Nick Pepperling
4/21/2014
CS475

HW3: False Sharing Trials

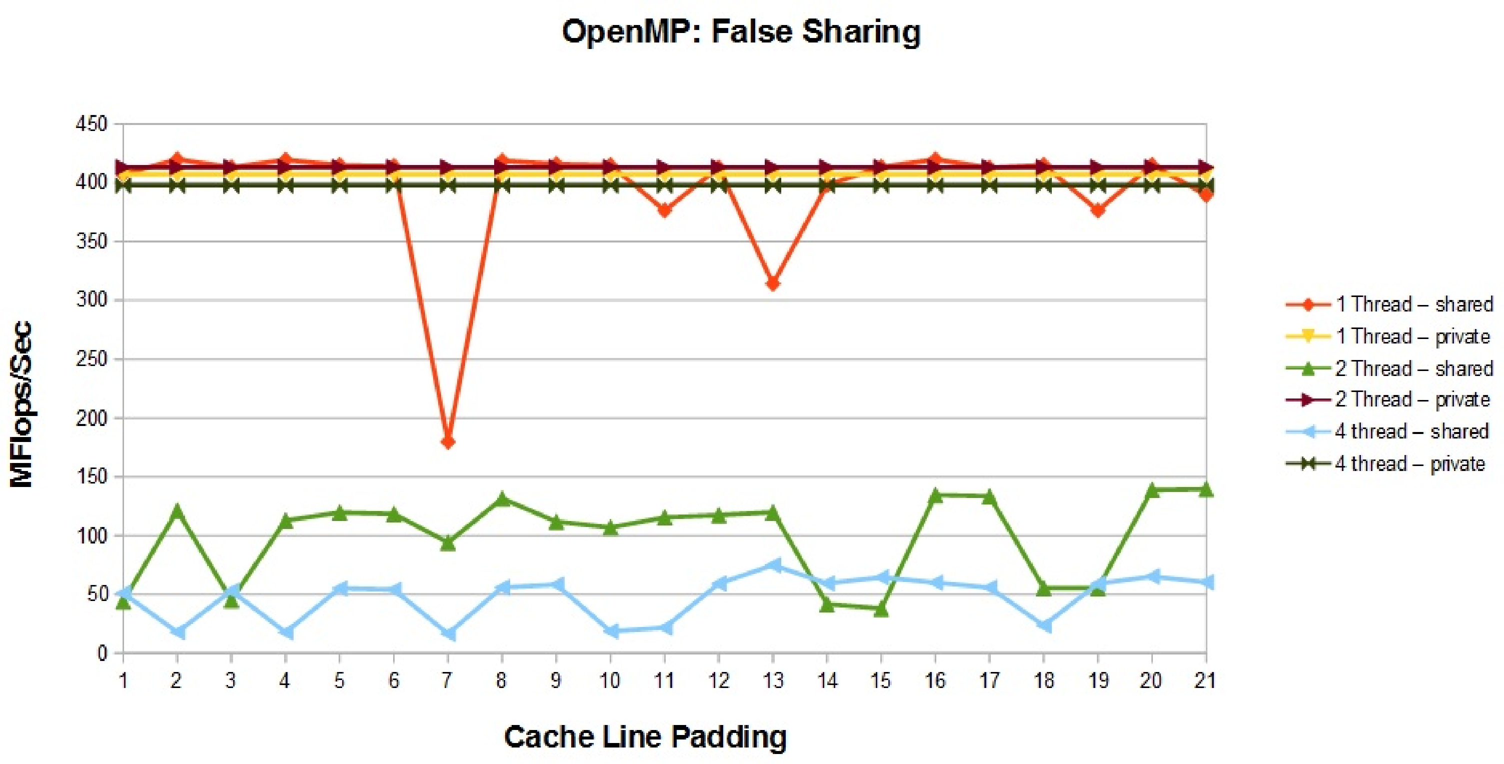
1.) Machine Configuration

I ran this on the machine intel gave us for our senior design project. It contains:

