Linge Ge - CS475
Project 3 - False Sharing

Overview: We explore fixes to false sharing by creating a program that creates a struct of a floating value and an array padding of variable elements. Depending the number of concurrent threads being used, the program creates that many arrays of the struct, and does a billion arithmetic addition (modification) on the same struct's floating value of each array element. Without sufficient padding, the structs' floating values line on the same cache line, and these modifications cause false sharing. We explore two fixes (padding and modifying temp variables instead) to fix this issue.

1. The code ran on:
    - Linux flip3.engr.oregonstate.edu 3.10.0-693.11.1.el7.x86_64 #1 SMP Mon Dec 4 23:52:40 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
    - 96GiB RAM Memory
    - 12 (hyperthreading x2) CPUs @ Intel(R) Xeon(R) CPU X5650 @ 2.67GHz

2.

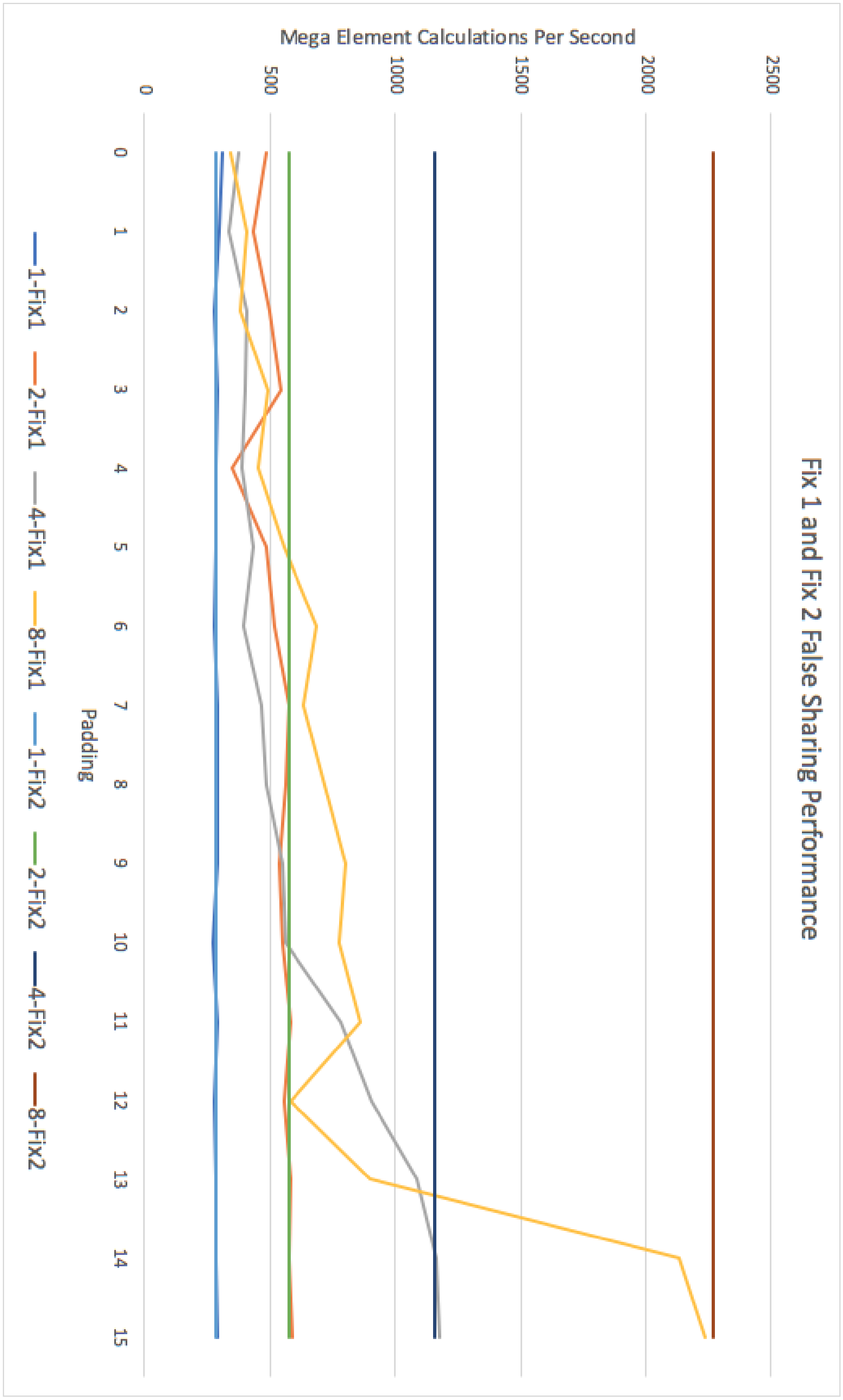
Sharing Fix #	Thread Count	Padding (Fix #1)	megaElementsCalculated/s
2	1	-	289.246
2	2	-	577.943
2	4	-	1157.07
2	8	-	2269.23
1	1	0	312.472
1	1	1	301.122
1	1	2	281.726
1	1	3	295.871
1	1	4	283.124
1	1	5	288.904
1	1	6	280.959
1	1	7	290.89

