CS540 Assignment 1

Dustin Ingram

October 22, 2011

1 Introduction

For this assignment, various methods of matrix multiplication are compared, including different compiler optimizations, different loop ordering, block matrix multiplication, and matrix multiplication implementations provided by high-performance libraries. For each configuration, three metrics are examined for various array sizes: the total number of Floating Point Operations (FLOP) per second, the L1 Data Cache misses (normalized over the number of FLOPs), and the total number of instructions (also normalized over the number of FLOPs).

2 Summary of Results

The results can be organized as follows:

2.1 GCC Optimizations

It was found that GCC optimizations give good improvement in speed and instructions/FLOP, however only trivially reduced the number of L1 Data Cache misses. The optimal optimization was `-O2` or `-O3`, as they were essentially the same.

2.2 Loop Ordering

It was found that various methods of loop ordering can vastly improve the number of L1 Data Cache misses, thereby improving the total MFLOPS significantly. However, this had no effect on the number of instructions per FLOP. The optimal loop ordering was either `ikj` or `kij`, as they were essentially the same.

2.3 Block Matrix Multiply

Using a block matrix multiply algorithm similarly improved the number of L1 Data Cache misses, and consequently the total MFLOPS as compared to the regular, unoptimized `ijk` matrix multiply. However, unlike the loop orderings, block matrix multiplication had a much higher number of instructions/FLOP.

2.4 ATLAS & MKL

The high-performance matrix multiplication algorithms provided by the ATLAS and MKL libraries thoroughly out-performed any other implementation, but the complex implementation may make them prohibitively time-consuming for the developer, resulting in a high ratio of WTFs per second. It would seem that MKL will perform better than ATLAS for large matrices (greater than 600x600).

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:
```

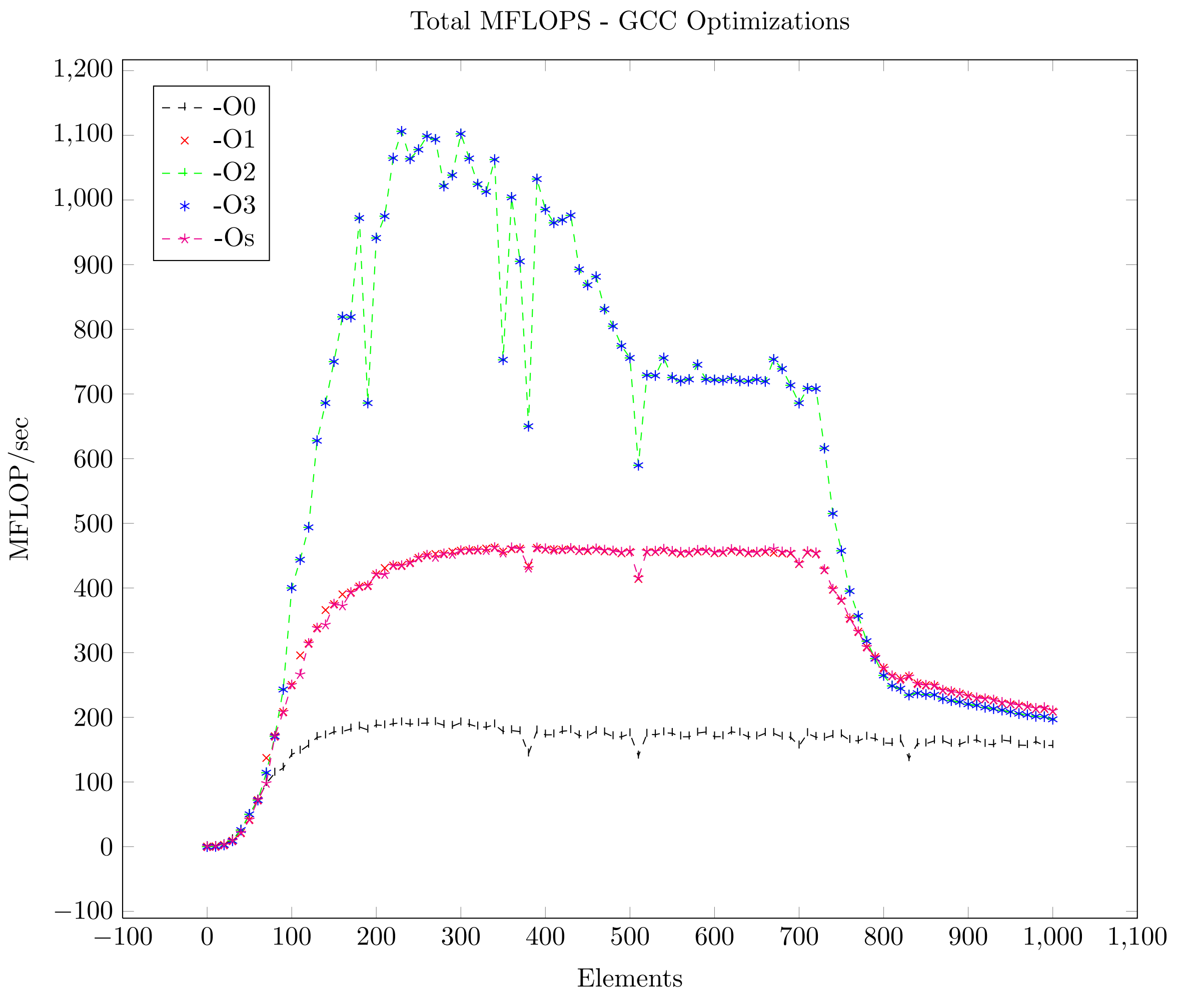
3.4 Memory Information