- Intel Xeon E5-2690
- 8-Cores @ 2.90GhZ
- Hyperthreaded (16 total cores)
- 20Mb CPU cache
- 64 GB RAM
- SSD

2.) Table Data: See back pages

3.) Graph:



4. and 5.) First of all, I would like to point out that my results for these trials were in some instances what I expected, but also had many anomalies (huge and uncharacteristic performance hit) that seem like they may require a more in-depth analysis. For now, this may be out of my realm of understanding since the machine I used has a specialized chip for high performance computing.

Anyhow, as far as patterns there are some somewhat noticeable ones, but not necessarily night and day. First of all each of the thread number values (1,2,4) for the shared trials (2nd fix) hover around the same value.

> **WHY?** This is understandable since the outer for loop had NUMTHREADS as a conditional check. Thus the problem size increases proportional to the number of threads, and roughly the same computation will be done over any unit of time. To calculate MFlops/sec I used the following:
>
> $$\frac{NUMTHREADS * SomeBigNumber}{(EndTime - StartTime)/1000000}$$
>
> As you can seen, Increasing NUMTHREADS increased the number of calculation needed and the amount of time needed happened to follow suit.

The second noticeable pattern are the valley's (lowest performance) and can be seen around padding numbes ~7, ~14, and ~19. Here we see a major dip in performance.

> **WHY?** This is likely due to the fact that at this value of padding, multiple cores are trying to access values on the same cache line. These attempts are happening between writes to Array[i] by other threads, thus invalidations are happening causing more frequent trips to and from memory. It is hard to say why the dips are so drastic with one thread, since it is the only thread reading to and writing from that cache line so it should not encounter invalid data. For clarification, I ran it multiple times and got roughly the same results

The third noticeable pattern are peaks (highest performance). These can generally be seen between pad numbers 4-6, 8-9, 15-17, and 20. This is likely because the padding value separate the struct variable that is being operated on enough that each thread is accessing data from its own cache line, so less invalidations occur. For the case with one thread, we see near max performance. This is because it is the only thread running, so it does not have to contend with other threads, thus it will rarely encounter invalid data when reading from Array[i].

Table Data

	1 Thread – shared	1 Thread – private	2 Thread – shared	2 Thread – private	4 thread – shared	4 thread – private
0	407.47	407.15	44.19	413.19	51.01	397.75
1	419.89	407.15	121.28	413.19	17.94	397.75
2	413.22	407.15	45.33	413.19	52.96	397.75
3	419.52	407.15	112.92	413.19	17.82	397.75
4	415.03	407.15	119.70	413.19	55.17	397.75
5	414.24	407.15	118.54	413.19	53.99	397.75
6	179.83	407.15	94.08	413.19	16.50	397.75
7	418.88	407.15	131.55	413.19	55.98	397.75
8	415.96	407.15	111.89	413.19	58.31	397.75
9	415.00	407.15	106.98	413.19	18.66	397.75
10	376.41	407.15	115.49	413.19	21.72	397.75
11	412.99	407.15	117.41	413.19	59.22	397.75
12	314.20	407.15	119.92	413.19	74.94	397.75
13	398.16	407.15	41.73	413.19	59.35	397.75
14	413.52	407.15	38.04	413.19	64.60	397.75
15	419.82	407.15	134.61	413.19	60.05	397.75
16	412.94	407.15	133.49	413.19	55.80	397.75
17	415.13	407.15	55.48	413.19	23.58	397.75
18	376.42	407.15	55.35	413.19	59.13	397.75
19	415.17	407.15	138.90	413.19	65.31	397.75
20	389.56	407.15	139.71	413.19	60.33	397.75

The above table shows the padding number in the leftmost column (NUM) and the number of threads+trial type in the topmost row (NUMT). The data incident to each row column pair is the calculated number of MegaFlops per Second.