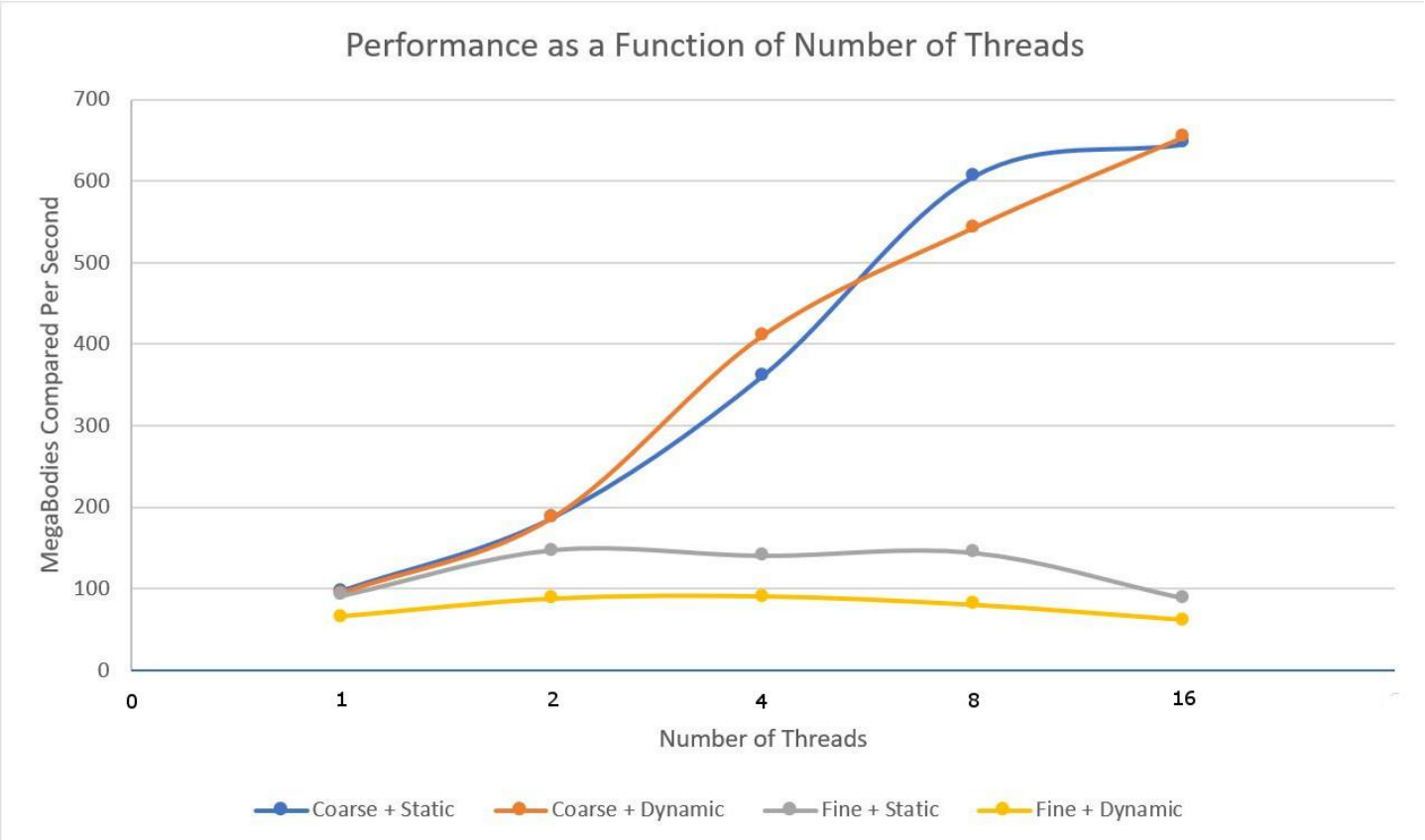
1. All tests were ran on OSU's flip server (flip2), Load average at time of tests: 1.02, 1.07, 1.06
2.

Seconds to Process 100 bodies for 200 Steps					
	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
Coarse + Static	0.206	0.1071	0.0553	0.033	0.0309
Coarse + Dynamic	0.21	0.1066	0.0486	0.0368	0.0305
Fine + Static	0.22	0.136	0.1428	0.1392	0.2285
Fine + Dynamic	0.3046	0.2289	0.2223	0.2511	0.3271

MegaBodies Compared Per Second					
	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
Coarse + Static	97.06	186.59	361.09	605.83	645.59
Coarse + Dynamic	95.23	187.49	410.99	542.65	654.09
Fine + Static	90.87	146.98	139.99	143.67	87.5
Fine + Dynamic	65.65	87.34	89.96	79.65	61.13

3.



4. When MegaBodies are computed using the coarse parallel method the amount per second increases with the number of thread. Using the coarse method there does not appear to be a significant difference between static and dynamic assignment from the thread pool. Speedup between static and dynamic is about the same as well with static being 6.666 for 1 to 16 threads and dynamic being 6.885 for 1 to 16 threads.
   When MegaBodies are computed using the fine parallel method the amount per second increases initially but then begins to decrease. The amount of MegaBodies per second computed are also drastically smaller than then for the coarse method. The speedup is also negative for 1 to 16 threads for both static and dynamic.

5. I think that the cause of the odd behavior in the fine-grained parallelism is because of the overhead incurred by using the fine grained method. When using coarse-grained parallelism a thread pool is created only 200 times (equal to NUMSTEPS). However when using fine-grained parallelism a thread pool is created 20000 times (equal to NUMSTEPS x NUMBODIES). When fine-grained parallelism is used a thread pool is created 100 times more often than with coarse-grained and it is outweighing the benefit of parallelizing the code for higher thread counts.