```
$ 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
```

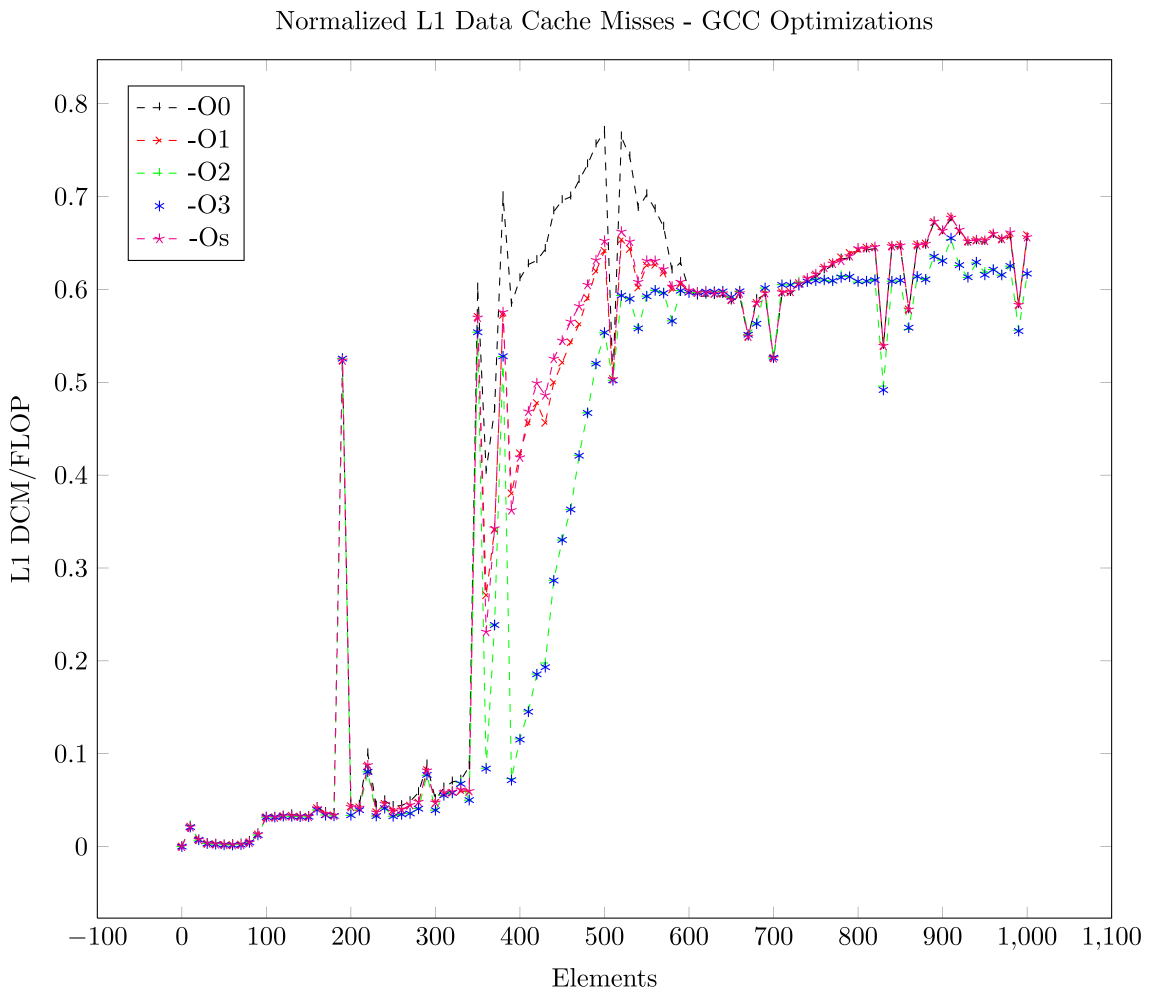
```
L1 Data TLB:
  Page Size:              4 KB
  Number of Entries:     64
  Associativity:          4

