CS 475 Parallel Computing
Instructor: Matthew Meyn
Student: Khuong Luu (luukh)

Project 3

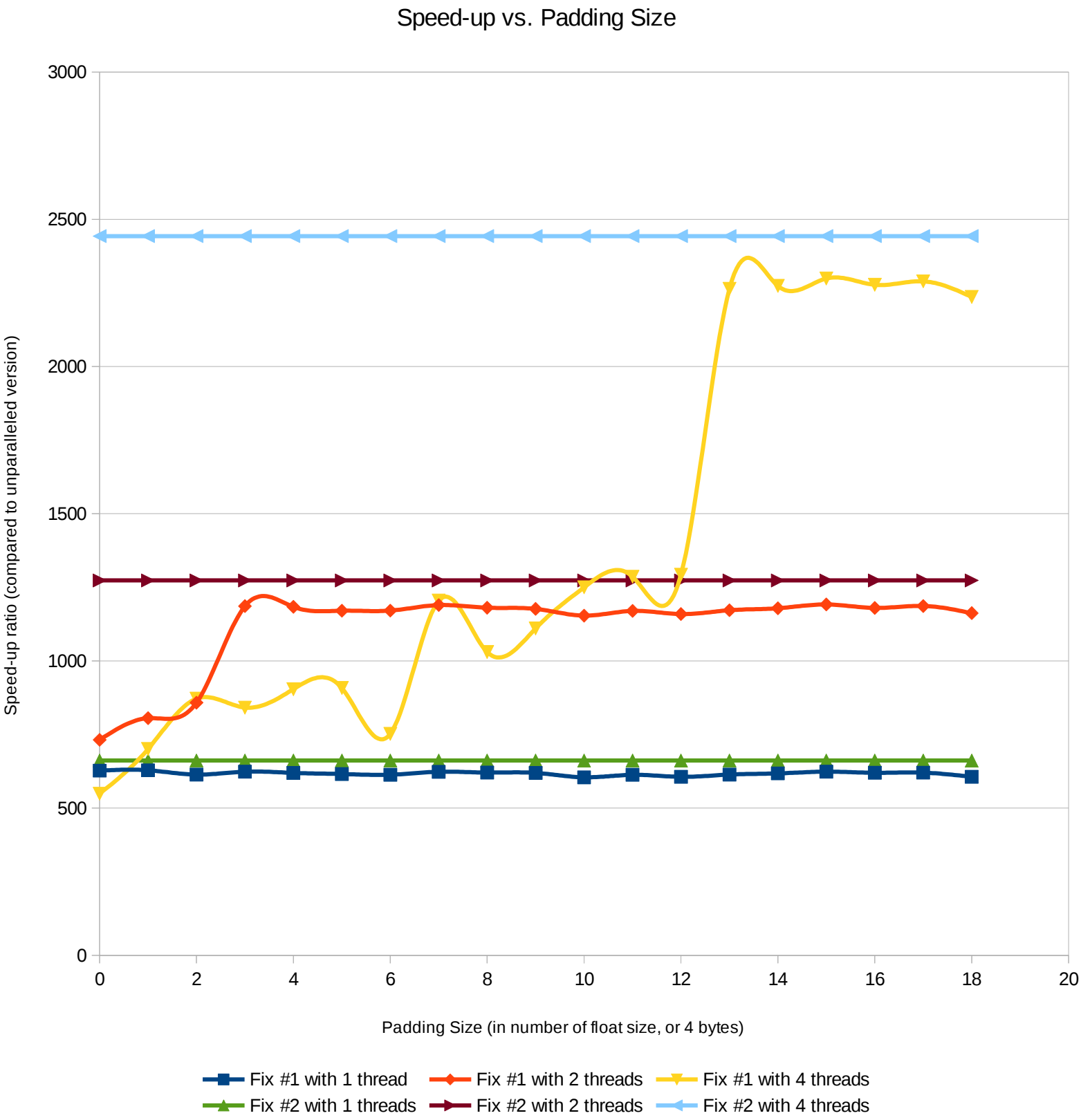
1. **Tell what machine you ran this on**
The computer I ran this experiment on has:
- Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
- Memory: 16GB, 2400Mhz
- Operating System: Ubuntu 16.04
- Kernel: 4.13.0-37-generic
- **Number of Cores: 4**
- **Cache line size: 64 bytes**
- **Compiler: gcc version 5.4.0 20160609, OpenMP 3.1**

2. Results

Performance unit is in **Speed-up ratio (Compared to unparalleled version)**
Padding Size unit is in **4 bytes (a size of a float)**

