ASTR 415

Evan Shipley-Friedt

October 4, 2022

Problem Set #2

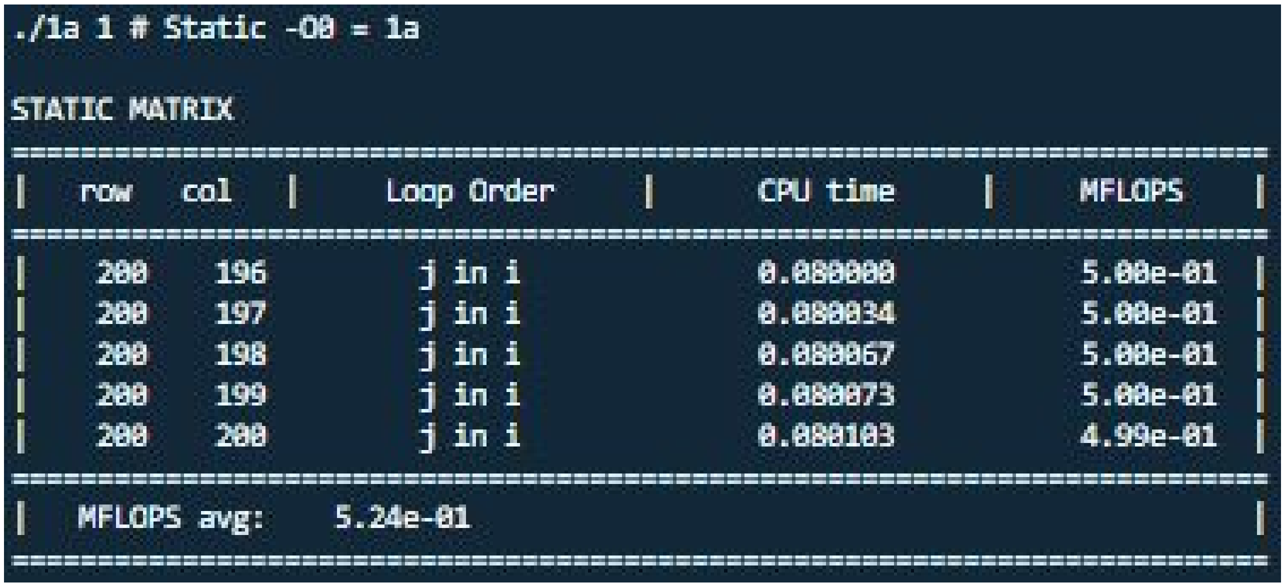
1

After compiling without optimizations, time the execution in two cases: (A) with the fast changing index of the array being the columns, and (B) being the rows. Make sure you run the code without optimization (-O0).

