Holly Straley
straleyh@oregonstate.edu
CS475 – Spring 2018
Project 3

False Sharing Fixes

1. Machine

I am on my home computer running my program on the flip3 server.

2. Performance Data

2.1 Notes About Data

2.1.1 Error in Calculations

For both fixes, performance was calculated as: someBigNumber / (time1 – time0) / 1000000 where someBigNumber was the iterations of the for loop doing the arithmetic and it was set to 1000000000 for all cases. After completing all data collection and while doing the data analysis, I realized that I probably should have calculated performance as: someBigNumber *** 4** / (time1 – time0) / 1000000. Because of this error and lack of time to recollect all data, I have denoted that the performance is megaAdds/sec/4 to account for the factor of 4 that all calculations are off by. I feel confident that this minor error did not effect the graph trends and I still have valuable data to analyze.

2.1.2 Accounting for Differing Variables

Fix #1 involved setting the padding (NUM) variable to 0 – 16, setting the number of threads to 1, 2, and 4, then calculating the performance with each set of variables. Fix #2 only involved calculating the performance with variable number of threads set to 1, 2, and 4. Because Fix #2 data did not have a padding variable, the data for each thread was extrapolated as a horizontal line for NUM 0 - 16.

2.2 Fix #1 Data

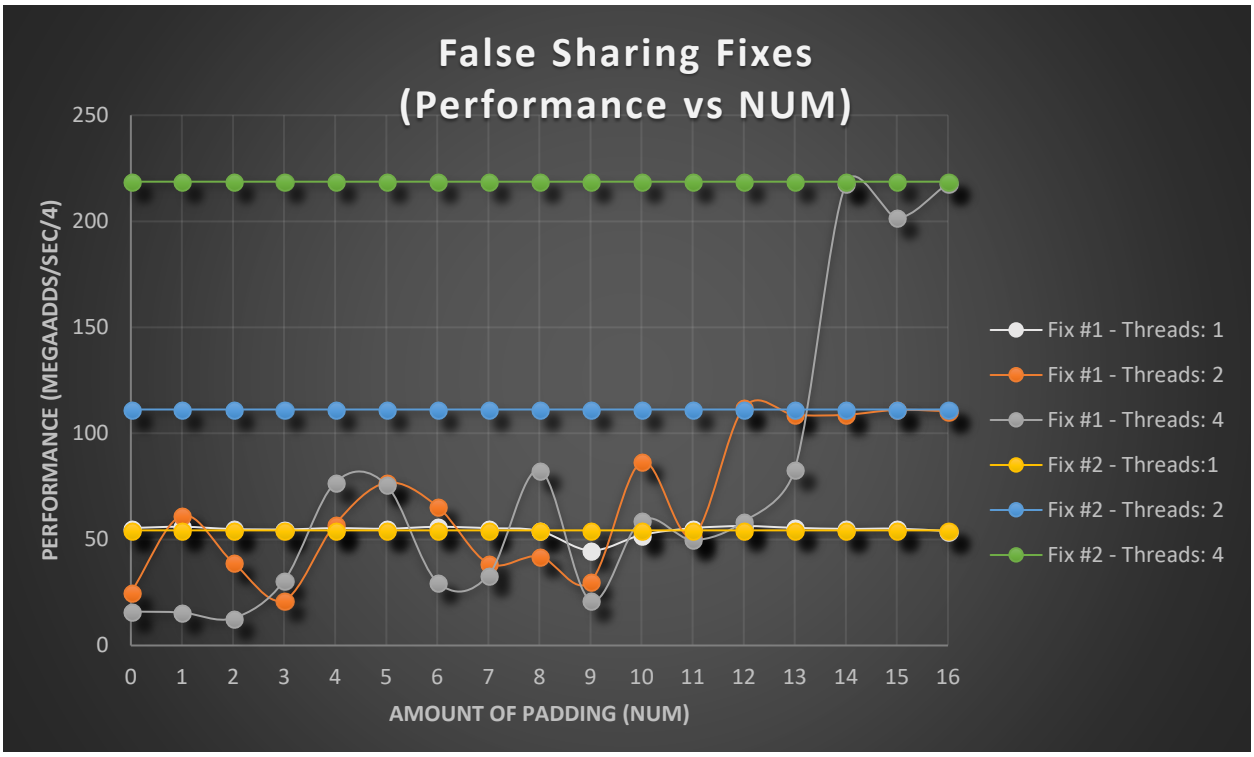
Number of Threads	Padding (NUM)	Performance (megaAdds/sec/4)
1	0	55.4002
1	1	55.9646
1	2	54.9034
1	3	54.7206
1	4	55.3322
1	5	55.0547
1	6	55.9397
1	7	55.3253
1	8	54.0082
1	9	44.785
1	10	52.0936
1	11	55.3031
1	12	56.3172
1	13	55.4714
1	14	54.9859
1	15	55.0774
1	16	53.9416
2	0	24.9728

