Project #3
OpenMP: False Sharing

Haoxiang Wang; Student ID: 932359049

1. Machine Running the Test

I used school's server "flip" to run the test. At the time my test is running, the "uptime" is around 3.

2. Performance Results

I used Benchmarks to let the program running with different number of threads and pad numbers. The number of thread I used for the test are 1, 2 and 4. The pad numbers are from 0 to 16. Since the problem size is set to "someBigNumber = 1000000000" each outer loop, the total problem size is 4 * 1000000000 = 4000000000. The performance is evaluated by calculating units computed per second. Which is dividing the total problem size by total runtime.

The tests are run with two different fix methods. The first fix method is adding some size of pad to let different threads work on different cache lines. The second fix method is to use local variables, instead of contiguous array locations. This will spread our writes out in memory, and to different cache lines. The fix #1 method is done with different pad numbers, the fix #2 has no pad number issue, but I used the same script to calculate performance several times (number of pads) under the same condition and calculate the average of them. The following form is the result I got from the test.

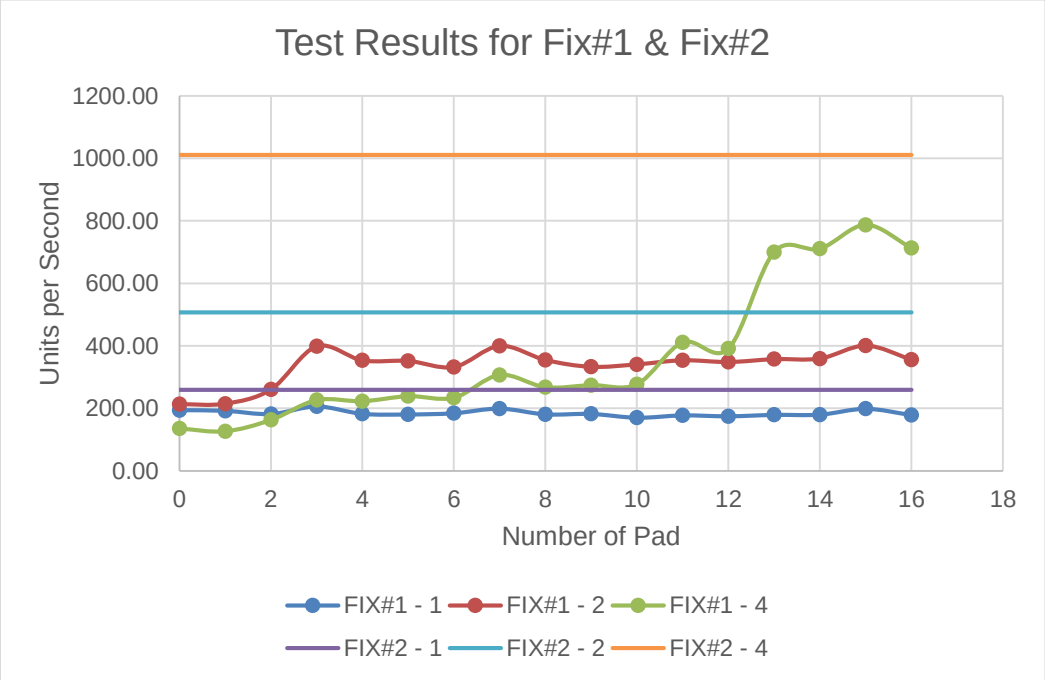
The form is too long to show in one complete one, so I separate it to two parts:

	0	1	2	3	4	5	6	7
FIX#1 - 1	194.63	192.52	182.17	206.35	182.75	180.82	184.69	199.50
FIX#1 - 2	213.34	214.98	260.43	398.55	353.72	351.96	332.48	399.99
FIX#1 - 4	135.82	126.91	163.09	226.66	223.09	239.18	234.26	307.11
FIX#2 - 1	259.51	259.51	259.51	259.51	259.51	259.51	259.51	259.51
FIX#2 - 2	506.59	506.59	506.59	506.59	506.59	506.59	506.59	506.59
FIX#2 - 4	1011.22	1011.22	1011.22	1011.22	1011.22	1011.22	1011.22	1011.22

…

8	9	10	11	12	13	14	15	16
181.05	182.57	170.37	177.76	175.01	179.43	180.18	198.92	178.94
355.01	333.10	340.94	354.05	348.46	357.56	359.51	400.87	356.20
267.93	274.49	277.47	411.33	391.47	700.55	710.97	787.40	713.18
259.51	259.51	259.51	259.51	259.51	259.51	259.51	259.51	259.51
506.59	506.59	506.59	506.59	506.59	506.59	506.59	506.59	506.59
1011.22	1011.22	1011.22	1011.22	1011.22	1011.22	1011.22	1011.22	1011.22

Results show in graph:



3. Behavior Explanation

The results shown in the graph is not as good as I expected. The three curves with nodes stand for results from Fix#1, the three straight lines without nodes stand for results from Fix#2. Since the results of Fix#2 are the average results of several times of running, there is not much discussion about it. As for the Fix#1, three curves stands for different threads' results. The result for using 1 thread turns out close to a straight line. This makes sense since no parallel is involved so that no false sharing it has. The cases for using 2 threads and 4 threads are not as good as expected. The result of using 2 threads shows that the performance increased after the number of pads is larger than 3, and the case of using 4 threads shows that the performance increased after the number of pads larger than 3 and 13. Also, the Fix#2's results are always better than Fix#1, they don't match perfectly.

The reason for these wired situations is due to the machine I ran programs on in my opinion. I ran the programs on flip, and at the time I run my program, some other people are also using the machine to run their programs. The same cache line might be used by others at the time I used it. If the cache line is not completely empty, my results of Fix#1 method could be strongly affected. At least the expected "performance increased points" are no longer reliable. However Fix#2 method uses local variables, which it won't be affected as badly as Fix#1 does, so its performance could be better than Fix#1.