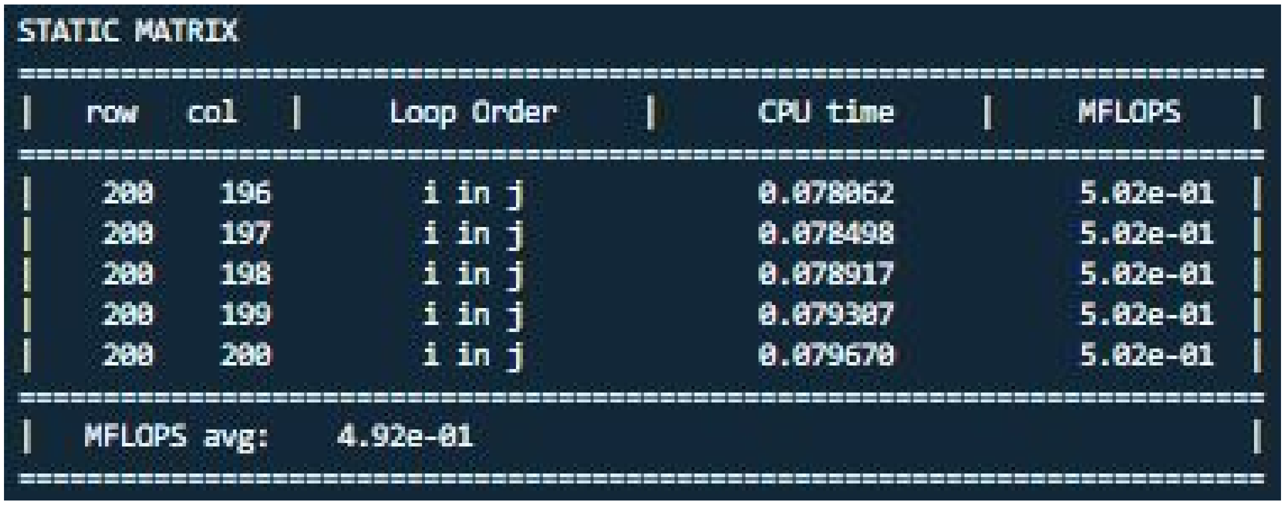
1.a Problem Statement Summary

(a) First define the array as a static array (e.g., double A[N][N];). How large can N be before you encounter a problem at runtime ? Explain what happens and how can you increase N further.

CPU Expense Tables using -O0 for a Static Matrix



```
./1a 1 # Static -O0 = 1a

STATIC MATRIX
==============================================================================
|   row   col   |     Loop Order     |      CPU time      |      MFLOPS    |
==============================================================================
|    200    196        j in i              0.080000             5.00e-01  |
|    200    197        j in i              0.080034             5.00e-01  |
|    200    198        j in i              0.080067             5.00e-01  |
|    200    199        j in i              0.080073             5.00e-01  |
|    200    200        j in i              0.080103             4.99e-01  |
==============================================================================
|   MFLOPS avg:    5.24e-01                                                 |
==============================================================================
```

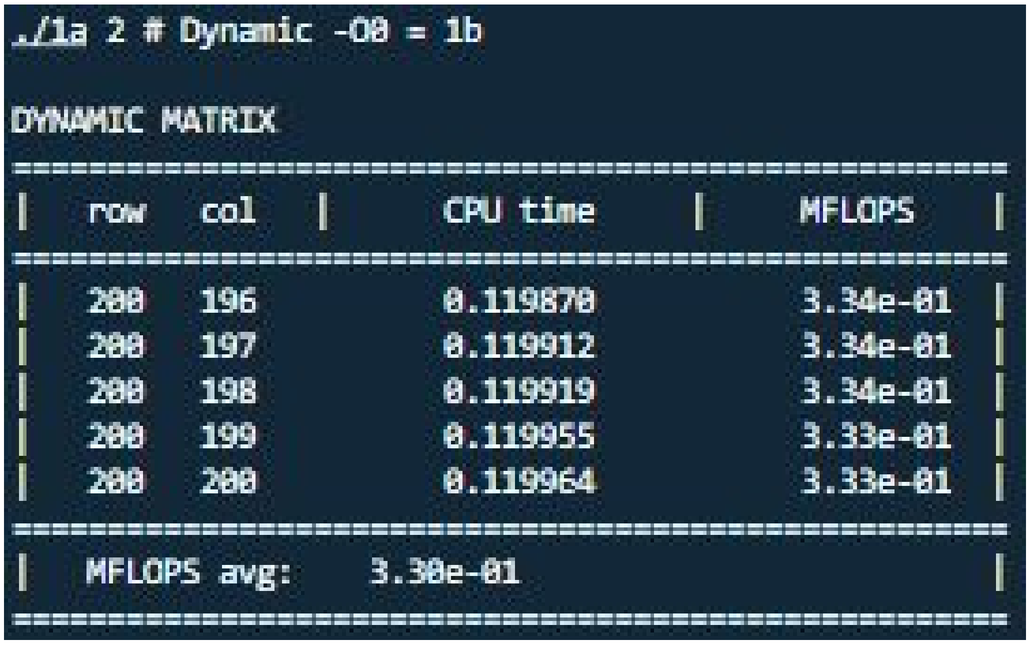


```
STATIC MATRIX
=========================================================================
|   row   col   |      Loop Order     |      CPU time      |     MFLOPS   |
=========================================================================
|    200   196          i in j              0.078062            5.02e-01  |
|    200   197          i in j              0.078498            5.02e-01  |
|    200   198          i in j              0.078917            5.02e-01  |
|    200   199          i in j              0.079307            5.02e-01  |
|    200   200          i in j              0.079670            5.02e-01  |
=========================================================================
|   MFLOPS avg:    4.92e-01                                               |
=========================================================================
```

1.b Problem Statement Summary

(b) Next, use dynamical allocation of memory (with malloc). Knowing the difference in execution times between case (A) and (B) for a set of N values, given the clock speed can you determine the size of the cache and/or the hit rate? Explain your reasoning.

