     : 1
```

```
siblings        : 8
core id         : 10
cpu cores       : 4
apicid          : 53
initial apicid  : 53
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
    cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
    pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good xtopology
    nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2
    ssse3 cx16 xtpr pdcm dca sse4_1 sse4_2 popcnt aes lahf_lm ida arat dts
    tpr_shadow vnmi flexpriority ept vpid
bogomips        : 4266.84
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

3.4 Memory Information

```
$ papi_mem_info
Memory Cache and TLB Hierarchy Information.
------------------------------------------------------------------------
TLB Information.
  There may be multiple descriptors for each level of TLB
  if multiple page sizes are supported.

L1 Instruction TLB:
  Page Size:            2048 KB
  Number of Entries:       7
  Associativity:        Full

L1 Instruction TLB:
  Page Size:            4096 KB
  Number of Entries:       7
  Associativity:        Full

L1 Data TLB:
  Page Size:               4 KB
  Number of Entries:      64
  Associativity:           4

L1 Data TLB:
  Page Size:            2048 KB
  Number of Entries:      32
  Associativity:           4

L1 Data TLB:
  Page Size:            4096 KB
```

```
  Number of Entries:      32
  Associativity:           4

L1 Instruction TLB:
  Page Size:               4 KB
  Number of Entries:      64
  Associativity:           4


Cache Information.

L1 Data Cache:
  Total size:             32 KB
  Line size:              64 B
  Number of Lines:       512
  Associativity:           8

L1 Instruction Cache:
  Total size:             32 KB
  Line size:              64 B
  Number of Lines:       512
  Associativity:           4

L2 Unified Cache:
  Total size:            256 KB
  Line size:              64 B
  Number of Lines:      4096
  Associativity:           8

L3 Unified Cache:
  Total size:          12288 KB
  Line size:              64 B
  Number of Lines:    196608
  Associativity:          16
```

