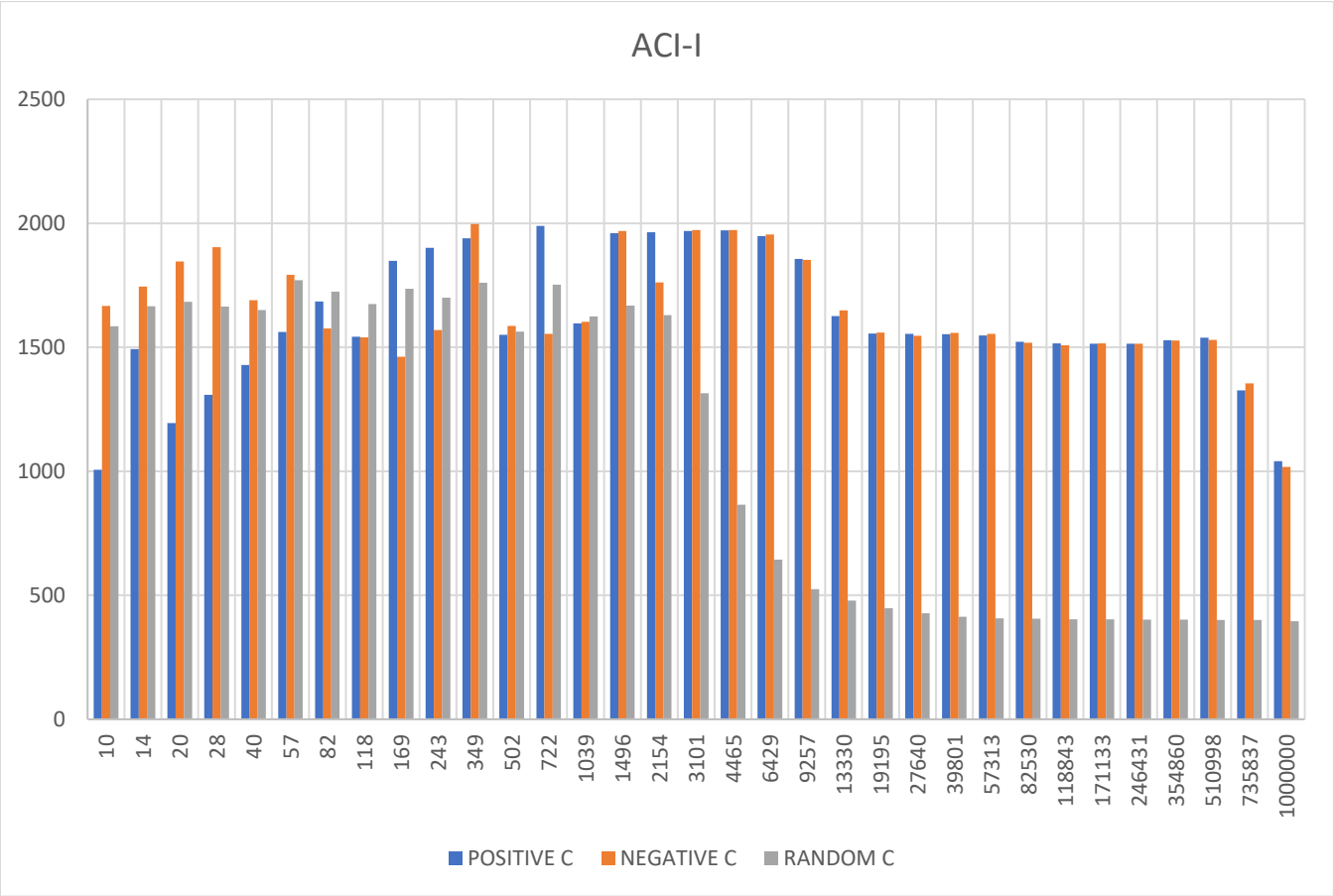
Shashank Ravichandran

Task 1:

Task 1 was a repetition of task 1 homework 1. However, the values of C were altered and the inner most for loop was altered. Below is a graph that Presents the MFLOPS value vs N value. Note the RANDOM C values are decimal values between -1 and 1 while POSITIVE and NEGATIVE C are constant integers.