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

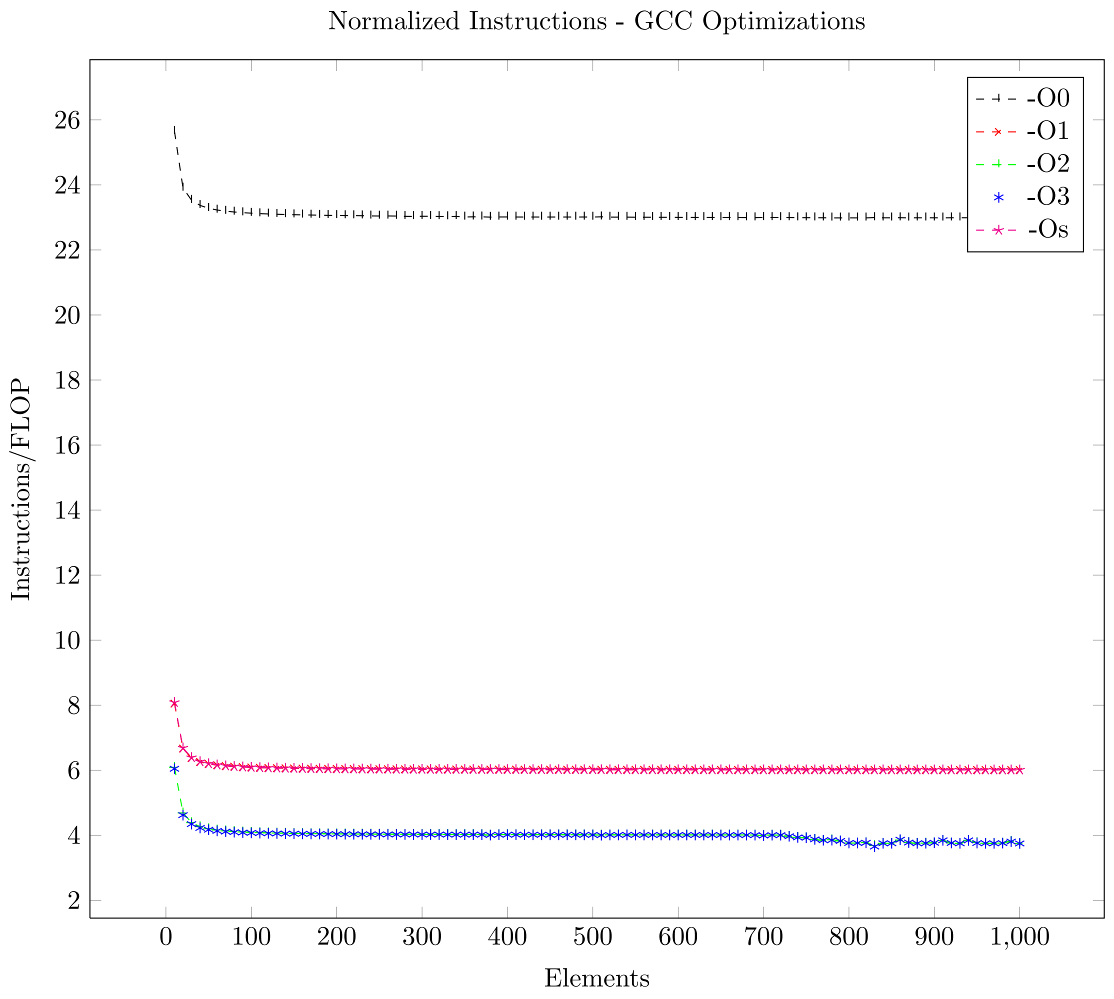
4.1 Optimations of `matrix` with GCC



Here we see that the `-O1` and `-Os` optimizations are essentially identical, as are the `-O2` and `-O3` optimizations, which appear to be the best performing optimizations.

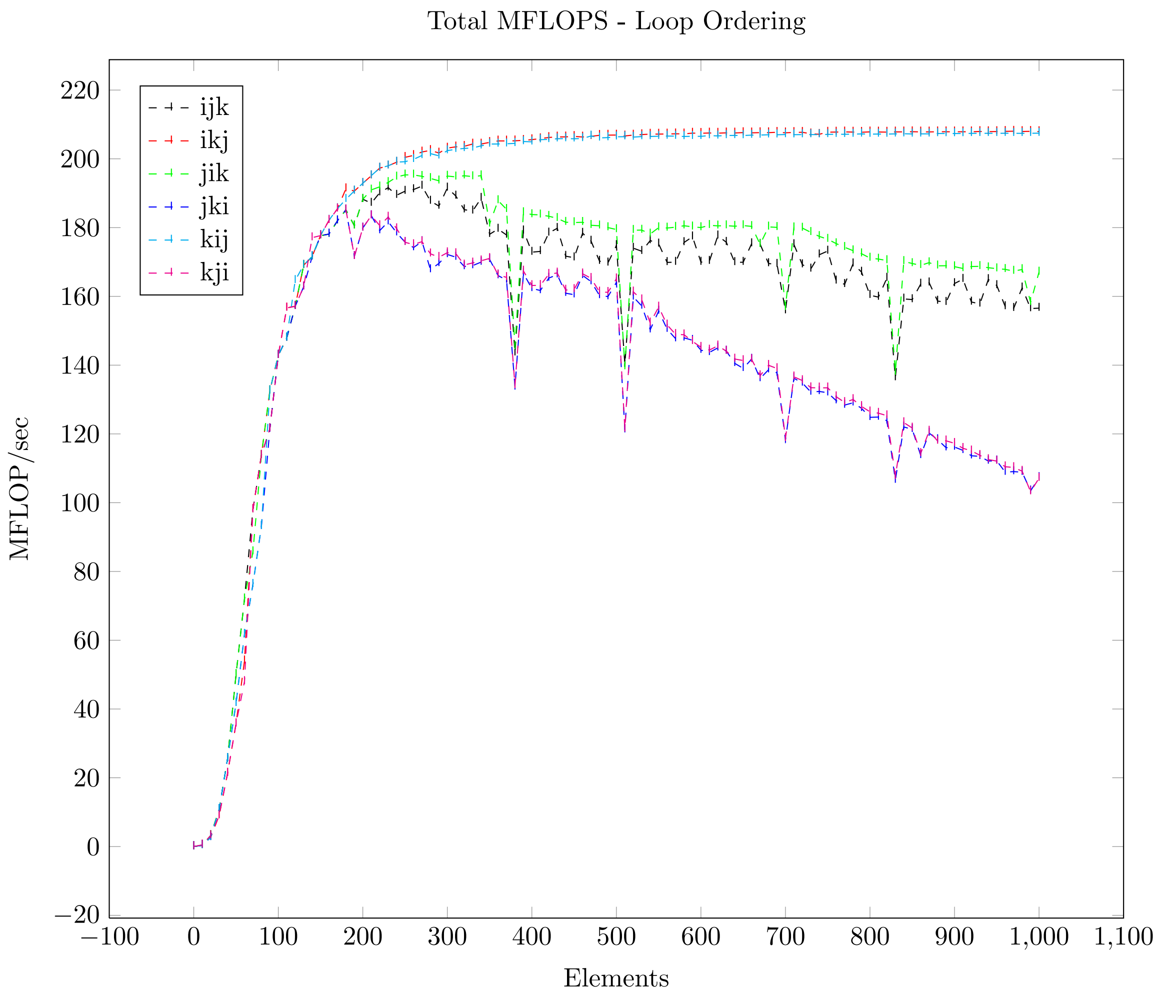


Again, we see that the `-O1` and `-Os` optimizations are essentially identical, as are the `-O2` and `-O3` optimizations. There is no clear optimization that is significaltly better, so we may assume that the optimization flags to not attempt to improve data cache misses.

