Erin Alltop
CS475 – Spring 2018
Project 2 Commentary

1. 1. This was run on Flip1. More specifically:
flip1.engr.oregonstate.edu Dual Intel(R) Xeon(R) CPU X5650 (6 cores each with Hyperthreading, 24 total threads) 96GB RAM

2. Here you will find five tables. I ran four trials so that I could obtain the average data across the files. The averaged data can be found in the fifth table. I ran each type with 1, 2, 4, and 8 threads.

TRIAL 1				
NUMT	COARSE STATIC	COARSE DYNAMIC	FINE STATIC	FINE DYNAMIC
1	9.429249	18.155773	9.084654	11.551748
2	34.271599	31.14352	13.661664	18.297255
4	49.645076	51.568411	23.237708	9.392183
8	76.73629	65.987566	15.309807	8.306847

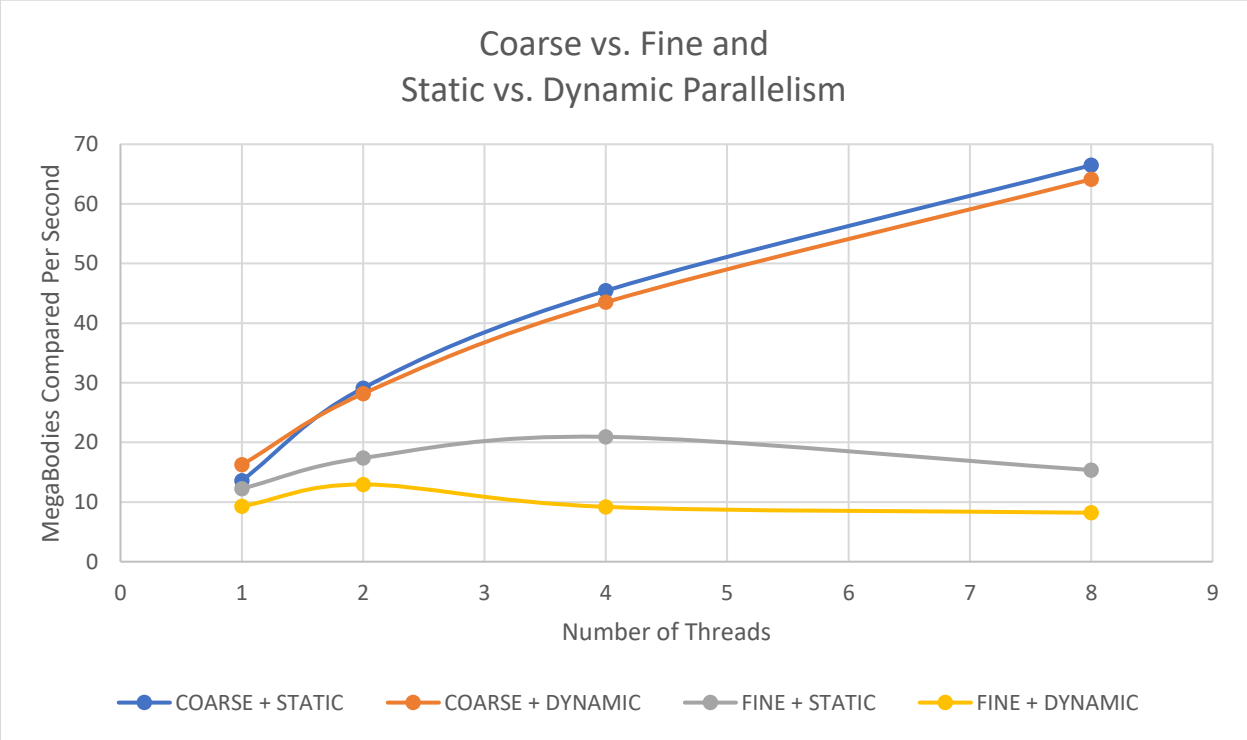
TRIAL 2				
NUMT	COARSE STATIC	COARSE DYNAMIC	FINE STATIC	FINE DYNAMIC
1	17.379458	11.171774	10.037373	6.516415
2	18.489393	24.432328	15.598049	19.095216
4	34.778809	31.770322	20.702422	9.981365
8	78.416258	65.068463	15.820655	9.224118

TRIAL 3				
NUMT	COARSE STATIC	COARSE DYNAMIC	FINE STATIC	FINE DYNAMIC
1	17.734798	17.596173	17.153489	10.886112
2	32.987644	27.006917	19.72501	6.89993
4	44.863695	41.945603	16.49457	7.887203
8	48.4323	61.193868	15.298388	7.938407

TRIAL 4				
NUMT	CORASE STATIC	COARSE DYNAMIC	FINE STATIC	FINE DYNAMIC
1	9.828426	18.036523	12.713043	8.202172
2	30.682128	30.12213	20.561494	7.507719
4	52.442579	48.72975	23.305167	9.507357
8	62.322659	64.213359	15.025262	7.341995

AVERAGE ACROSS TRIALS				
NUMT	COARSE + STATIC	COARSE + DYNAMIC	FINE + STATIC	FINE + DYNAMIC
1	13.59298275	16.24006075	12.24713975	9.28911175
2	29.107691	28.17622375	17.38655425	12.95003
4	45.43253975	43.5035215	20.93496675	9.192027
8	66.47687675	64.115814	15.363528	8.20284175

3. The graph has the averages from all trials:



4. The data shows some interesting results regarding the speeds of the different types of scheduling and parallelism used. Most notably, there is a significant difference in speeds between coarse- and fine-parallelism. There is a large gap between these two indicating a marked increase in speed for coarse-parallelism. Both static and dynamic scheduling types of coarse-parallelism however indicate a fairly similar trajectory of speed. So much so that I double checked that my code was working correctly! Interestingly, both static and dynamic scheduling types of fine-parallelism runs see a slight dip in speed after 4 threads. I would be interested in running this with more threads to see the behavior after even more threads are added. For both coarse- and fine-parallelism, the dynamic scheduling type performed worse than the static type.

5. Static scheduling is when the work is divided in equal-sized chunks, and Dynamic scheduling is when the work is divided in a round-robin style – after a thread is finished, the next block of loop iterations begins. This seems fairly straight-forward to explain the speed differences involved. The data I obtained shows that the dynamic scheduling type for both coarse- and fine-grained parallelism is less than static.

This makes sense because each thread will have to wait for the next to finish before doing its work. This will automatically reduce speed because of the extra overhead involved.

False sharing happens when two threads impact the performance of each other by modifying and attempting to modify the same cache line. You can begin to see the performance differences between coarse and fine parallelism when thinking of how false sharing works. Since coarse-grained parallelism uses larger chunks of data less frequently, and fine-grained parallelism uses small chunks of data more frequently, I would hypothesize that fine-grained parallelism has a much larger probability of overlapping data and creating more opportunity for false sharing. This therefore reduces performance because more time is needed to correct this issue. In coarse-grained parallelism, since data is interacting with each other must less frequently, the occurrence of false sharing is going to be less in general, which results in a faster performance speed. This is apparently in the data that I obtained.