1	1	8	293.016
1	1	9	290.094
1	1	10	275.982
1	1	11	294.45
1	1	12	278.079
1	1	13	288.401
1	1	14	286.525
1	1	15	293.349
1	2	0	488.088
1	2	1	432.678
1	2	2	498.644
1	2	3	543.906
1	2	4	349.652
1	2	5	483.807
1	2	6	519.055
1	2	7	576.526
1	2	8	565.985
1	2	9	539.715
1	2	10	552.049
1	2	11	581.632
1	2	12	558.271
1	2	13	583.063
1	2	14	574.688
1	2	15	586.872

1	4	0	378.798
1	4	1	334.771
1	4	2	407.456
1	4	3	399.505
1	4	4	390.965
1	4	5	436.96
1	4	6	396.473
1	4	7	468.407
1	4	8	485.245
1	4	9	549.293
1	4	10	561.64
1	4	11	786.264
1	4	12	903.881
1	4	13	1084.37
1	4	14	1161.04
1	4	15	1179.41
1	8	0	346.216
1	8	1	411.916
1	8	2	380.142
1	8	3	495.115
1	8	4	457.299
1	8	5	556.212
1	8	6	684.453
1	8	7	632.899

1	8	8	716.48
1	8	9	805.189
1	8	10	777.459
1	8	11	863.1
1	8	12	586.326
1	8	13	896.474
1	8	14	2130.55
1	8	15	2236.94

3.


Fix 1 and Fix 2 False Sharing Performance
Mega Element Calculations Per Second
0
500
1000
1500
2000
2500
Padding
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
1-Fix1
2-Fix1
4-Fix1
8-Fix1
1-Fix2
2-Fix2
4-Fix2
8-Fix2

4.

- Despite false sharing, there is still a performance increase of having multiple cores doing the work, even if the speedup is not great.
- For Fix #1, the performance speedup is abysmal with little padding. However for each thread count, the performance slowly grows better with more padding until it is close to the performance seen with Fix #2.
    i. 2-thread Fix #1 reaches the same performance as Fix #2 at around 6-7 padding.
    ii. 4-thread and 8-thread Fix #1 gradually increases in performance from padding 0 to 10-12, and then from there dramatically increase in performance until it matches with Fix #2.
- For Fix #2, where we use a temporary variable to modify, the padding does not matter (in fact in the code we get rid of it), and thus the speedup is consistent.
- For Fix #2, there is a good multiplier in the speedup: having 2 cores is almost twice as fast; having 4 cores is about 4x as fast; having 8 cores is almost 8x as fast as compared to a single core performance.

5.

- As discussed per lecture, we know Threads 2, 4 and 8 in Fix #1 is behaving this way because of conflicting cache lines which is causing false sharing. As multiple threads are bringing the same surrounding data to its cache line and making modifications on that range of data in its cache, other threads must reload and update their cache line after each change. This causes false sharing and at this point, each thread might as well be fetching the data and updating it in memory, since the data is changing so quickly by different threads.
- However, as the padding increases, the floating values in which each thread must update are farther apart in memory. Eventually, they are far enough in contiguous memory where they are brought to separate cache lines, and the threads modifying their data will run into less conflicts and false sharing. This increases performance as the amount of reloading and fetching from memory decreases.
- Eventually, as seen with padding 15 for all multicore performance, all the floating values of different structs are on different cache lines. At this point, there is no conflict when each thread updates their own floating value, and hence performance is greatly increased as it takes full advantage of the cache properly.
- For Fix #2, we get the best performance right away because there is no false sharing. Separate thread has their own local variable stack, which exist in different memory addresses far away from each other. Thus there is no issues with having shared data on a caching line, and each thread is able to update their own local variable floating value data without sharing or conflicts. We thus take advantage of the cache and get the best performance possible from multicore threading.