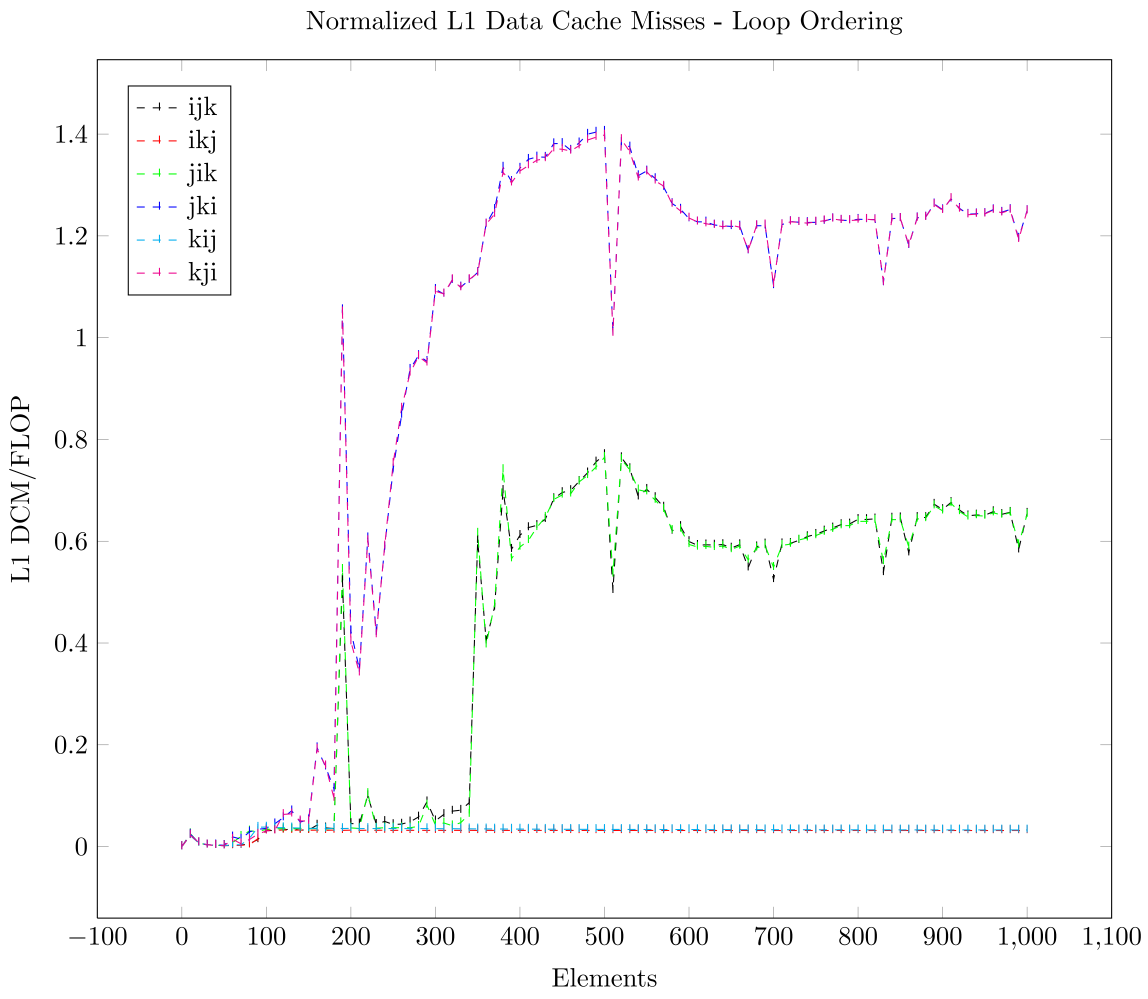


Again, we see that the `-O1` and `-Os` optimizations are essentially identical, as are the `-O2` and `-O3` optimizations, and that the total number of instructions per FLOP for each is approaching a constant value. Here we see the speed of the optimizations is a results of a lower number of instructions per FLOP.

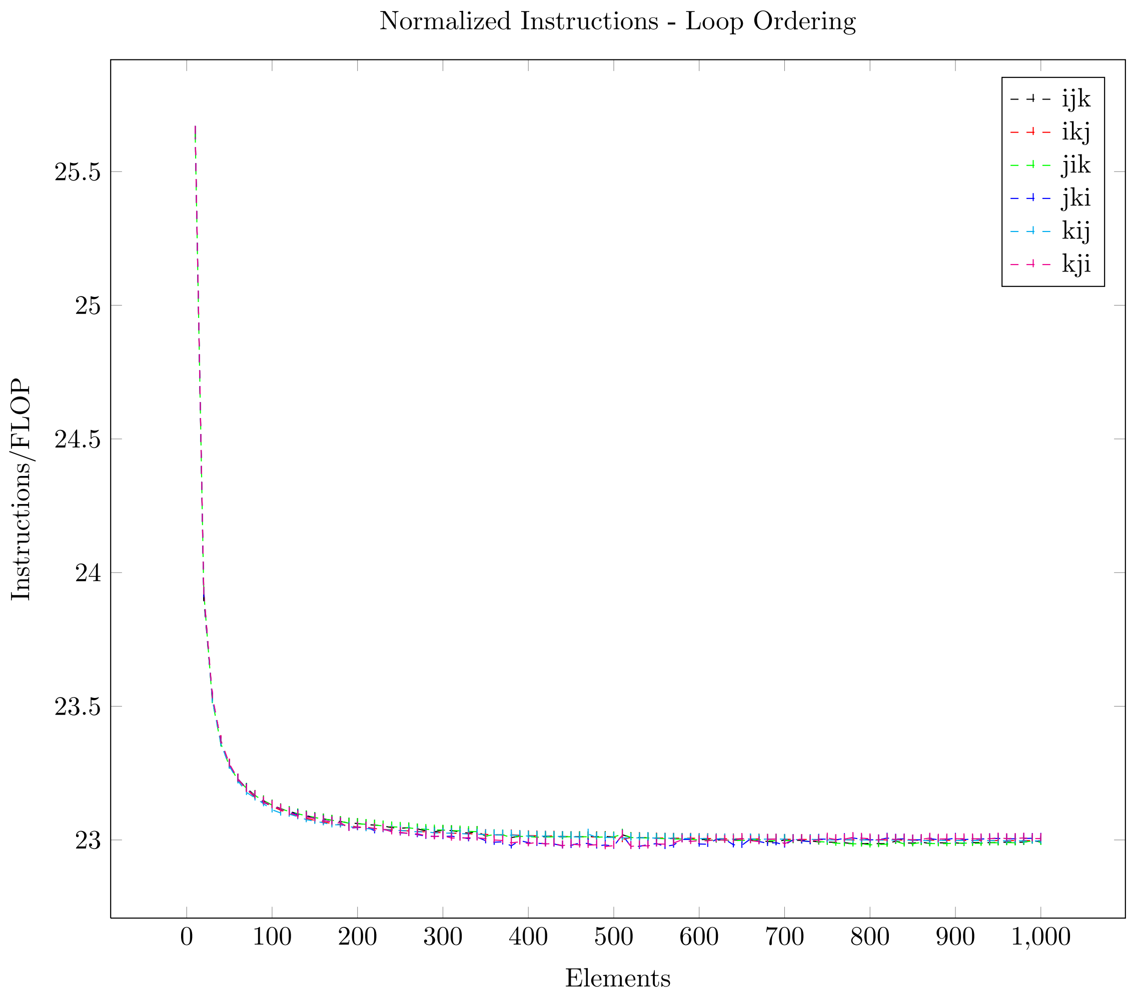
5 Loop Ordering



It is a little hard to distinguish, but we can see how the speed of the `ikj` and `kij` orders remain consistently high, while others quickly succumb to lower speeds.

