Jessica Spokoyny
May 8, 2017
Project #3
False Sharing

1. <u>My Machine:</u>
   I ran this program on bash in linux from a windows 10 laptop. My main function is contained in a file called project3.cpp
   I compiled and executed the program for fix 1 with a script by typing:

```
        for t in 1 2 4
        > do
        > echo NUMT = $t
        > for p in 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
        > do
        > echo NUMPAD = $p
        > g++ -DNUMT=$t -DNUMPAD=$p -DFIX=1 project3.cpp -o proj3 -lm -fopenmp
        > ./proj3
        > done
        > done
```

   Then I compiled and executed the program again for fix 2 by typing:

```
        for t in 1 2 4
        > do
        > echo NUMT = $t
        > g++ -DNUMT=$t -DNUMPAD=0 -DFIX=2 project3.cpp -o proj3 -lm -fopenmp
        > ./proj3
        > done
```

2. <u>My performance results:</u>

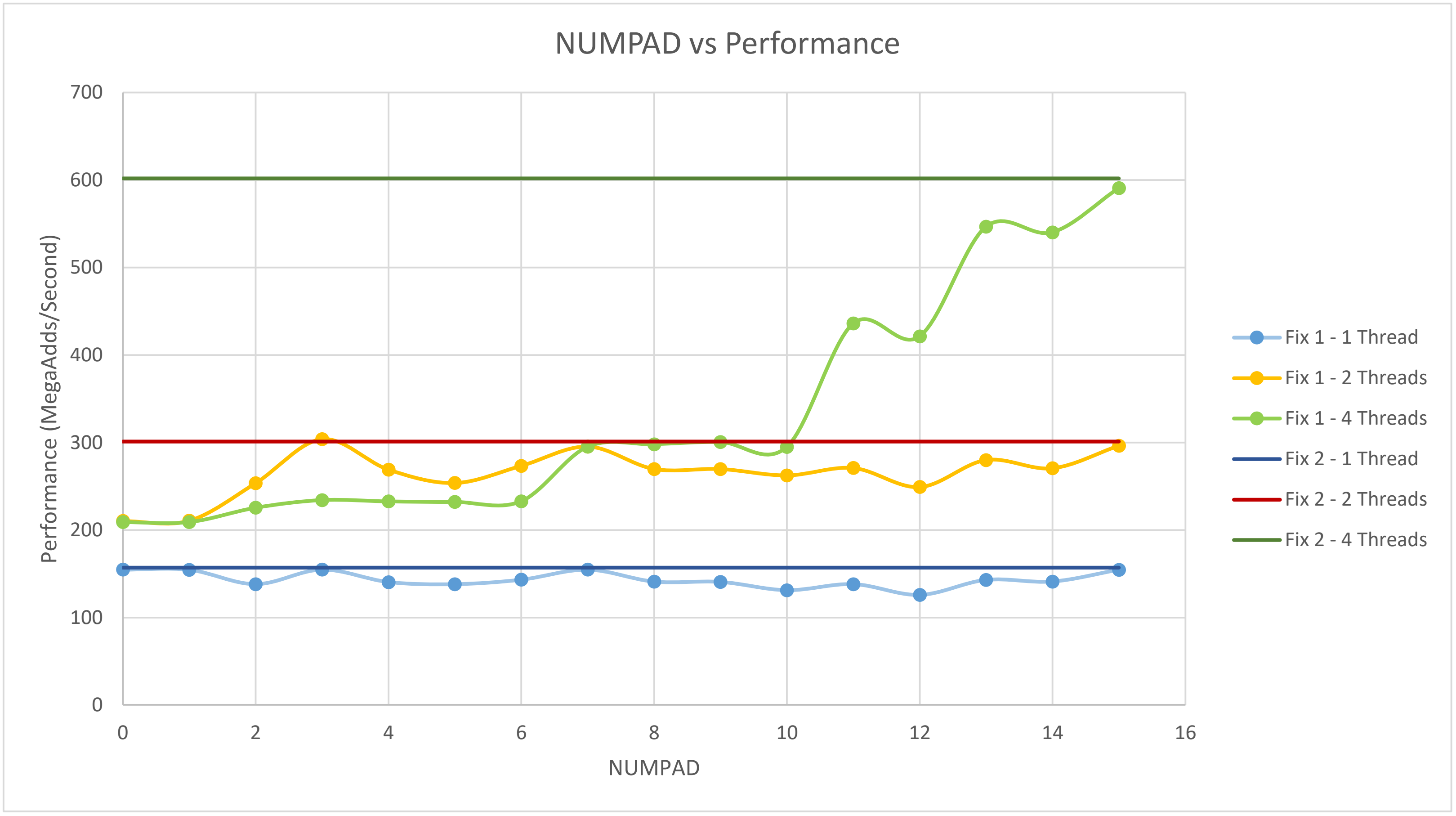
Fix #1:

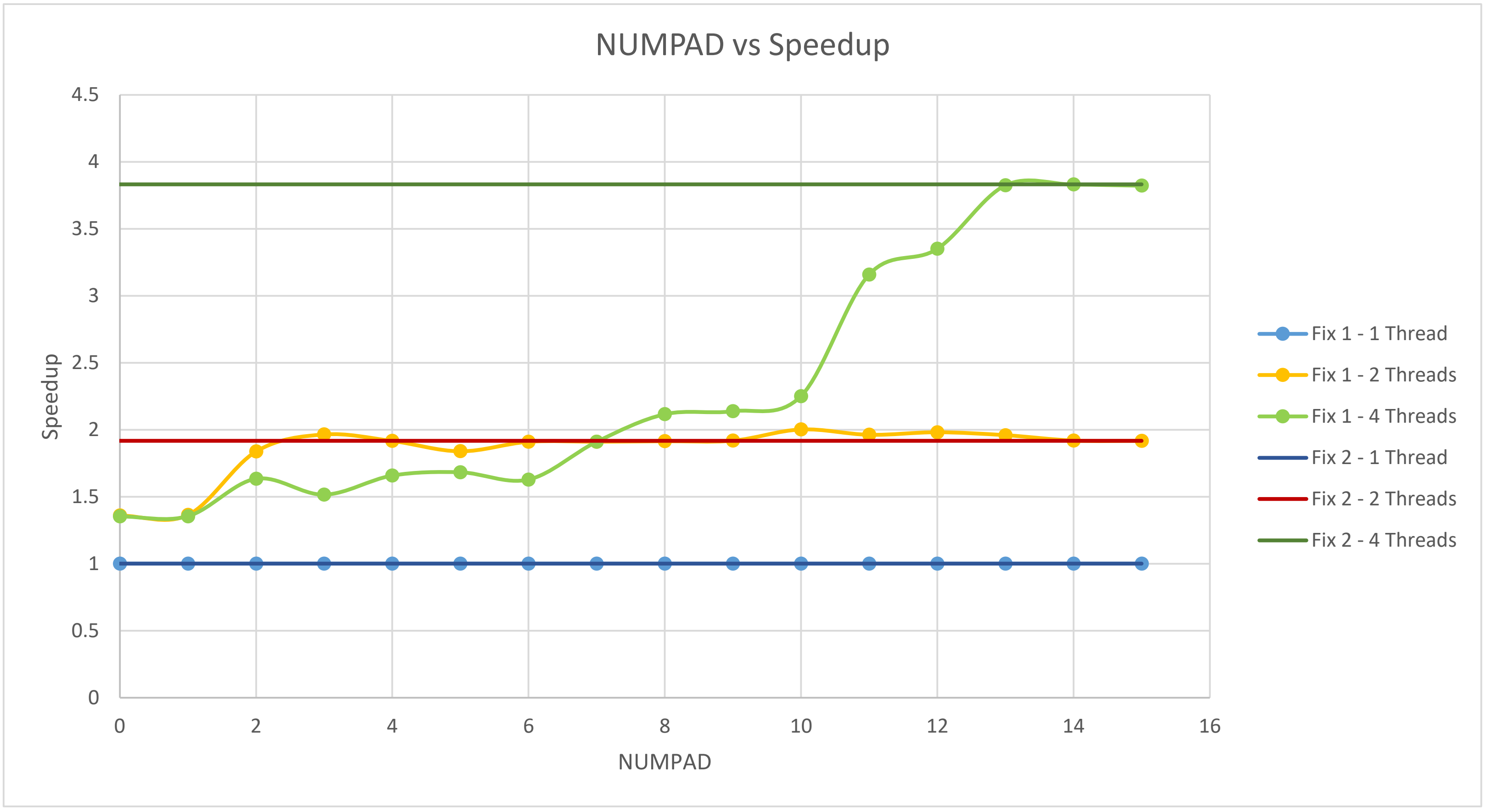
| | 1 thread | 2 threads | 4 threads |
|---|---|---|---|
| **0** | 154.69 | 210.31 | 209.1 |
| **1** | 154.54 | 210.74 | 209.15 |
| **2** | 138.1 | 253.62 | 225.52 |
| **3** | 154.7 | 303.81 | 234.27 |
| **4** | 140.5 | 269.12 | 232.77 |
| **5** | 138.06 | 253.81 | 232.14 |
| **6** | 143.16 | 273.36 | 232.94 |
| **7** | 154.67 | 295.4 | 295.27 |
| **8** | 140.98 | 269.81 | 298.13 |
| **9** | 140.66 | 269.75 | 300.62 |
| **10** | 131.17 | 262.5 | 295.13 |
| **11** | 138.11 | 270.89 | 436.11 |
| **12** | 125.81 | 249.16 | 421.42 |
| **13** | 142.96 | 280.01 | 546.66 |
| **14** | 141.09 | 270.66 | 540.32 |
| **15** | 154.65 | 296.41 | 590.95 |

Fix #2:

| 1 thread | 2 threads | 4 threads |
|---|---|---|
| 157.08 | 301.06 | 601.87 |

3. <u>Graph of Performance:</u>





4. Performance Patterns I Noticed:

   The performance values I observed were similar to those discussed in the lecture notes.
   Using fix #1 to pad the array:

   - 1 thread: the values are consistently just under 150 MegaAdds/second for all values of NUMPAD
   - 2 threads: the values increase sharply, then stay pretty consistent just under 300 MegaAdds/second for the remaining values of NUMPAD
   - 4 threads: the performance values jump up when NUMPAD = 7, and 15 and should stay constant after that at around 600 MegaAdds/second

   Using fix #2 to solve the false sharing problem produces performance values of 157, 301, and 601 for thread counts of 1, 2, and 4 respectively.