CPU Expense Table using -O0 for a Dynamic Matrix



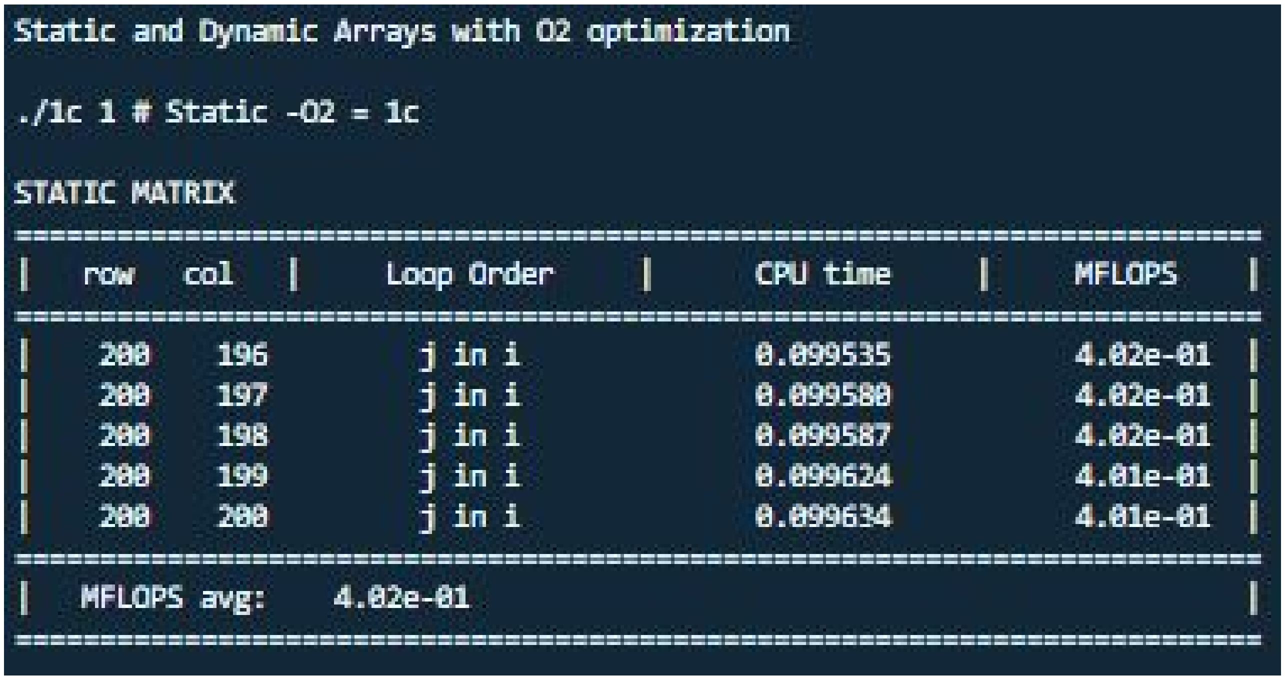
```
./1a 2 # Dynamic -O0 = 1b

DYNAMIC MATRIX
=========================================================
|   row   col   |       CPU time      |     MFLOPS     |
=========================================================
|   200   196          0.119870            3.34e-01    |
|   200   197          0.119912            3.34e-01    |
|   200   198          0.119919            3.34e-01    |
|   200   199          0.119955            3.33e-01    |
|   200   200          0.119964            3.33e-01    |
=========================================================
|   MFLOPS avg:    3.30e-01                            |
=========================================================
```