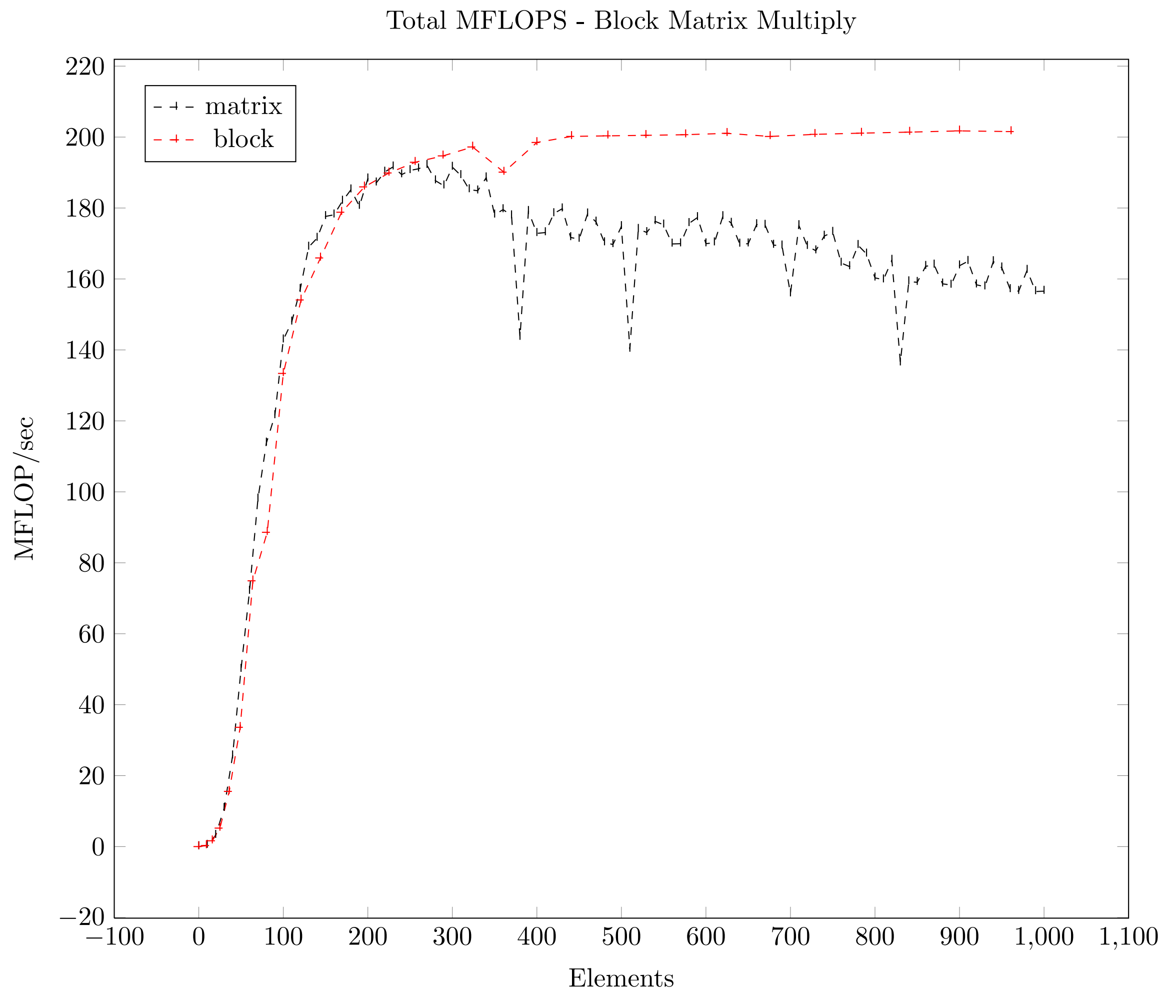


Here it is evindent that extremely low data cache misses will result in higher speeds. It is unfortunate that a larger test array was not run to force the `ikj` and `kij` orderings to exceed the size of the L1 data cache and result in some misses.