mem_info.c PASSED

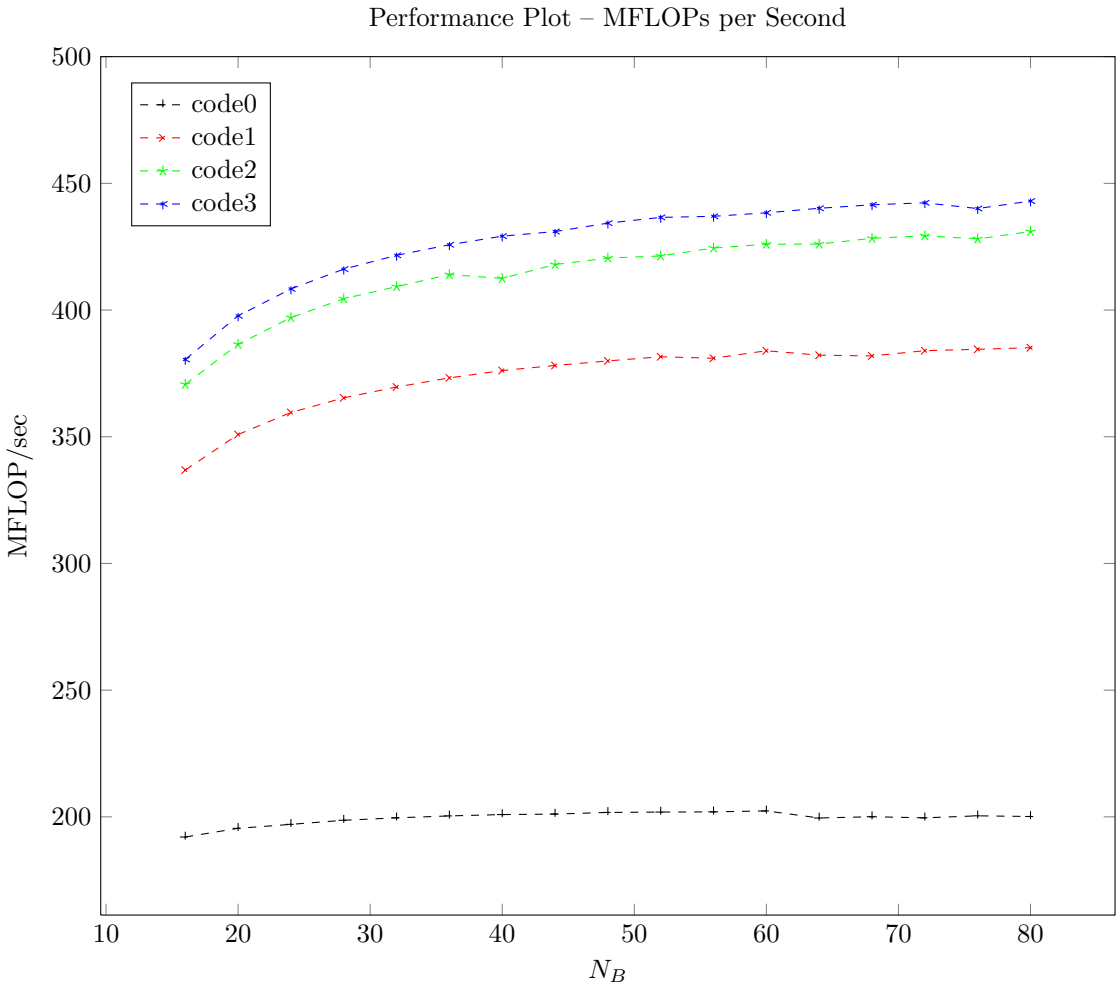
4 Experiments Performed & Results

4.1 Optimations of Mini-MMM: Blocking, Unrolling, and Scalar Replacement

Here the following implementations are compared for a range of block sizes such that $16 \leq N_B \leq 80$, with 4 divides N_B.

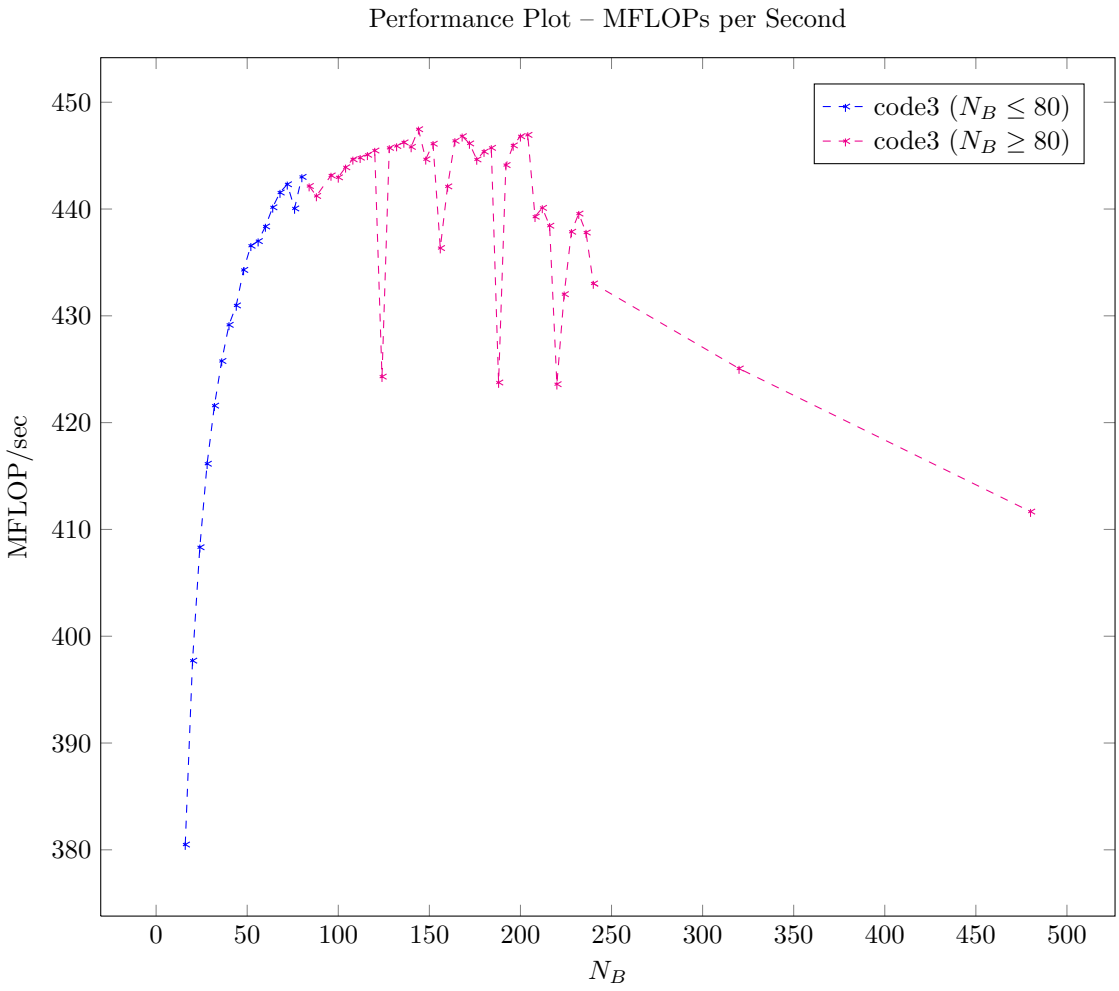
- `code0` – the numerical recipes code;
- `code1` – blocked into micro MMMs and unrolled;
- `code2` – further unrolling; and
- `code3` – even further unrolling.