1.c Problem Statement Summary

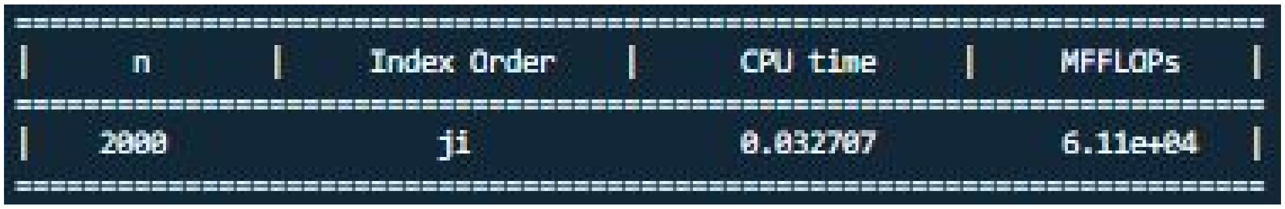
(c) Finally, compile with -O2 optimization and compare the execution times for a large value of N . Comment on the results.

CPU Expense Table using -O2

Static Matrix

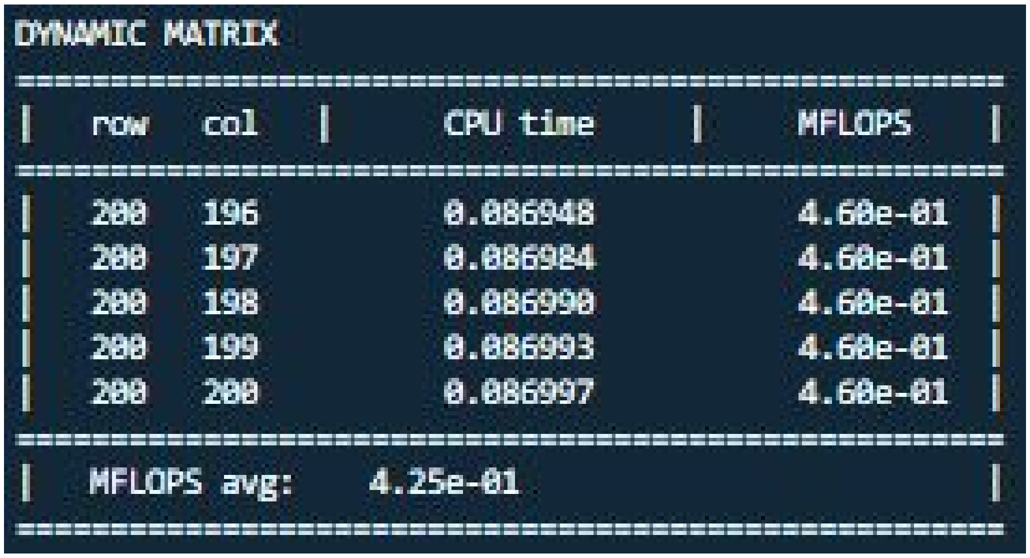


```
Static and Dynamic Arrays with O2 optimization

./1c 1 # Static -O2 = 1c

STATIC MATRIX
===============================================================================
|   row   col   |       Loop Order      |       CPU time       |      MFLOPS    |
===============================================================================
|    200   196          j in i                0.099535              4.02e-01    |
|    200   197          j in i                0.099580              4.02e-01    |
|    200   198          j in i                0.099587              4.02e-01    |
|    200   199          j in i                0.099624              4.01e-01    |
|    200   200          j in i                0.099634              4.01e-01    |
===============================================================================
|   MFLOPS avg:    4.02e-01                                                     |
===============================================================================
```