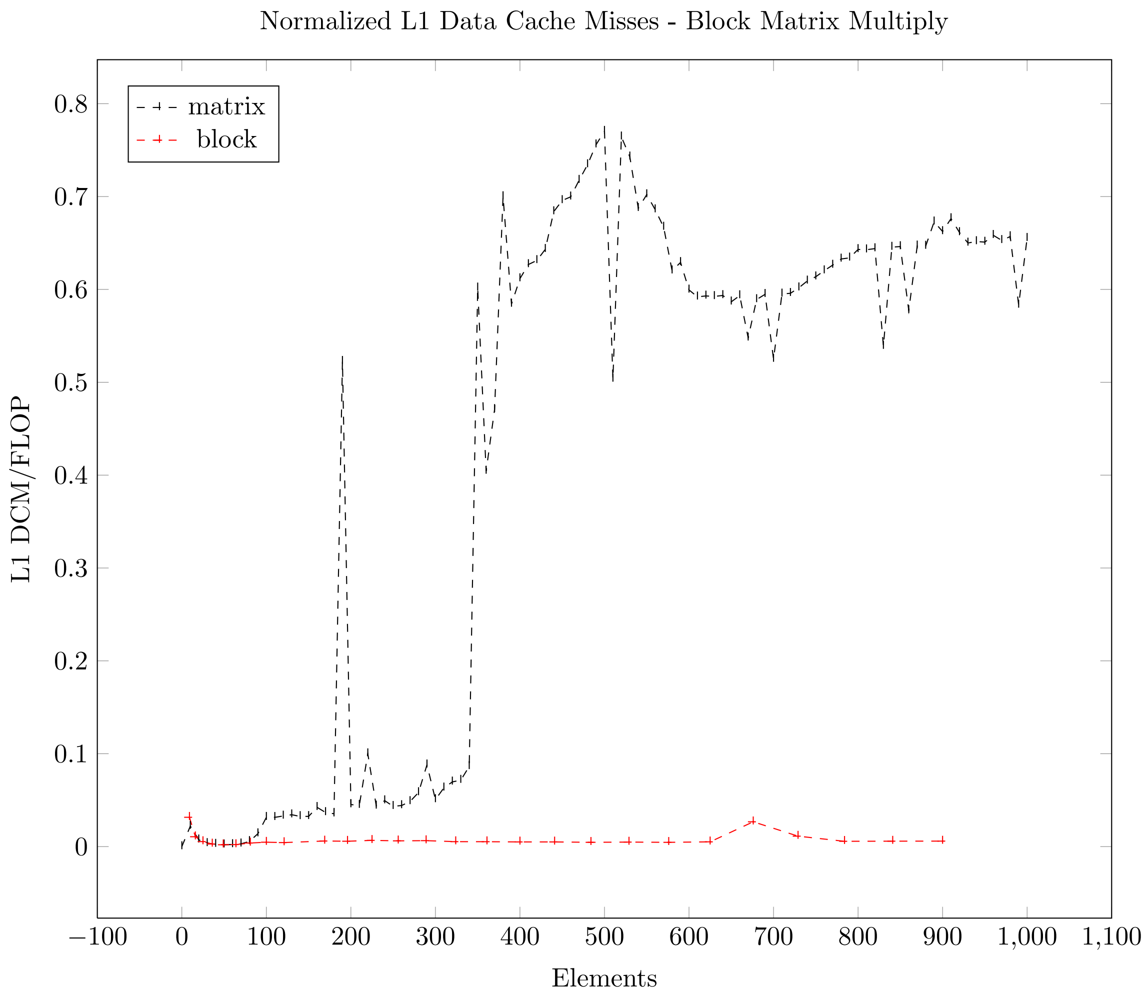


Here it is clear that the number and amount of data cache misses has no bearing on the number of instructions per FLOP.

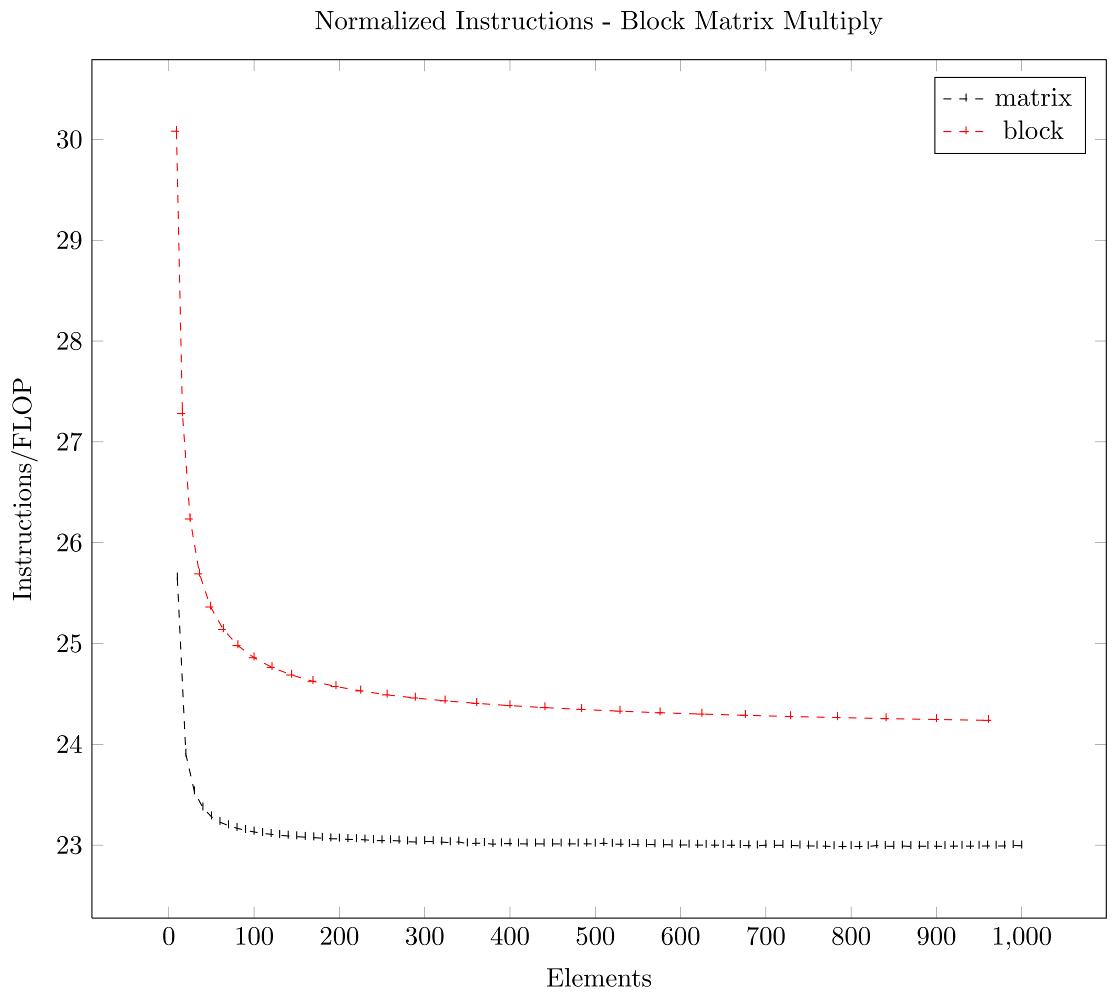
5.1 Block Matrix Multiply



Block matrix multiplication offers a clear improvement in speed over a normal `ijk` algorithm, but it is not much.