Since, in this comparison, the blocks and the matrices both have a size $N_B \times N_B$, the time to compute the MMM is not useful for determining a optimal N_B, therefore we must instead compare the performance of each block in terms of MFLOPs/sec.

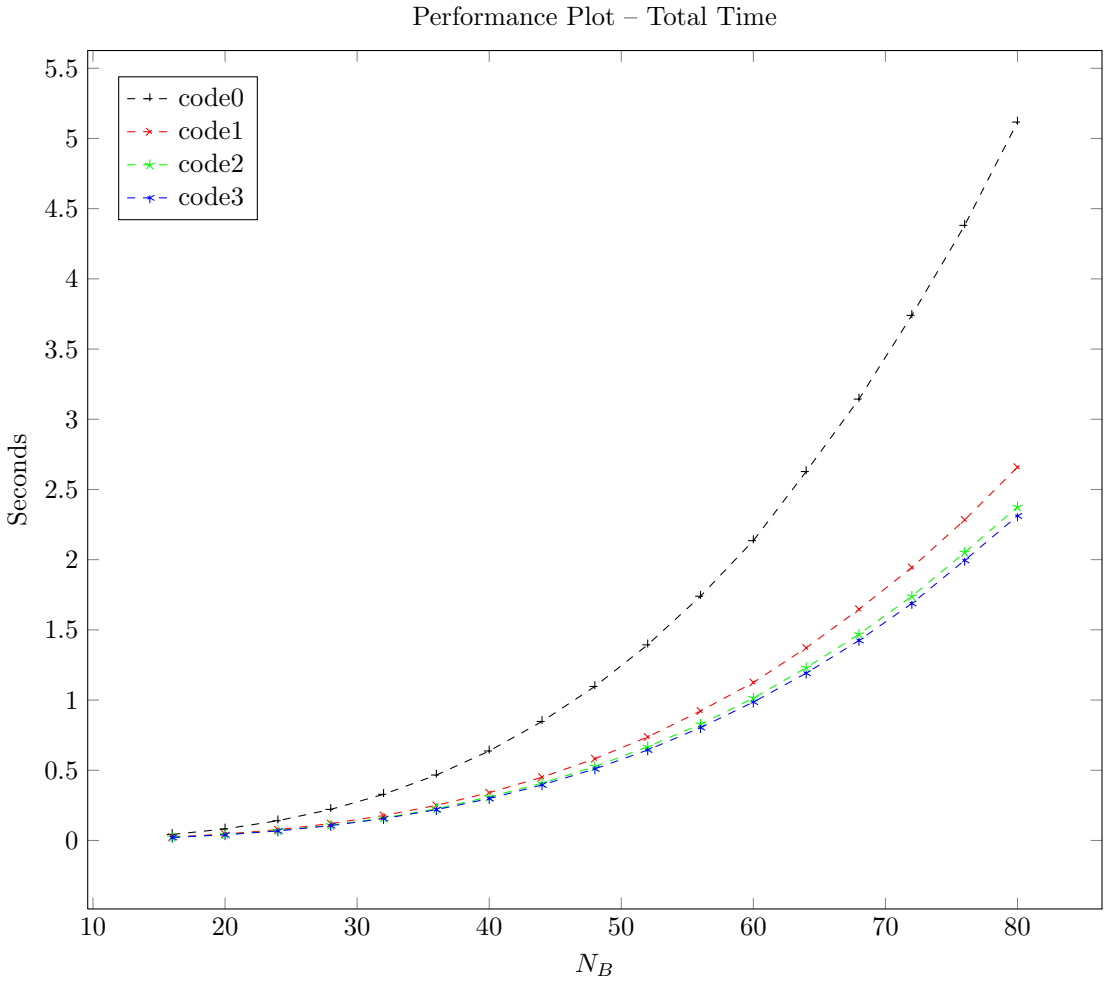


Here, it is clear that `code3` outperforms all other implementations, and it appears that the optimal size for N_B is $N_B = 80$; however, since this is seemingly a completely arbitrary and baseless number, it cannot be taken for granted.