To analyze this the graph should be split into the L1 cache, L2 cache and main. Note that the time taken to generate random C values was not taken into consideration for calculating MFLOPS, only the run time of the for loops was taken. Using homework 1 task 1, the N value for L1 is between 0 and 3,000, L2 N value is from 3,000 to about 50,000 and for main N value is above 320,000 till 10^6. For low N values (L1 for N) MFLOPS of NEGATIVE C and Random C are similar however, for PSITIVE C, MFLOPS starts off very low and at higher values of N it is similar to RANDOM and NEGATIVE C MFLOPS value. This shows that in L1 cache for smaller loops POSITIVE C values slow down the code. However, after N=82 till the end of L1 cache (N=3,000), RANDOM C starts to perform worse than POSITIVE and NEGATIVE C and POSITIVE and NEGATIVE C seem to have a very similar performance. In L2 POSITIVE and NEGATIVE C have identical performance whereas RANDOM C starts to slow down at a rapid pace. In main, the performance of POSITIVE and NEGATIVE C does start to taper off but is still higher than that of RANDOM C. So for larger

loops the performance of POSITIVE and NEGATIVE C is better than that of RANDOM C. When comparing POSITIVE and NEGATIVE C to the original MFLOPS value from homework 1 it is clear that POSITIVE and NEGATIVE C MFLOPS value is similar to that of homework 1.

Task2:

Below is the run time summary of the code for task 2. The optimization flags used is also shown. Note that the run time is given in seconds.

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Before changing the code

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ACI-I

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt

[spr5353@comp-ic-0022 hw2]$ ./vt

run time=44.6587

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt -O1

[spr5353@comp-ic-0022 hw2]$ ./vt

run time=8.33518

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt -O2

[spr5353@comp-ic-0022 hw2]$ ./vt

run time=8.34206

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt -O3

[spr5353@comp-ic-0022 hw2]$ ./vt

run time=8.35358

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt -O3 -mavx

[spr5353@comp-ic-0022 hw2]$ ./vt

run time=8.38088

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ACI-B

[spr5353@aci-lgn-002 ~]$ cd hw2

[spr5353@aci-lgn-002 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt

[spr5353@aci-lgn-002 hw2]$ ./vt

run time=26.3582

[spr5353@aci-lgn-002 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt -O1

[spr5353@aci-lgn-002 hw2]$ ./vt

run time=9.5333

[spr5353@aci-lgn-002 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt -O2

[spr5353@aci-lgn-002 hw2]$ ./vt

run time=9.47419

[spr5353@aci-lgn-002 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt -O3

[spr5353@aci-lgn-002 hw2]$ ./vt

run time=9.51142

[spr5353@aci-lgn-002 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o vt -O3 -mavx

[spr5353@aci-lgn-002 hw2]$ ./vt

run time=9.49674

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

After changing the code

change: loop blocking

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ACI-I

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2

[spr5353@comp-ic-0022 hw2]$ ./task2

run time=11.0107

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2 -O1

[spr5353@comp-ic-0022 hw2]$ ./task2

run time=3.43691

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2 -O2

[spr5353@comp-ic-0022 hw2]$ ./task2

run time=3.52911

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2 -O3

[spr5353@comp-ic-0022 hw2]$ ./task2

run time=3.28934

[spr5353@comp-ic-0022 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2 -O3 -mavx

[spr5353@comp-ic-0022 hw2]$ ./task2

run time=3.46361

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ACI-B

[spr5353@aci-lgn-006 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2

[spr5353@aci-lgn-006 hw2]$ ./task2

run time=9.13848

[spr5353@aci-lgn-006 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2 -O1

[spr5353@aci-lgn-006 hw2]$ ./task2

run time=5.67616

[spr5353@aci-lgn-006 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2 -O2

[spr5353@aci-lgn-006 hw2]$ ./task2

run time=5.83622

[spr5353@aci-lgn-006 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2 -O3

[spr5353@aci-lgn-006 hw2]$ ./task2

run time=6.02078

[spr5353@aci-lgn-006 hw2]$ g++ -pg hw2_task2_spr5353.cpp -o task2 -O3 -mavx

[spr5353@aci-lgn-006 hw2]$ ./task2

run time=5.82758

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

The performance of the code is faster on aci-b than aci-i before any flags are added. Even though I am not sure of the reason for this I feel it is an interesting observation to mention.

In the original code, each loop is traversed N times, as a result, the performance of the code is bounded by memory. In order to improve the performance of the code, the memory used needs to be reduced. The textbook talks about Loop blocking, it is essentially breaking down the problem and making use of cache which leads to a reduction in memory used. This is possible as instead of traversing every x value

for each y value, all the loops are divided using a block size. The way I decided my block size was by taking the square root of N. Doing so should divide the loop in half and since my run time before optimization was about 8 seconds and the target run time is 3 seconds which is just a little less than 8/2 =4 seconds I though it was best to take this block size. Using Loop blocking enabled the first block of the array X to be stored in cache while the other block was iterated. The results above show that using this method was effective. I did try different block sizes to see if there was still some room to speed up the program. The block size values I used were (square root of N)/2 , (square root of N)/4 , and even 10. But I did not get any significant performance improvements.

Note:There may be a block size that could be faster that I am yet to find

Citation:

Task2: Code for change is loop developed using page 82 of the textbook and DataOptimization slide from class.