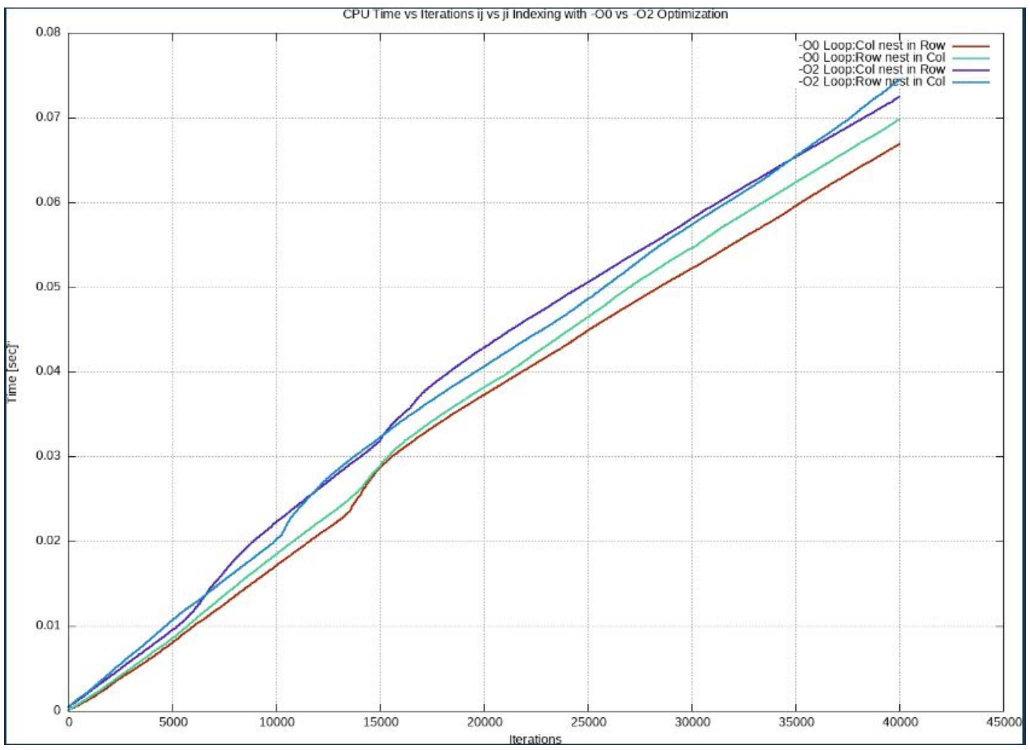


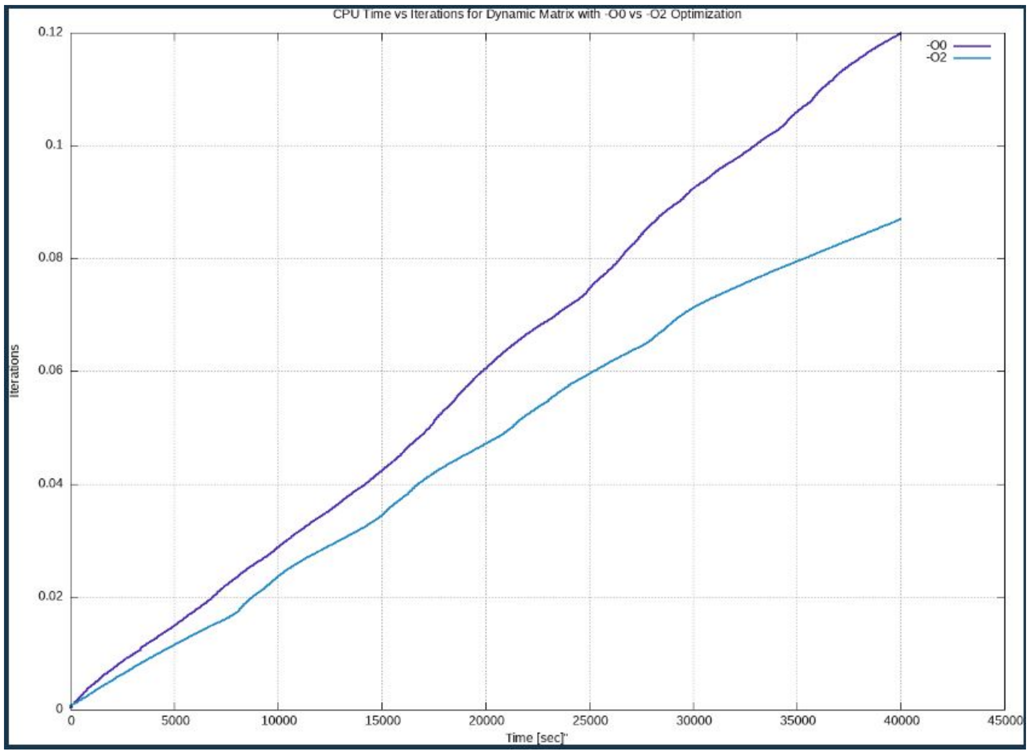
```
===============================================================================
|       n        |      Index Order      |      CPU time       |     MFFLOPs    |
===============================================================================
|    2000                  ji                  0.032707              6.11e+04   |
===============================================================================
```

