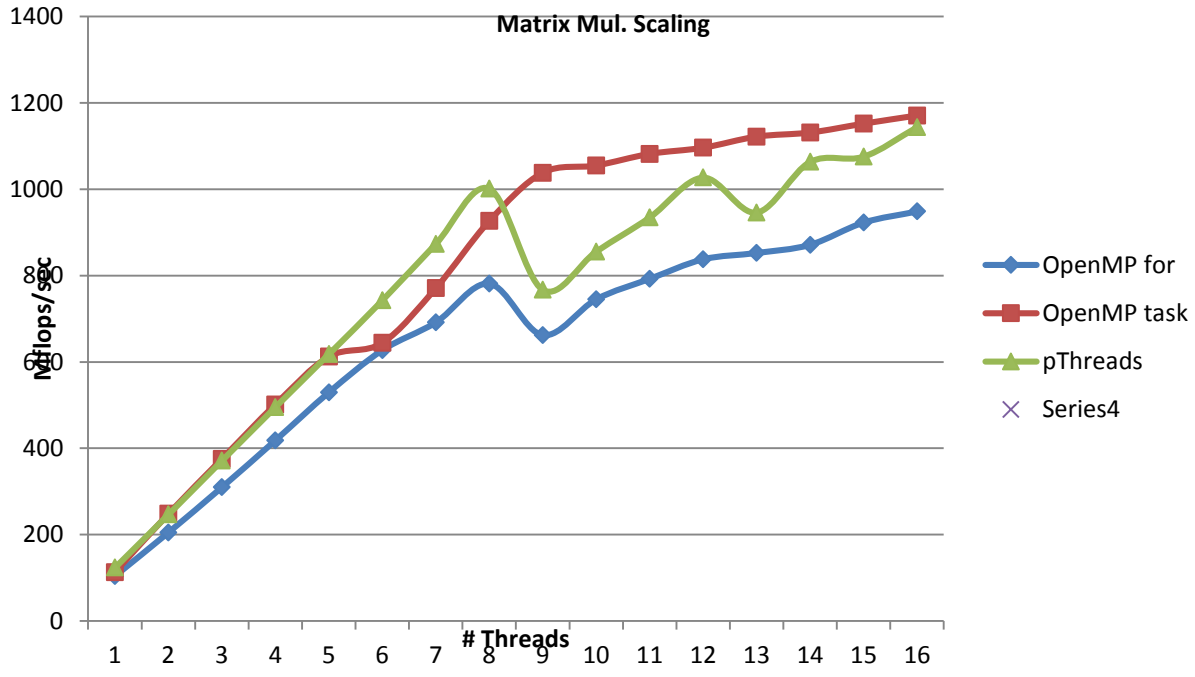
Sriharsha Ayinala
1378833

3.2.1)

Threads	OpenMP for	OpenMP task	pThreads
1	104.1	112.8	123.7
2	204.9	248.1	246.8
3	309.7	375.2	371.4
4	418.3	501.3	495.4
5	529.4	612.2	618.1
6	627.1	643.9	742.6
7	691.5	770.6	873.4
8	781.3	926.2	1001.5
9	662.1	1038	767.2
10	744.9	1054.8	855.6
11	792.3	1081.7	934.6
12	837.6	1095.9	1027
13	852.3	1121.7	946
14	871.2	1131.1	1063.3
15	922.6	1151.7	1075.5
16	948.4	1170.6	1143.6



3.2.2) The fastest speed achieved was 1170.6 Mflops/sec with OpenMP-task, which is almost 10x faster than the serial computation. Linear scaling was not achieved as we would need to have 16x speed up (about ~1800 Mflops/sec) to call it linear for 16 threads. After 8 threads, the scaling dropped. The overhead cost to handle more than 8 threads outweighs adding more processors. After 8 threads, there

is always a dip in performance as the overhead to handle the new processor costs more resources than the resources obtained with an extra tread. The same occurs after 12 processors.

3.2.3) My matrix multiply this about 76x slower compared to that peak. There are many tools such as the Intel intrinsics SSE that I have not used. The matrix multiply method used is also the naïve method. There are other faster techniques that can be used to harness the cache. The proper looping technique to increase the probability of having the value in a lower-level cache drastically increase the speed as well.

