Josh Huff, huffj@oregonstate.edu
Professor Bailey
CS 475 – Spring 2017
May 9, 2017

Project 3: Commentary

---

1. Tell what machine you ran this on

My main workhorse, a home-built machine running Ubuntu 16.04 LTS (64-bit). The processor is Intel® Core™ i5-4690K CPU @ 3.50GHz × 4 and it has 15.6 GiB of usable RAM.

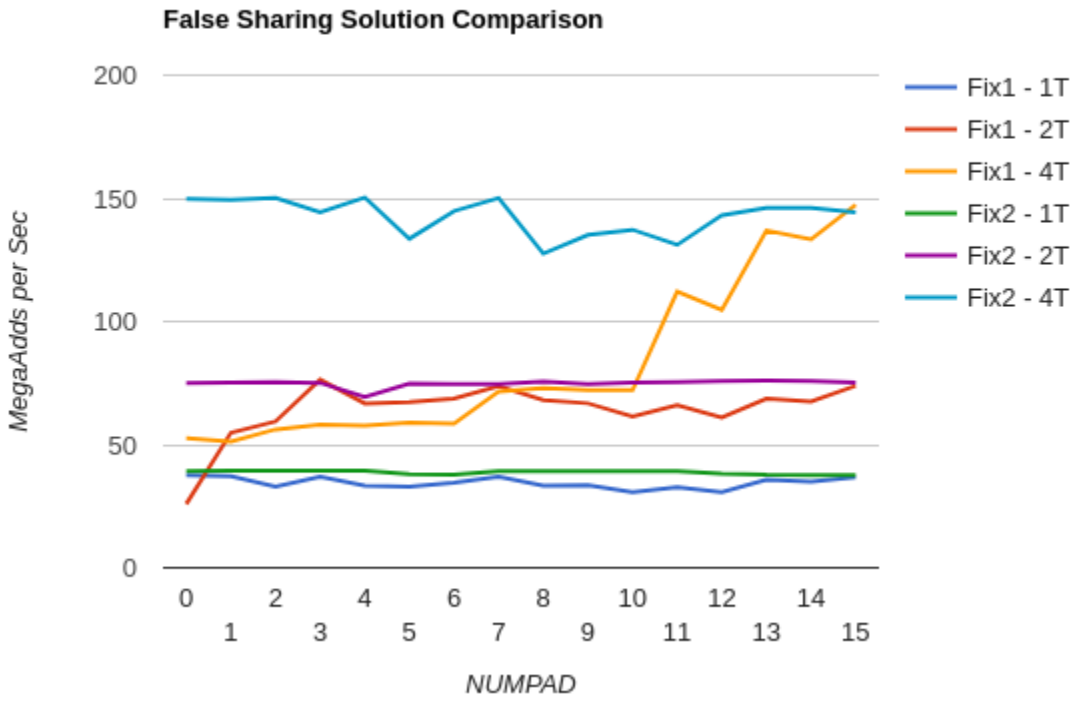
The difference between peak and average performance was minimal, and the load averages were very low (less than 1) across the board. However, the results from running the program locally were suspect and flip2 was used for this report. At the time, the load averages were in the low range of 3 to 6.

2. Create a table with your results

False Sharing Comparisons (in MegaAdds Per Second)

NUMPAD	FIX #1 – NUMPAD			FIX #2 – Private Variables		
	Number of Threads			Number of Threads		
	1	2	4	1	2	4
0	37.59	25.92	52.61	39.31	75.06	149.78
1	37.16	54.86	51.31	39.36	75.16	149.30
2	33.03	59.42	56.18	39.38	75.37	150.17
3	36.88	76.41	58.03	39.44	75.10	144.29
4	33.41	66.69	57.74	39.41	69.43	150.38
5	32.84	67.30	59.02	38.00	74.83	133.63
6	34.46	68.63	58.51	37.88	74.59	144.80
7	36.91	73.76	71.55	39.30	74.68	150.10
8	33.35	68.04	72.93	39.26	75.59	127.60
9	33.58	66.77	72.15	39.25	74.67	135.18
10	30.75	61.43	72.06	39.20	75.25	137.26
11	32.68	66.11	112.15	39.27	75.38	131.10
12	30.68	60.97	104.76	38.24	75.87	143.11
13	35.73	68.71	136.99	37.74	75.98	146.14
14	34.87	67.57	133.36	37.70	75.78	146.18
15	36.84	73.87	147.38	37.54	75.11	144.40

3. Draw a graph. The X axis will be the NUMPAD. The Y axis will be the performance.



4. What patterns are you seeing in the performance?

With one thread using the NUMPAD solution, the calculation speeds are fairly uniform. This is to be expected, because there is no parallelism. The average speed for this run was 34.42

With two threads using the NUMPAD solution, the speeds begin worse than the baseline before roughly doubling the one thread's performance. The average speed for this run was 64.15

With four threads using the NUMPAD solution, the speeds are sometimes similar to, or worse than, those reached with two threads, and only truly double its performance with a NUMPAD of 15. The average speed for this run was 82.30

With one thread using the private variable solution, the average performance is 38.77 and the calculation speeds are fairly consistent in the range between 37 and 39. This is to be expected, because the solution does not have the variation that NUMPAD does. For this baseline, the results are similar to those of the NUMPAD run.

With two threads using the private variable solution, the average speed is 74.87 MegaAdds/sec. It consistently outperforms the NUMPAD solution with two threads.

With four threads using the private variable solution, the speeds are almost all greater than those offered by the NUMPAD solution. Its average speed is 142.71

5. Why do you think it is behaving this way?

The NUMPAD solution is a bit “hacky” and uses memory in an extraneous fashion. This doesn’t necessarily explain the lackluster performance, though.

That difference can be accounted for by the manner in which the array is accessed. Using padding doesn’t prevent the program from having to look up the value of Array[i] on each iteration. On the other hand, looking up the value of Array[i] just once and putting it in an accumulator variable (then giving each thread its own copy of that private variable) reduces collisions and takes advantage of having multiple stacks.