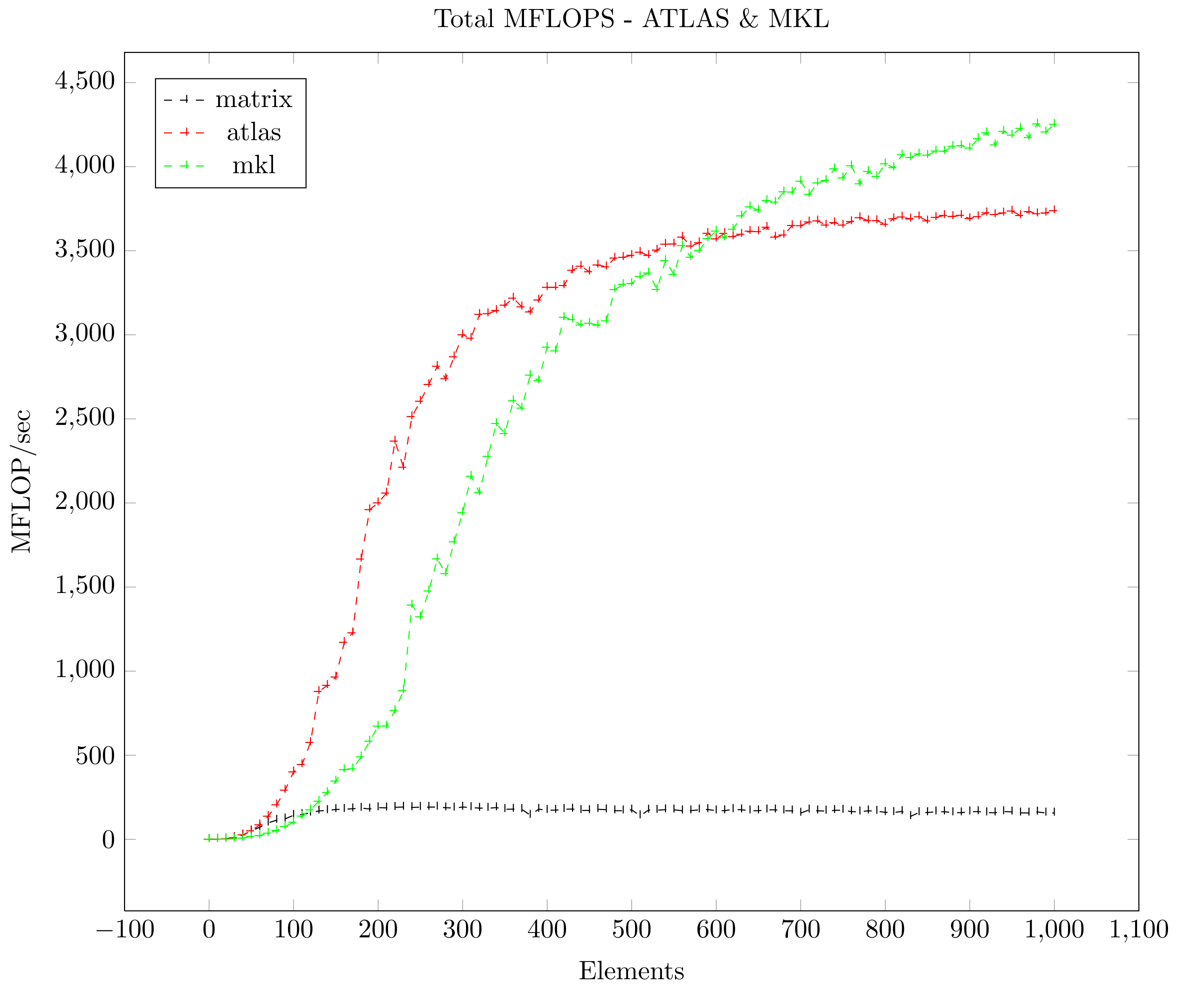


Again, this speed is due to relatively low numbers of data cache misses.