Dynamic Matrix



```
DYNAMIC MATRIX
==========================================================
|   row   col   |       CPU time       |      MFLOPS     |
==========================================================
|   200   196           0.086948              4.60e-01   |
|   200   197           0.086984              4.60e-01   |
|   200   198           0.086990              4.60e-01   |
|   200   199           0.086993              4.60e-01   |
|   200   200           0.086997              4.60e-01   |
==========================================================
|   MFLOPS avg:    4.25e-01                              |
==========================================================
```


CPU Time vs Iterations ij vs ji Indexing with -O0 vs -O2 Optimization
-O0 Loop:Col nest in Row
-O0 Loop:Row nest in Col
-O2 Loop:Col nest in Row
-O2 Loop:Row nest in Col
Time [sec]
Iterations
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0
5000
10000
15000
20000
25000
30000
35000
40000
45000


CPU Time vs Iterations for Dynamic Matrix with -O0 vs -O2 Optimization
-O0
-O2
Iterations
Time [sec]
0
0.02
0.04
0.06
0.08
0.1
0.12
0
5000
10000
15000
20000
25000
30000
35000
40000
45000

1 (a) Response

My stack size is limited to 8192 kb according to

```
$ ulimit -s
```

or -a (for more memory stats)

My cache seems to be using virtual memory on Ubuntu using WSL2, so my limiting factor is always my stack size. I tried to set my stack size to unlimited, but using

```
$ set ulimit -s unlimited
```

has no effect. I will need to look into whether I can change my stack size more.

In problem 1a the segmentation error problem arises when I set my matrix size to be anything larger than 401x401. Earlier in my working I was able to set it to 1047. I believe I used up a lot more of my remaining available stack size adding arguments to so that I could run every question sequentially using one .c file with one function each for timing the static matrix and the dynamic.

In my case I could increase n indefinetly if I could increase my stack size because I have seemingly unlimited cache running on virtual memory. However, using vmstat or pthread.h functions, or better still

```
$ cat /proc/meminfo
```

it appears that I could be limited to 923500 bytes in my cache or rather it seems I have that much data already stored in my cache. I would prefer not to run my code in the dynamic case to see how high of N values I can get before I fill up my cache. If I did have this maximum I could use then I could find a good estimate knowing that the clock timer uses 8 bytes, floats use 4 bytes, doubles use 8 bytes etc. I can just divide my cache size by what is left over and by either 4 for a float matrix or 8 for a double matrix and then take the square root with it being a square matrix.

$$x = \sqrt{923500/4} \approx 480$$

1 (b) Response

If your microprocessor fetches instructions from off-chip memory, you should be able to observe changes in the speed of execution by observing the microprocessor bus. Though, given the execution time differences between the static cases for the loop that causes misses versus the loop that has 100% hit rate you should be able to get a good estimate for the number of misses by knowing the clock speed. Comparing the dynamic matrix to the static one that causes misses should allow you to calculate the cache size the same way.