If one instead examines possible block sizes as large as $N_B = 480$ and beyond, one sees that $N_B = 80$ is not in fact the highest-performing block size.



This reveals that peak performance is achieved at $N_B = 144$. One can notice obviously large amounts of data cache misses at $N_B = \{124, 156, 188, 220\}$.

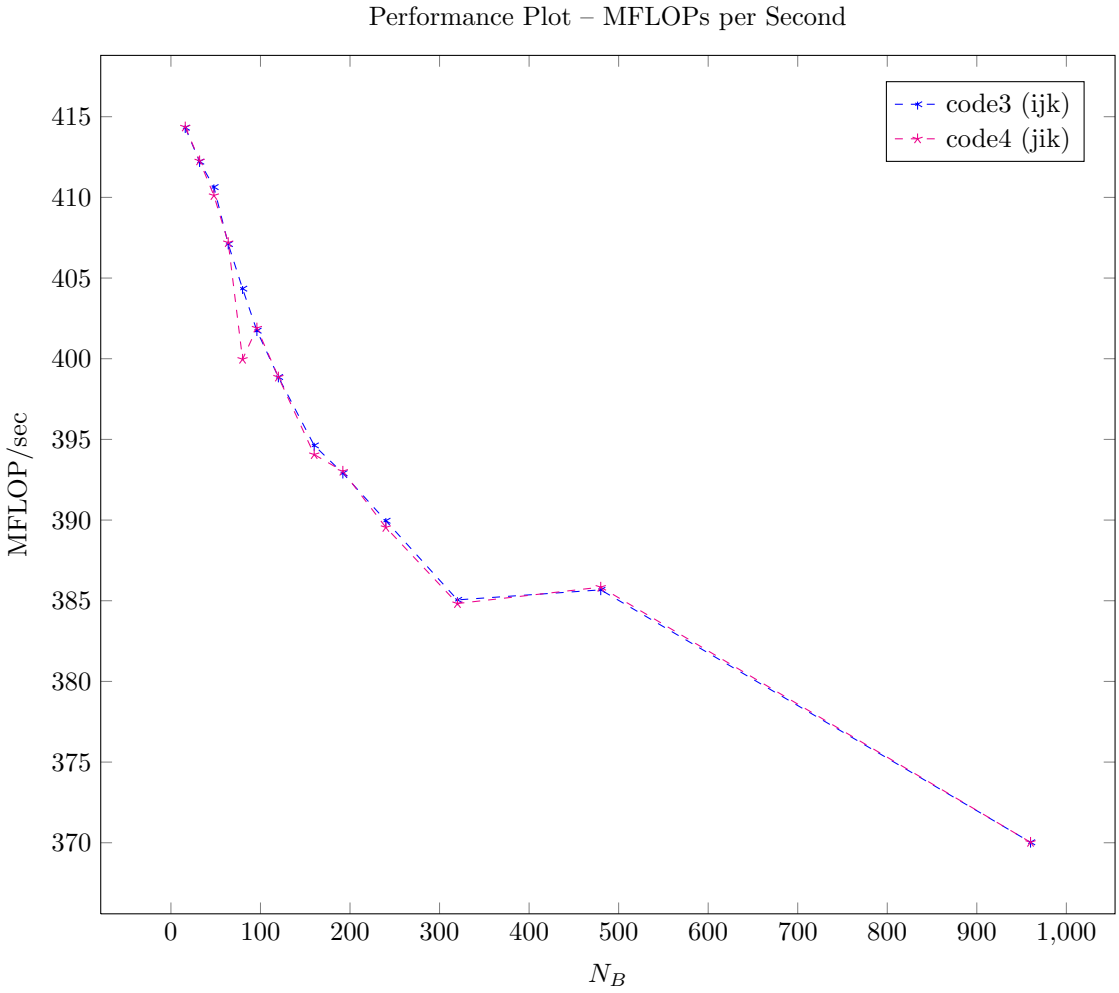


This plot gives an idea of how much faster each version is for various block sizes. There is a nice, smooth trend.

