Drake Seifert

CS475 Project 3

**Tell what machine you ran this on**

All tests were ran on OSU's FLIP servers. Each individual test was ran a certain number of times, and the result was only saved if the average number of mega-adds per second didn't deviate too far from the maximum mega-adds per second test value.

**Create a table with your results.**

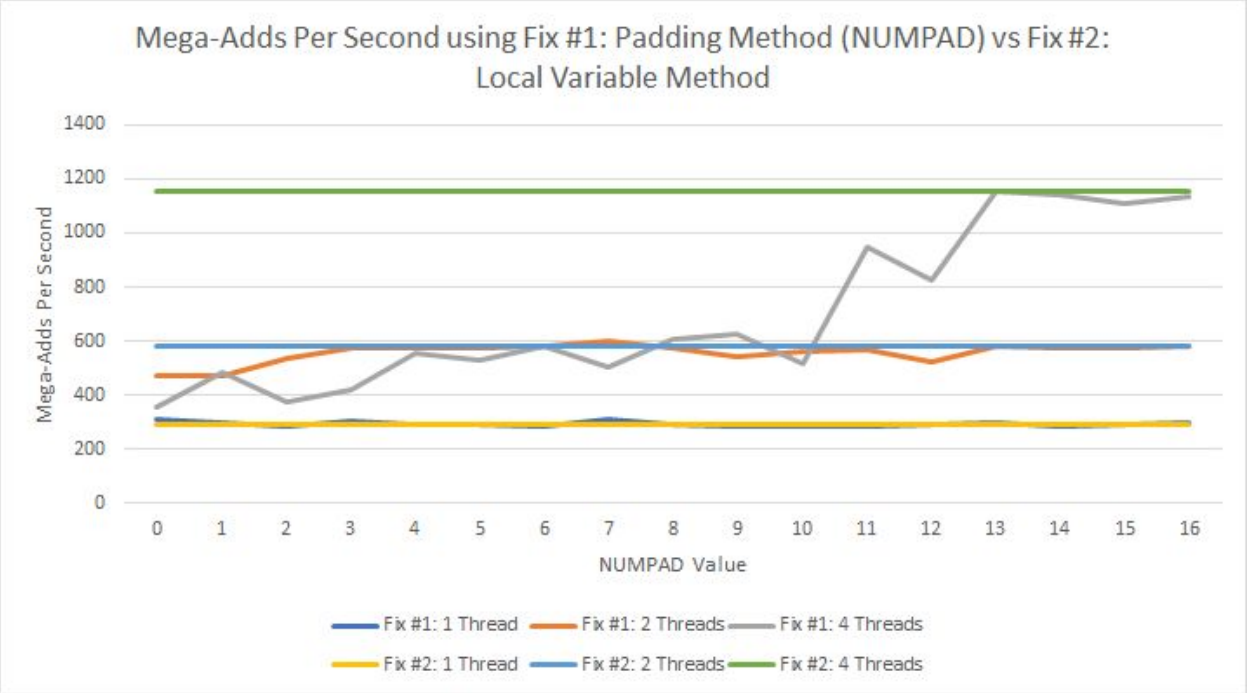
Rows: Method and Number of threads
Columns: NUMPAD value

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Fix #1: 1 Thread | 310.14 | 297.94 | 282.57 | 302.54 | 290.04 | 291.17 | 282.74 | 309.97 |
| Fix #1: 2 Threads | 470.61 | 473.45 | 538.26 | 576.6 | 577.26 | 575.34 | 581.03 | 599.13 |
| Fix #1: 4 Threads | 354.47 | 486.53 | 375.8 | 421.29 | 552.22 | 531.17 | 583.43 | 504.34 |
| Fix #2: 1 Thread | 291.21 | 291.21 | 291.21 | 291.21 | 291.21 | 291.21 | 291.21 | 291.21 |
| Fix #2: 2 Threads | 579.78 | 579.78 | 579.78 | 579.78 | 579.78 | 579.78 | 579.78 | 579.78 |
| Fix #2: 4 Threads | 1152.44 | 1152.44 | 1152.44 | 1152.44 | 1152.44 | 1152.44 | 1152.44 | 1152.44 |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|
| 289.6 | 287.04 | 283.43 | 283.3 | 292.56 | 296.5 | 285.5 | 288.63 | 294.98 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 574.08 | 544.57 | 561.72 | 567.63 | 520.55 | 582.86 | 575.52 | 574.53 | 578.42 |
| 606.76 | 628.79 | 519.56 | 947.51 | 827.04 | 1155.4 | 1143.61 | 1106.84 | 1136.87 |
| 291.21 | 291.21 | 291.21 | 291.21 | 291.21 | 291.21 | 291.21 | 291.21 | 291.21 |
| 579.78 | 579.78 | 579.78 | 579.78 | 579.78 | 579.78 | 579.78 | 579.78 | 579.78 |
| 1152.44 | 1152.44 | 1152.44 | 1152.44 | 1152.44 | 1152.44 | 1152.44 | 1152.44 | 1152.44 |

**Draw a graph. The X axis will be NUM, i.e., the amount of integers used to pad the structure. The Y axis will be the performance in whatever units you sensibly choose. There should be at least 6 curves shown together on those axes:**
**1-3: Using padding with 1, 2, and 4 threads.**
**4-6: Using a private variable with 1, 2, and 4 threads.**




**What patterns are you seeing in the performance?**

As predicted in the cache notes, there is a great increase in performance using fix #1 with 4 threads as the array padding increases. Unlike the notes, the performance for 2 threads stayed relatively consistent throughout testing. Fix #2 Seemed to work just as well as the best performing methods measured from all tests in Fix #1.

**Why do you think it is behaving this way?**

Considering Fix #1, the NUMPAD value greatly determines the number of times that each thread contains an invalid number in cache due to another thread having updated it recently. This causes a huge dip in performance as can be seen in the lower values of NUMPAD for 4 threads. If this fix was not addressed, the 4 thread case could be 4 times slower than otherwise, which on a large scale could be absolutely detrimental to the performance of a system. By adding values to NUMPAD, the space between cache values increases, and therefore there is less likely to be cache misses. The performance is optimized in the 4 thread case when NUMPAD is 15. This is because that each thread is then forced onto separate cache lines, and therefore don't have any conflicts. This same behavior can be seen from fix #2 for the same reasons; each thread isn't intruding with other threads' data in the "j" for loop. This drastically decreases the amount of invalid data that has to be updated in each threads' cache, and therefore maximizes the performance gains. This explains why the best case scenarios in fix #1 are nearly equivalent to those in fix #2. However, fix #2 is most likely preferable in a real-world scenario as it is much less memory intensive than the method suggested by fix #1, and works just as well.