Geoffrey Corey
CS475 Project 2

Test Platform:
I tested this on my personal laptop:

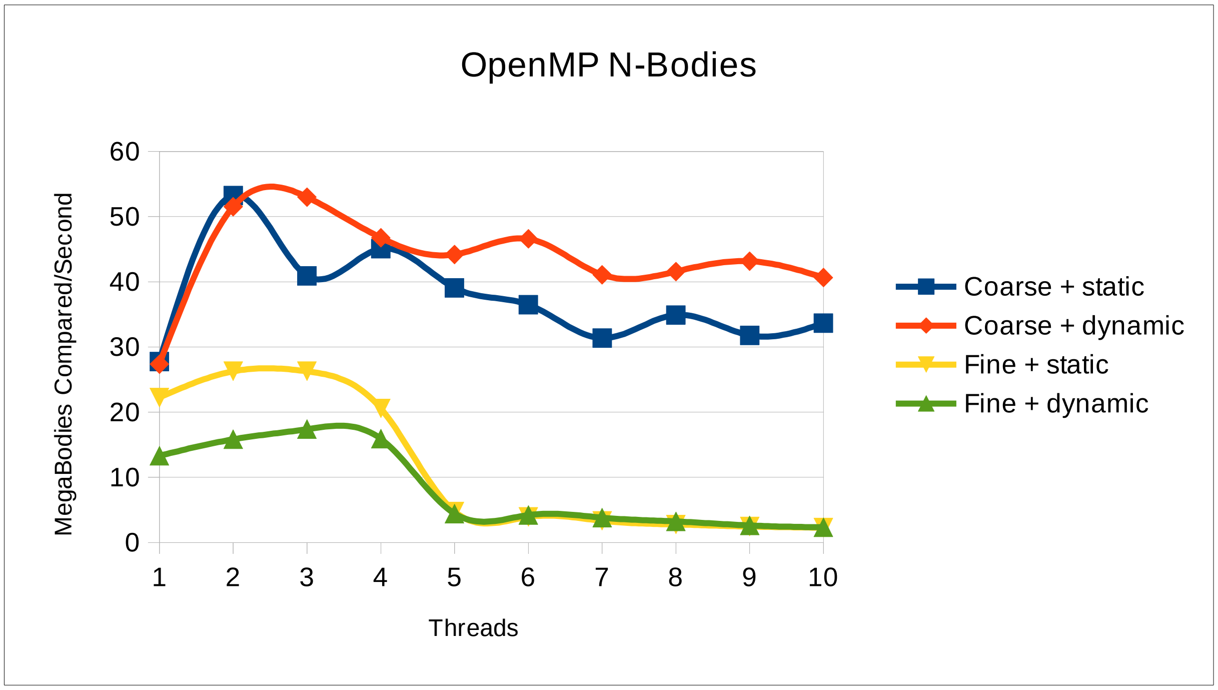
- Intel(R) Core(TM) i3-2330M CPU @ 2.20GHz
- 2 Cores, 4 Threads

I minimized my computer usage by closing out all web browsers and closing active processes.
In the test script that I ran, I added a sleep 1 command to allow the cpu to calm down between runs and get a better set of results.

Test Results:

		Threads									
		1	2	3	4	5	6	7	8	9	10
MegaBodies Compared Per Second	Coarse + static	27.760511	53.243974	40.920199	45.071017	39.078458	36.488933	31.371209	34.911479	31.788415	33.647809
	Coarse + dynamic	27.350085	51.523279	53.001089	46.746571	44.188412	46.581775	41.079595	41.573074	43.161339	40.648293
	Fine + static	22.245177	26.270849	26.270849	20.579139	4.754845	3.932562	3.336066	2.76732	2.44764	2.278029
	Fine + dynamic	13.304485	15.830367	17.390954	15.913908	4.430141	4.212284	3.813216	3.226383	2.612676	2.329414

Graphs:



Analysis:
At 1 thread, the results are as expected as there is no speedup to be had. And depending on the compiler's optimization, linking against pthreads and openmp could even cause some performance degradation. It is interesting the note the difference between Fine + static and Fine + dynamic when threads = 1.

At 2 threads, all types get a speed increase, however the coarse parallel for gets the most benefit.

At 3 threads, Coarse + dynamic has the best performance while coarse + dynamic takes a drastic performance hit. In the other 3 types of tests, 3 threads appears to be the sweet spot in terms of performance. In general, the speed increase for each thread introduced in the test until 4 threads are used.

Once threads >= 4, the performance decreases and then levels off for the most part at higher thread numbers.

Behavior:

Between 1 and 3 threads, the increase is as expected. It seems to be using the cache in the most efficient way for how the simulation is programmed. Once the 4th core is introduced, and with this program being heavily floating point operations, it could be a limitation of the CPU and the FPUs as well as the way the program is using cache. The drastic drop after 4 threads can be attributed to the CPU not having enough FPUs for the number of threads that are being run.

The difference between the coarse and the fine grain parallel can be attributed to the way the compiler and OpenMP can optimize the different for loops. In the coarse grain, you are able to optimize the outer for loop, which allows the compiler to optimize the inner for loops and do some loop unrolling and function in-lining. With the fine grained for loop parallel, there is less optimization to be done in the for loops and there is even more context switching going on which can decrease cache usage due to it being invalid.