5 Loop Ordering

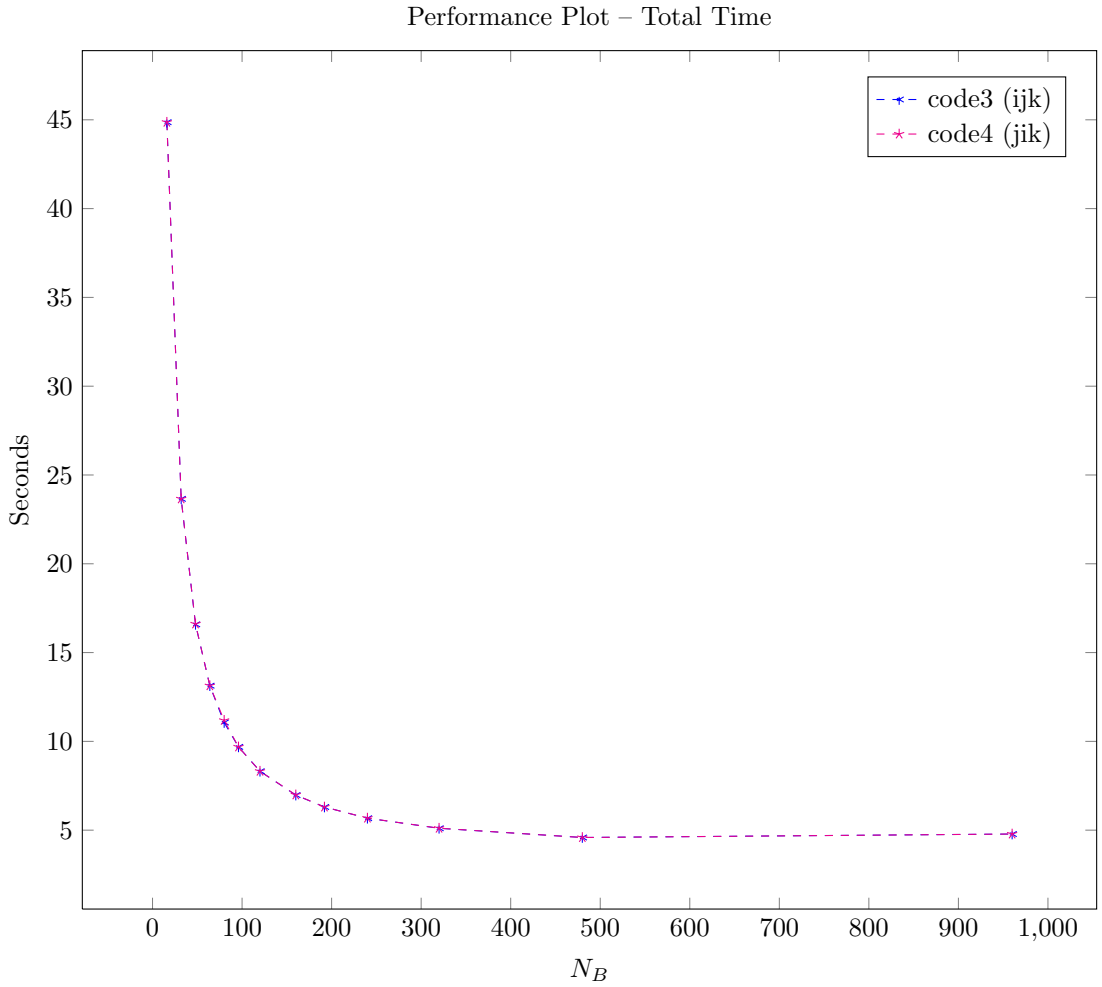
When comparing the ordering of the outermost loop, it now becomes necessary to move beyond the performance of single-block matrices, to the performance of small blocks within larger matrices.

Since it is necessary that N_B divides into the dimensions of the matrix, we must choose certain values. Selecting a reasonable set S of block sizes and a reasonable sized $N \times N$ matrix such that LeastCommonMultiple$(S) = N$ gives $S = \{16, 32, 48, 64, 80, 96, 120, 160, 192, 240, 320, 480, 960\}$, and $N = 960$.



From this graph it appears that there is no performance increase as a result of the new loop ordering.