   I did notice that my performance values for both 2 threads and 4 threads never went below the values for 1 thread which show that false sharing didn't have as dramatic of an effect on my data. Also, after NUMPAD = 7, my performance using 4 threads rose consistently while I expected it to jump up at 10 and then at 15.

5. Explanation of Behavior:

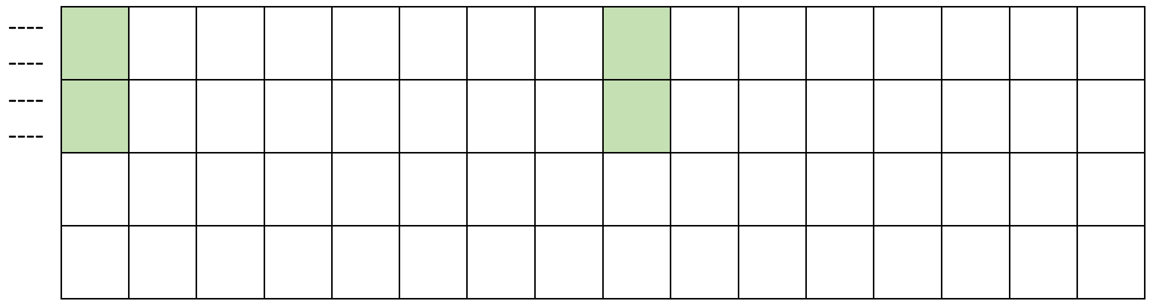
   To get a clear picture of how these changes are affecting performance, I found it useful to look at a graph of NUMPAD vs Speedup.

   When we only have 1 thread available, there exists no false sharing because only 1 thread is accessing a thread line at any given time. This results in a speedup of 1 for both fix #1 and fix #2 as expected.

   When we have 2 threads, we would expect the speedup to nearly double and the values do reflect this. Using fix #2, our speedup is 1.92 which is a good result. Because of false sharing, the speedups for low values of NUMPAD are smaller than we would like because we have both cores banging on the same thread line. Eventually, the speedup for fix #1 also reaches and stabilizes around 2.

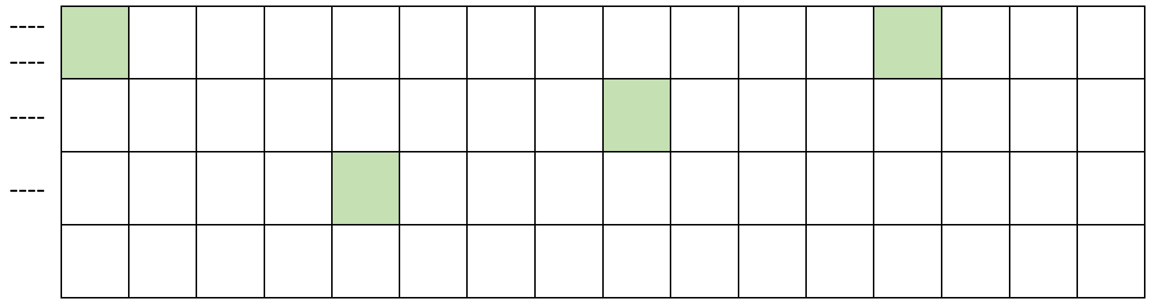
   When we increase the number of threads to 4, we would expect the speedup to almost quadruple however, our values take some time to get there. With fix #2, we have a consistent speedup of 3.83 which is expected as it doesn't rely on NUMPAD. However, for fix #1, our speedup is actually lower with 4 threads than it is with 2 threads until NUMPAD reaches 7. This is because for values of NUMPAD < 7, we have all 3-4 cores banging on the same thread line (whereas with 2 cores, there could only have ever been 2 cores banging on the same thread line). This causes performance to suffer until NUMPAD = 7 where we have 2 cores each banging on 2 thread lines:

NUMPAD = 7



There is a big jump at NUMPAD = 11 where we have 2 cores banging on 1 thread line and the other thread lines each only have 1 core (although I would have expected this increase in performance to occur at NUMPAD = 10):

NUMPAD = 11



Then, speedup keeps increasing until we finally get a speedup of 3.82 (which is the maximum speedup we can achieve) at NUMPAD = 15. This is because each core is banging on a different cache line:

NUMPAD = 15