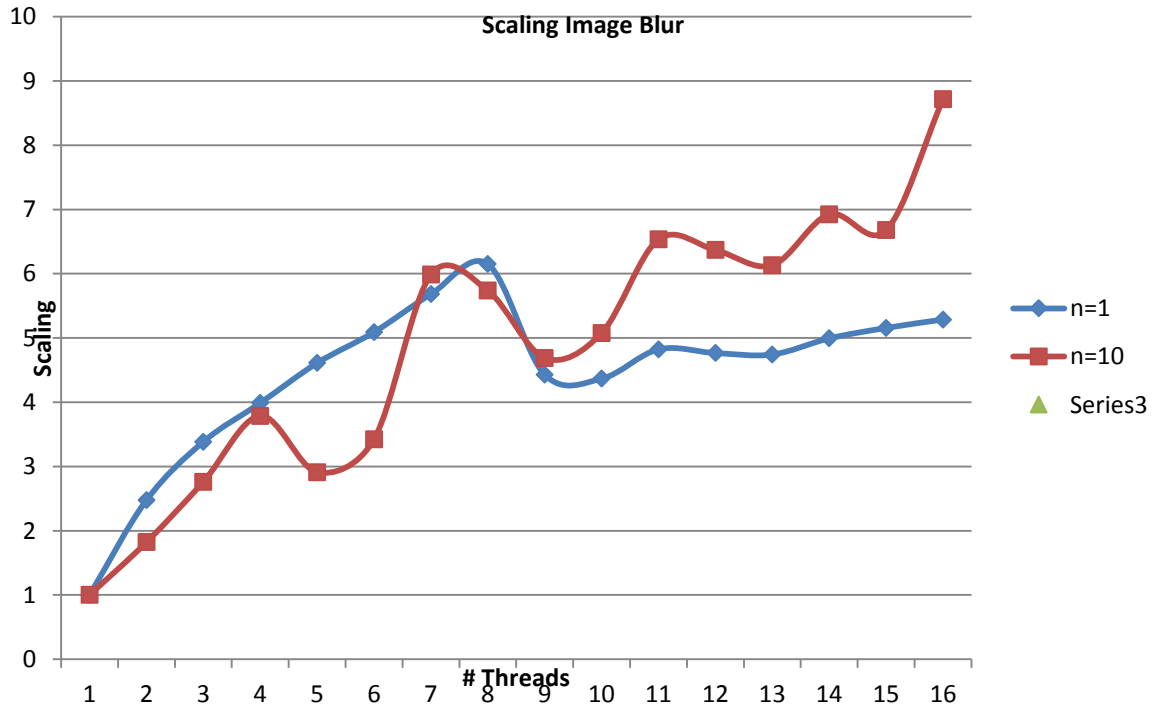
3.2.4) I like both for different reasons. OpenMP I am familiar with and the wrapper makes it clear what regions are parallel and what attributes the parallelization has. However, Pthreads keeps the simplicity of forking and joining.

3.26) It works very similar to Pthreads where it first sets up all the data in a struct and then it calls the parallel function with the struct passed arguments. The parallel function is very similar to Pthreads fork. After the parallel function is called, it ends with the join in reassigns all the calculated value back into the variable.

Sriharsha Ayinala
1378833

4.1.1 & 4.1.2)

Threads	Time (sec)		Scaling	
	n=1	n=10	n=1	n=10
1	0.00396	0.0428	1	1
2	0.0016	0.0235	2.475	1.821276596
3	0.00117	0.0155	3.384615385	2.761290323
4	0.000993	0.0113	3.987915408	3.787610619
5	0.000859	0.0147	4.610011641	2.911564626
6	0.000778	0.0125	5.089974293	3.424
7	0.000697	0.00715	5.681492109	5.986013986
8	0.000644	0.00746	6.149068323	5.737265416
9	0.000894	0.00913	4.429530201	4.687842278
10	0.000907	0.00843	4.366041896	5.077105575
11	0.000821	0.00655	4.823386114	6.534351145
12	0.000831	0.00672	4.76534296	6.369047619
13	0.000835	0.00698	4.74251497	6.131805158
14	0.000793	0.00618	4.99369483	6.925566343
15	0.000768	0.00641	5.15625	6.677067083
16	0.000749	0.00491	5.287049399	8.716904277



Scaling for the first eight threads of the constant radius test shows better results than the random radius test. After the first eight threads, the random radius test showed better scaling. This is probably because

Sriharsha Ayinala
1378833

the constant radius test for the first eight threads are such small task that the overhead for adding more threads does not weigh more than the computation. For the random radius test, the amount of blur computation needed to be done per pixel can be more than one adjacent pixel. This is the larger task and thus more computation is required. Adding more threads for this task showed better scaling then the constant radius as the overhead for adding more threads is relatively not as large as the computational task that needs to be done. Computing the blur for one adjacent pixel in the constant radius test it such a small task that adding more threads causes and overhead that weighs more than the computation. To improve the scaling for the non-constant blur radius, one way of improving scalability is to implement some sort of load-balancing. We can divide the task into smaller ones such that a small probable for the tasks to be completed at equal time. This way the tasks can be evenly distributed to all the processors.

4.2) I spent about 10 hours on this assignment. The code snippets given for the assignment are very useful and they gave me a chance to learn lot of things. There was hard trying to figure out how to compile and what each of the files were.