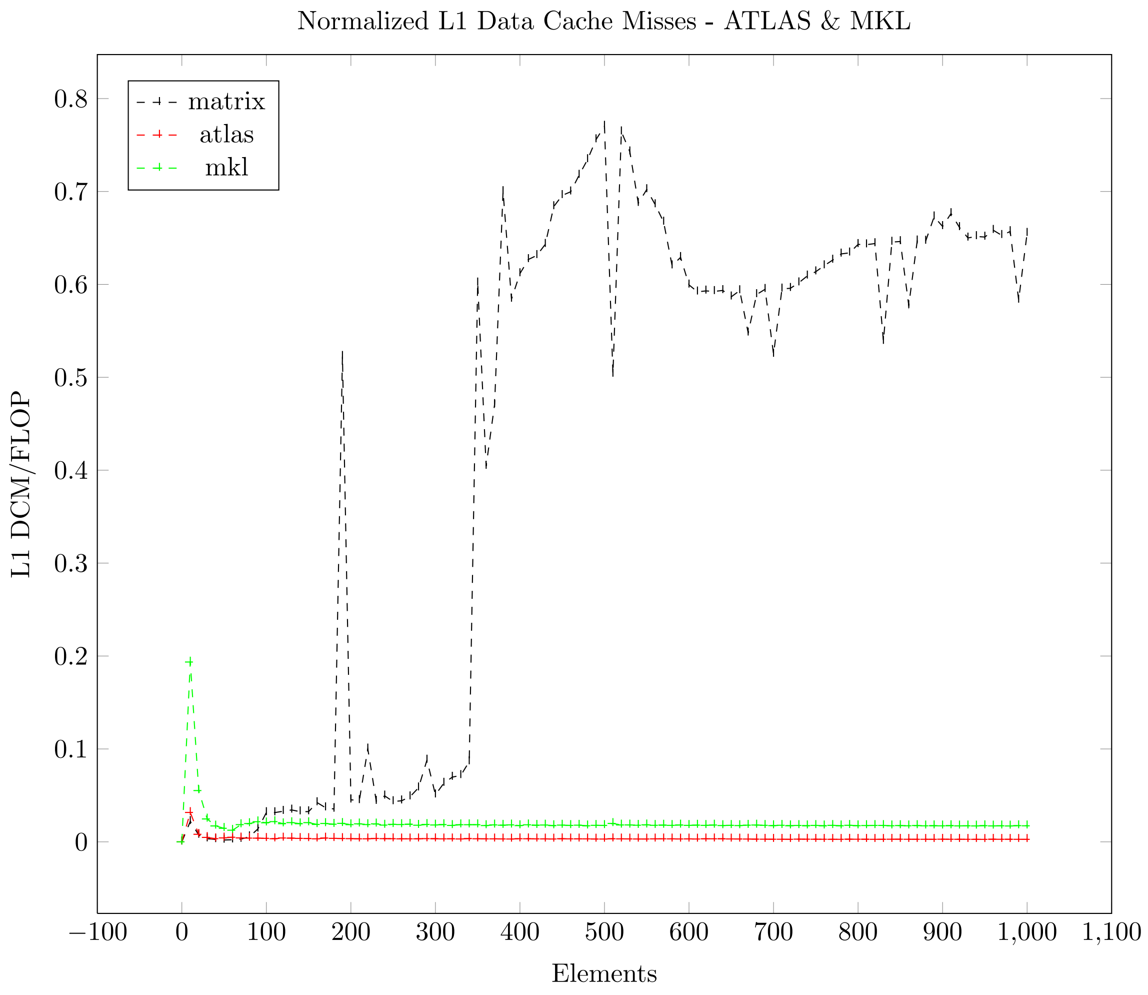


Block matrix multiplication does result in a higher number of instructions/FLOP, however.

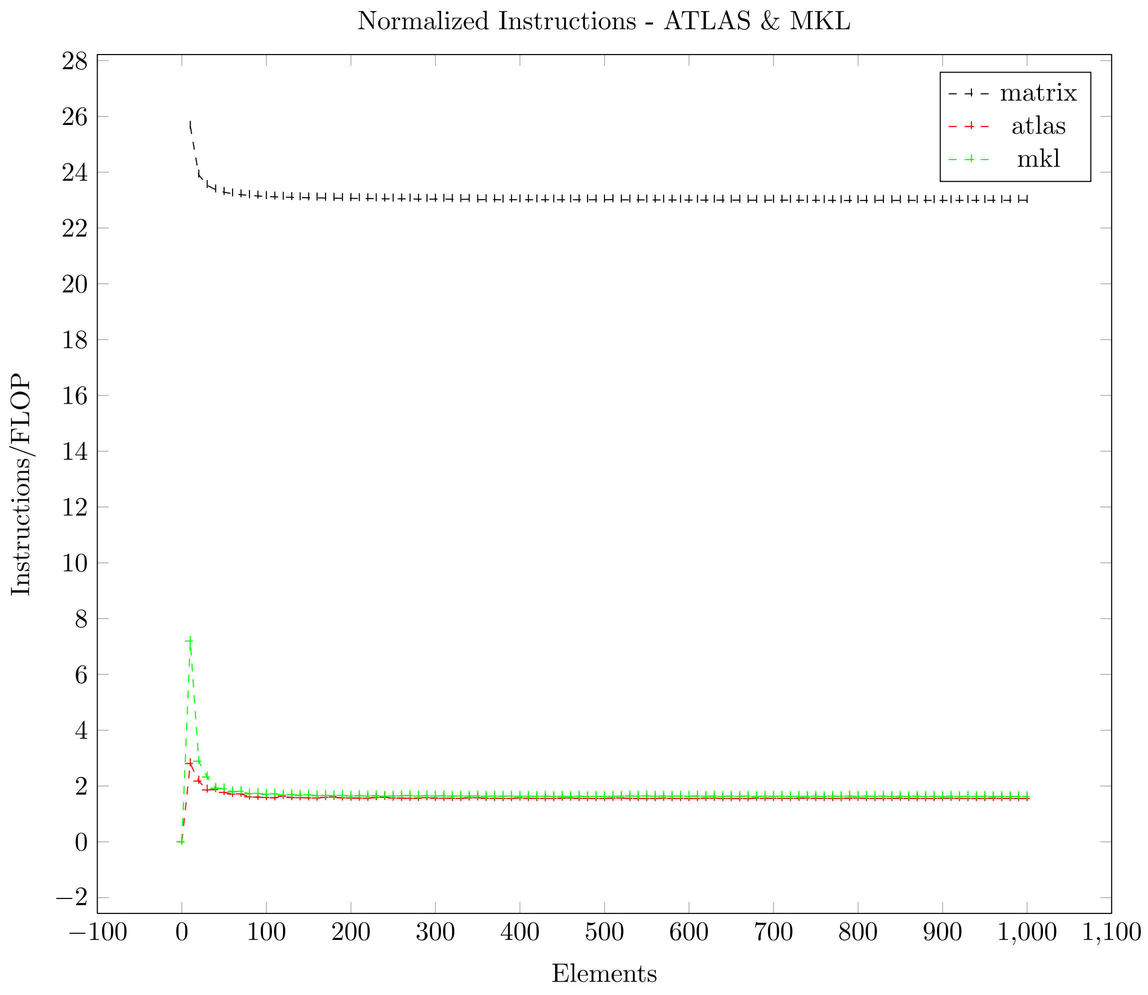
5.2 MKL & ATLAS



This graph is particularly interesting, as it shows that ATLAS initially is out-performing MKL for relatively small sizes of matrices, however after a matrix size of approximately 600x600, the higher growth rate of the MKL implementation catches up to ATLAS and provides a faster algorithm.



As expected, L1 data cache misses are extremely low. Again, it is unfortunate that the chosen maximum number of elements is within the L1 cache size.



ATLAS and MKL both offer an extremely low number of instructions/FLOP.

5.3 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:

- GCC offers a quick and easy way to get a minimal amount of optimization;
- Carefully planning how an algorithm loops through and accesses data may result in higher performance through lower data cache misses;
- Block matrix multiplication is a well performing alternative due to lower data cache misses
- If you're trying to implement something yourself, chances are there's already an algorithm in a library somewhere which will outperform anything you could independently write by several orders of magnitude, though it might be unreasonably complicated and definitely under-documented.