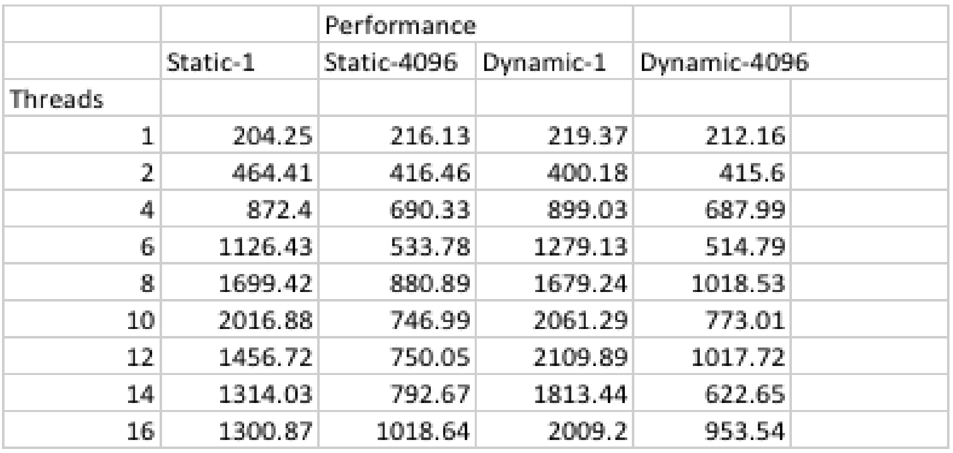
Project #2: Static vs Dynamic and Small vs Large Chunk size

Tanya Khemani
932988007

1. Tell what machine you ran this on

   OSU server- Flip

2. Create a table with your results.



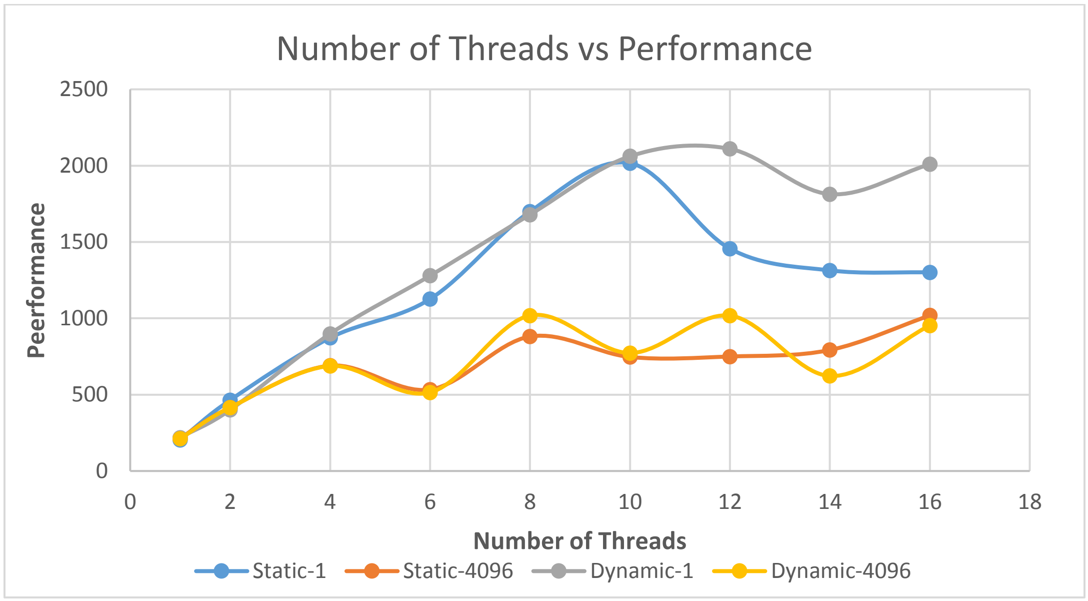
		Performance			
	Static-1	Static-4096	Dynamic-1	Dynamic-4096	
Threads					
1	204.25	216.13	219.37	212.16	
2	464.41	416.46	400.18	415.6	
4	872.4	690.33	899.03	687.99	
6	1126.43	533.78	1279.13	514.79	
8	1699.42	880.89	1679.24	1018.53	
10	2016.88	746.99	2061.29	773.01	
12	1456.72	750.05	2109.89	1017.72	
14	1314.03	792.67	1813.44	622.65	
16	1300.87	1018.64	2009.2	953.54	

3. Draw a graph. The X axis will be the number of threads. The Y axis will be the performance in whatever units you sensibly choose. On the same graph, plot 4 curves:

- static,1
- static,4096
- dynamic,1
- dynamic,4096

Project #2: Static vs Dynamic and Small vs Large Chunk size

Tanya Khemani
932988007



4. What patterns are you seeing in the speeds?

   We can see that as the number of thread increases, performance also increases. But the chunksize also effects the performance. From the graph, we can conclude dynamic scheduling with number of threads equal to 1 has the fastest performance as compared to static scheduling and also as compared to the number of threads. This is similar to the example of dealing a pre-arranged deck of cards with all first few cards to be the good ones and the last ones to be bad.

5. Why does chunksize 1 vs. 4096 matter like this?

   From the graph, we can see that chunksize 1 is having a better performance than the chunksize 4096. However, in our program, the compiler might give unbalanced workloads to the threads since by default, in our program the inner-loop is involved in

Project #2: Static vs Dynamic and Small vs Large Chunk size

Tanya Khemani
932988007

static-scheduling only, so as the number of iterations of inner-loop increases, unbalanced workloads would impact the performance.

6. Why does static vs. dynamic matter like this?

   Static scheduling divides the total loop counts to the number of threads and equally assigns the tasks to the threads. While in dynamic scheduling, OpenMP divides the iterations into chunks of size chunk-size. Each thread executes a chunk of iterations and then requests another chunk until there are no more chunks available. In dynamic scheduling, there is no particular order in which the chunks are distributed to the threads. The order changes each time when we execute the for loop. Dynamic scheduling, uses the internal work queue to give a chunk-sized block of loop iterations to each thread. As soon as a thread is finished, it retrieves the next block of loop iterations from the top of the work queue and gives to the ideal thread. Therefore, this can result in overheads in case of small number of threads as there can be extra overheads involved in waiting for the next block of loop iterations. However, dynamic scheduling is also not that effective in case of very large number of threads.