Padding Size	Fix #1 with 1 thread	Fix #1 with 2 threads	Fix #1 with 4 threads	Fix #2 with 1 threads	Fix #2 with 2 threads	Fix #2 with 4 threads
0	627.568	732.114	549.216	661.835	1273.4	2442.71
1	628.702	805.608	700.858	661.835	1273.4	2442.71
2	614.035	858.263	872.135	661.835	1273.4	2442.71
3	623.881	1185.95	841.587	661.835	1273.4	2442.71
4	619.721	1183.89	904.095	661.835	1273.4	2442.71
5	616.063	1170.59	908.006	661.835	1273.4	2442.71
6	613.632	1171.02	752.353	661.835	1273.4	2442.71
7	623.546	1189.74	1205.04	661.835	1273.4	2442.71
8	620.892	1180.72	1030.21	661.835	1273.4	2442.71
9	619.708	1176.79	1110.39	661.835	1273.4	2442.71
10	604.818	1153.98	1249.86	661.835	1273.4	2442.71
11	613.146	1169.9	1286.08	661.835	1273.4	2442.71
12	606.54	1159.19	1292.94	661.835	1273.4	2442.71
13	614.116	1172.17	2263.78	661.835	1273.4	2442.71
14	618.046	1178.88	2273.67	661.835	1273.4	2442.71
15	623.891	1191.83	2298.98	661.835	1273.4	2442.71
16	620.249	1180.08	2277.56	661.835	1273.4	2442.71
17	620.549	1186.6	2289.22	661.835	1273.4	2442.71
18	606.325	1162.17	2236.41	661.835	1273.4	2442.71

3. Graph


Speed-up vs. Padding Size
Speed-up ratio (compared to unparalleled version)
3000
2500
2000
1500
1000
500
0
0
2
4
6
8
10
12
14
16
18
20
Padding Size (in number of float size, or 4 bytes)
Fix #1 with 1 thread
Fix #1 with 2 threads
Fix #1 with 4 threads
Fix #2 with 1 threads
Fix #2 with 2 threads
Fix #2 with 4 threads

4. **Patterns and Explanations**

4.1. **Fix #2 performance pattern**

Pattern: Fix #2 performance pattern is independent of padding size and shows clear speed-up ratios respective to the number of threads used.

Explanation: This is because it doesn't use padding as its remedy to fix the cache false sharing issue. Instead, it fix the issue by reduce the number of direct read-write operations from/to cache line in each thread by using private variable (in this case is *tmp*). Thus, for each thread, there are only 1 direct read and 1 direct write from/to the cache line; and therefore, the cache line didn't have to be reloaded/revalidated repeatedly from memory which make the performance much better than fix #1 (when the padding size isn't large enough yet) or unfixed version.

4.2. **Fix #1 performance pattern**

- **(1) Pattern:** At the padding size of **0**, Fix #1-2 threads and Fix #1-4 threads is just as good as the performance baseline for 1-thread (unparalleled).
  Explanation: Because there's no padding, all threads read and write directly from/to the same cache line constantly, which makes the cache being reloaded/revalidated constantly from memory and thus suffer from the added latency for data to constantly move between CPU and memory and constantly make the computation (of CPU) wait for those latency. These negative effect of false-sharing is so much in this case that it out weights the benefit of both 2-threads and 4-threads parallelization, making their performance being just as good as the performance of 1-thread (unparalleled)

- **(2) Pattern:** Starting at the padding size of **3** and padding size of **7**, Fix #1-2 threads and Fix #1-4 threads, respectively, starts performing as good as the performance baseline for 2 threads. Starting at padding size of **13**, Fix #1-4 threads' starts performing as good as the performance baseline for 4 threads from then on.
  Explanation: Continue from the explanation of pattern (1), now as the padding size increases larger and larger, the threads experience less and less false sharing, which make their performance (or speed-up ratio) go up. Once the Fix#1-2 threads no longer experience false sharing, it performance as expected – as good as the performance baseline for 2 threads from then on. However, the Fix#1-4 threads still experience this issue since there are more threads thus the large enough padding size thresholds to eliminate the false-sharing is higher, which is in this case is padding size of **13** (of 4 bytes unit).

- **(3) Pattern**: The milestones of padding size of **3**, **7**, and **13** are several units smaller than those milestones expected in the lecture notes.
- **Explanation**: A possible explanation for this pattern is because my program didn't allocate the array element at the beginning of the cache line but a little off to the right. Thus, it takes smaller padding sizes for the data to be pushed and spread onto the other cache lines; and so the threshold is smaller than the case in which the first element of array is allocated at the beginning of the cache line