2	1	61.1459
2	2	39.1831
2	3	21.3859
2	4	57.1147
2	5	76.6839
2	6	65.5975
2	7	38.6926
2	8	41.9783
2	9	30.1174
2	10	86.6142
2	11	51.582
2	12	112.1179
2	13	108.7577
2	14	108.7636
2	15	111.0985
2	16	110.4647
4	0	15.9979
4	1	15.526
4	2	12.8751
4	3	30.5115
4	4	76.776
4	5	75.7803
4	6	29.9468
4	7	33.0703
4	8	82.7565
4	9	21.1628
4	10	58.9976
4	11	49.7896
4	12	58.4625
4	13	83.055
4	14	217.7031
4	15	201.6483
4	16	217.7978

2.3 Fix #2 Data

Number of threads	Performance (megaAdds/sec/4)
1	54.2771
2	111.2767
4	218.6886

2.4 Combined Fixes Graph



3. Patterns

3.1 Fix #1

3.1.1 Fix #1 - Threads: 1

For one thread, the performance stayed consistently low (around or below 50 megaAdds/sec/4) for all NUMs.

3.1.2 Fix #1 - Threads: 2

For two threads, the performance bounced quite erratically within about 25 megaAdds/sec/4 of Fix #1 – Thread: 1 until it bumped up and stayed consistently near 110 megaAdds/sec/4 at NUM >= 12.

3.1.3 Fix #1 - Threads: 4

For four threads, the performance stayed significantly below that of Fix #1 – Threads:1 until NUM = 4 then it started bouncing around within about 30 megaAdds/sec/4 of Fix#1 – Threads:1. Performance finally made a big leap at NUM = 14 and stayed above 200.

3.2 Fix #2

3.2.1 Fix #2 - Threads: 1

This data is represented as a horizontal line at 54.2771 megaAdds/sec/4.

3.2.2 Fix #2 - Threads: 2

This data is represented as a horizontal line at 111.2767 megaAdds/sec/4.

3.2.3 Fix #2 - Threads: 4

This data is represented as a horizontal line at 218.6886 megaAdds/sec/4.

4. Analysis of Behavior

4.1 Fix #1

Overall, the data for Fix #1 follows the expected trend of thread counts 2 and 4 jumping when there was enough padding to allow each thread to have its own cache line. Though the data lines bounced around more than those shown in lecture, the trends of a major jump in performance exist. It is odd that my data has the performance jumps in different places than the lecture (2 threads jumps at NUM = 12 and 4 threads jumps at NUM = 14). I am not sure how to explain this other than it could be user error as I did each case by hand. Creating a script for testing would likely create more consistent results.

4.1.1 Fix #1 - Threads: 1

For one thread, the performance stayed consistently low (around or below 50 megaAdds/sec/4) for all NUMs which was expected. This can be thought of as the baseline for the data since it is the performance without multi-threading.

4.1.2 Fix #1 - Threads: 2

It is most notable the performance for two threads is often lower than performance for a single thread until NUM = 12. This is likely the point where there is enough padding to allow each thread to have its own cache line so that false sharing is not affecting performance.

4.1.3 Fix #1 - Threads: 4

For four threads, it is important to note that the performance was significantly lower than that of Fix #1 – Threads:1 for the at least half of all NUM until NUM = 14 was reached. The performance jumped around a lot but it seems that there is no point in using four threads unless each thread has its own cache line because the performance is not likely to be better than the baseline of one thread.

4.2 Fix #2

The data for Fix #2 followed expected trends, especially that each horizontal line nearly matched the maximum performance of the respective thread number for Fix #1.

4.2.1 Fix #2 - Threads: 1

This data follows the expected trend as it stays nearly identical to the line for Fix #1: Threads:1 to act as the baseline for the data.

4.2.2 Fix #2 - Threads: 2

This data follows the expected trend as it meets the line Fix #1 – Threads : 2 where that line jumps in performance at NUM = 12. At NUM = 12, the performance for both lines are nearly identical then continue to be so for all remaining NUMs.

4.2.3 Fix #2 - Threads: 4

This data follows the expected trend as it meets the line Fix #1 – Threads : 4 where that line jumps in performance at NUM = 14. At NUM = 14, the performance for both lines are nearly identical then continue to be so for all remaining NUMs.

4.3 Analysis Conclusion

Though there are some oddities in the data, overall, the data seems to follow the expected trends of:

1. Performance of Fix #1 improves greatly once each thread has its own cache line.
2. Performance of Fix #2 matches with the maximum performance of its partner thread from Fix #1.