Drake Seifert

CS 475 Project 2

Tell what machine you ran this on

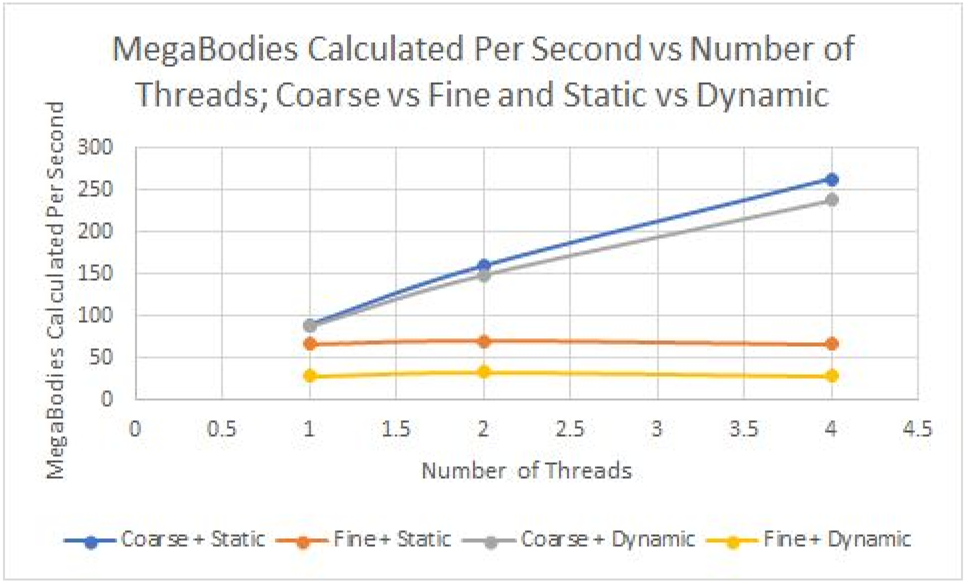
All tests were ran on OSU's FLIP servers. All tests were ran 500 times each, and the average MegaBodies Compared Per Second was added to the dataset assuming that there was not a large difference between the calculated average and the peak value.

Create a table with your results.

	Coarse + Static	Fine + Static	Coarse + Dynamic	Fine + Dynamic
1	89.48	67.43	87.85	28.96
2	159.75	70.83	149.02	33.8
4	263.48	67.15	238	29.03

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:

Coarse+static
coarse+dynamic
fine+static
fine+dynamic



What patterns are you seeing in the speeds?

Coarse-grained parallelism scales much better based off of the number of threads as opposed to fine-grained parallelism. Fine-grain parallelism actually slowed down given a larger number of threads. In general, static scheduling seemed to perform better than dynamic scheduling. The combination of static scheduling and coarse-grained parallelism produced the best results. This makes sense given that they are the combination of the two better-performing methods of parallelism and scheduling. The same can be said for the combination of fine-grained parallelism and dynamic scheduling; the combination of the two worst methods produced the worst results.

Why do you think it is behaving this way?

Coarse-grained parallelism gives the threads more work to do, whereas fine-grained parallelism gives the threads less work. Coarse-grained outperforms fine-grained however, because the calculations from the first for-loops computations are taken advantage of in parallel, and the threads are much more resource efficient. In addition, fine-grain shows some performance decreases, probably due to the overhead involved in spawning and managing multiple threads for a fewer number of computations. Overall, the biggest difference in performance for this assignment definitely shows that coarse-grained parallelism is the way to go.

The lecture slides define static and dynamic scheduling as follows:

Static Scheduling: Dividing the total number of tasks T up so that each of N
available threads has T/N sub-tasks to do.

Dynamic scheduling: Dividing the total number of tasks T up so that each of N
available threads has less than T/N sub-tasks to do, and then doling out the
remaining tasks to threads as they become available.

The performance differences here were a bit more subtle than coarse-grained vs fine-grained parallelism. In general, static scheduling outperformed dynamic scheduling. My best guess as to why this would be the case is due to the overhead in managing thread activity when using dynamic scheduling. Because static scheduling can spawn all threads with a preset number of tasks, the threads don’t have to be directly managed and can be free to run until finished. Dynamic scheduling on the other hand requires that the thread activity be more closely monitored so that the workload can be distributed once tasks expire. The time and resources required to do this actually end up slowing down the progress of the threads’ average performance.