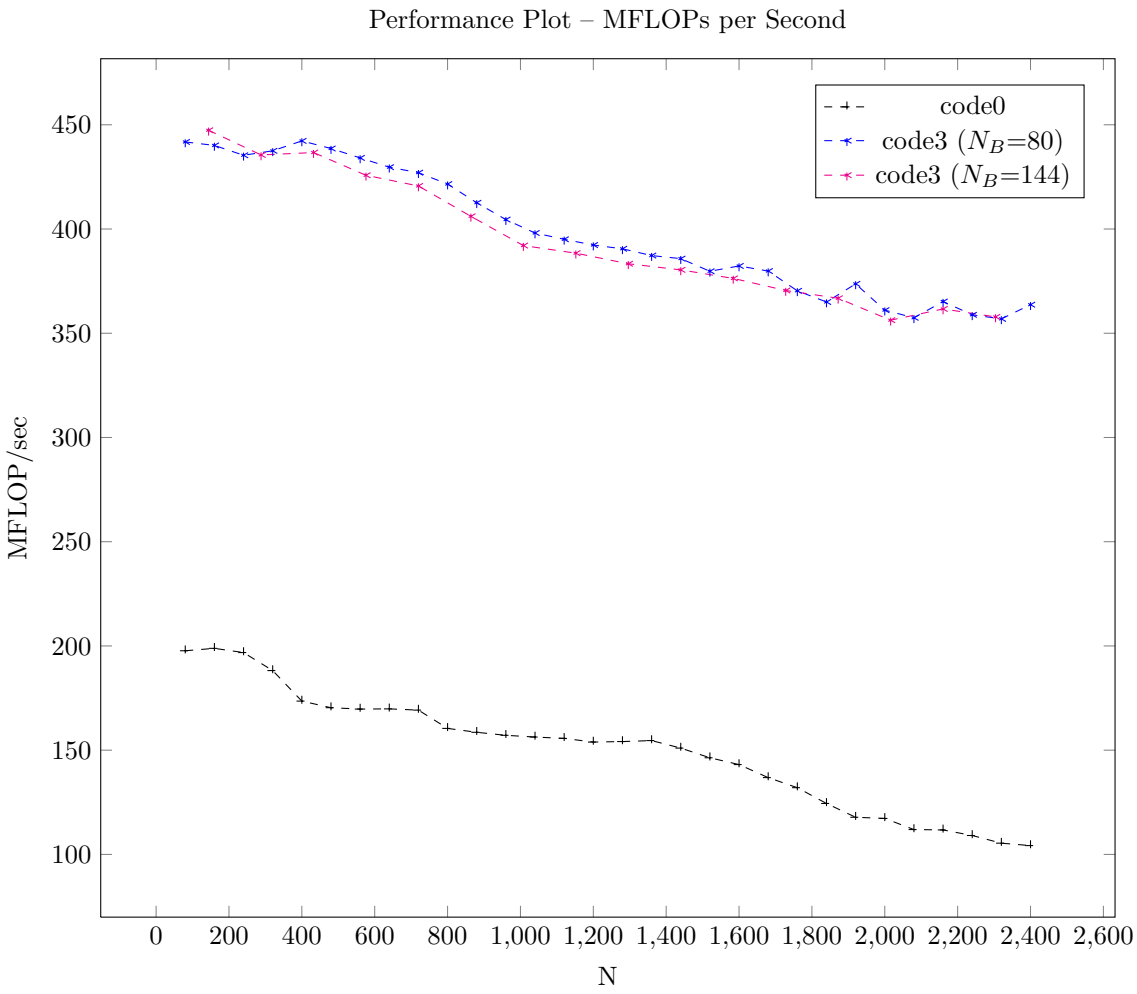
It is possible to check the timing of these different implementations of loop order for a more definite answer.



There is clearly no difference between the loop orderings.

5.1 Matrix Size Comparisons

Finally, one can compare the the performance of high-performing block sizes $N_B = \{80, 144\}$ for a range of matrix dimensions $0 \leq N \leq 2400$.



This reveals that perhaps the max performance of a particular block size does not wholly indicate how it will perform on larger, multi-block matrices – however these block sizes are still very close.

5.2 Source Code & Data

An explanation of the source used to generate these graphs as well as the resulting data is contained in the included tarball in the `README` file.

6 Conclusion

To conclude, a number of observations can be made:

- Blocking, unrolling, and scalar replacement all greatly improve performance.
- Optimal block size depends on the L1 cache size
- Don't trust arbitrary limits in assignments that are ≥ 7 years old.