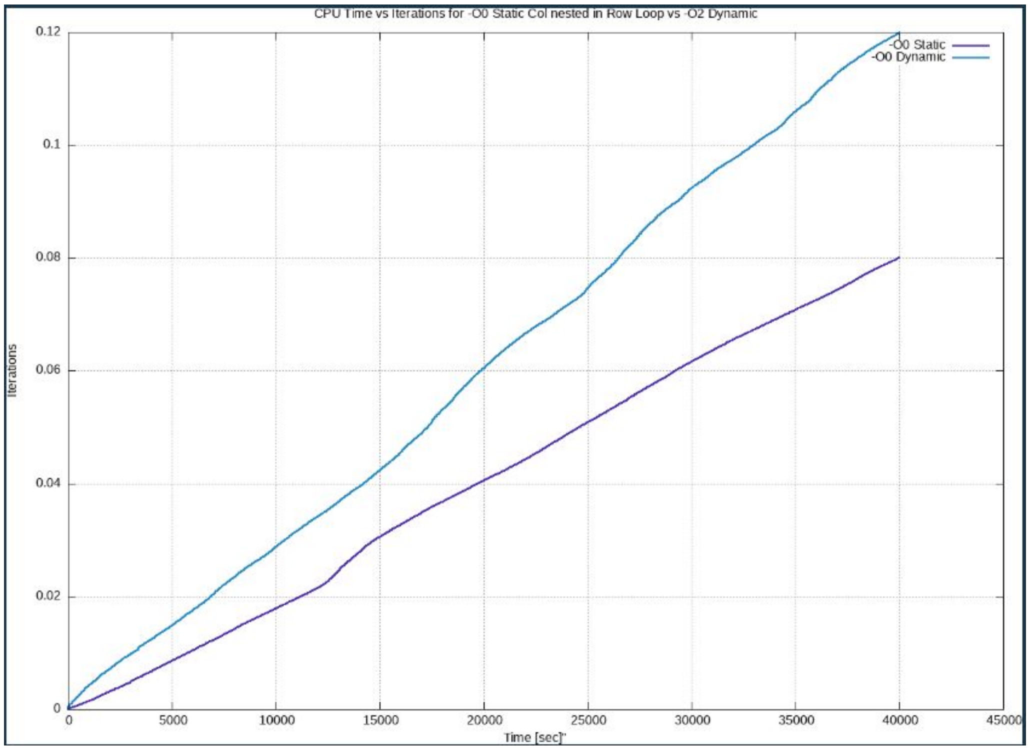
If we increase the block size while keeping the cache size the same, then we decrease the number of blocks that the cache can hold. Fewer blocks in the cache means fewer sets, and fewer sets means that collisions and therefore misses are more likely. Similarly if we increase the number of blocks the cache holds, then we decrease the effective size of the cache and can see a spike in the miss rate. If there is a spike in the miss rate then this could be a signal that you are near the cache size limit and therefore can deduce the size of the cache based on the size and number of blocks that were stored/accessed.

An easier way might be to just check GDB to determine the actual hit and miss rate for the L1, L2 and L3 caches.

1 (c) Response

I find using -O2 runs both the static and the dynamic matrix producing functions more slowly than -O0 in most cases. Sometimes running the code again produces different results.

Fastest of all is the static method with -O0 optimization.



2

Write a program that uses the quadratic formula to compute both roots of

$$x^2 - 4999x + 1 = 0$$

in single precision. The program should also recompute the smaller root from the larger, using the fact that the product of the roots must equal 1 in this case. Explain any discrepancy. Which method is preferable? What happens when you repeat this exercise in double precision?

1 (c) Response

Without solving for the second root using the first or some other advanced method such as multiplying the numerator and denominator of

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \ \ r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

by

$$b - \sqrt{b^2 - 4ac}$$

so that if $b^2 >> ac$ and $b > 0$ then using formula (1) for r1 will involve catastrophic cancellation, so instead we could use the resulting formula (2) for solving r_1 and (1) for solving r_1:

$$r_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}}, \ \ r_1 = \frac{2c}{-b + \sqrt{b^2 - 4ac}} \tag{2}$$

In this case we would do the opposite since $b << ac$. Yet, the easier way in the practice of coding is to just use the identities of Vieta's Theorem:

$$r_1 + r_2 = \frac{-b}{a} \quad \& \quad r_1 r_2 = \frac{c}{a}$$

Then if $b < 0$ like in our case we can again compute r_1 using the standard method in formula (1) and then solve for the other root with the strategy of knowing r_2 is equal to

$$r_2 = \frac{c}{r_1 a}$$