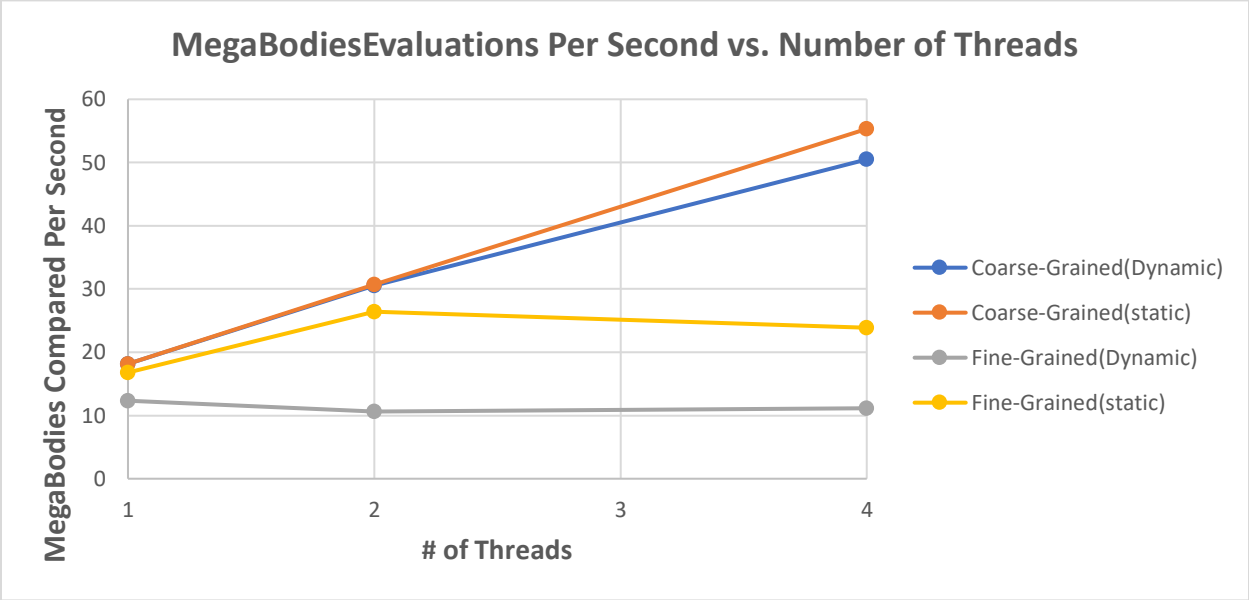
For this assignment, I created a bash script to help test my program with several different combinations of parameters – such as setting the number of threads used, using coarse-grained versus fine-grained parallelism in OpenMP, and using dynamic thread/schedule versus static thread/schedule in OpenMP. To run the script, simply enter in the following command: ./script

1. Tell what machine you ran this on
   - A Linux system on OSU's FLIP server was used to run Project #0.

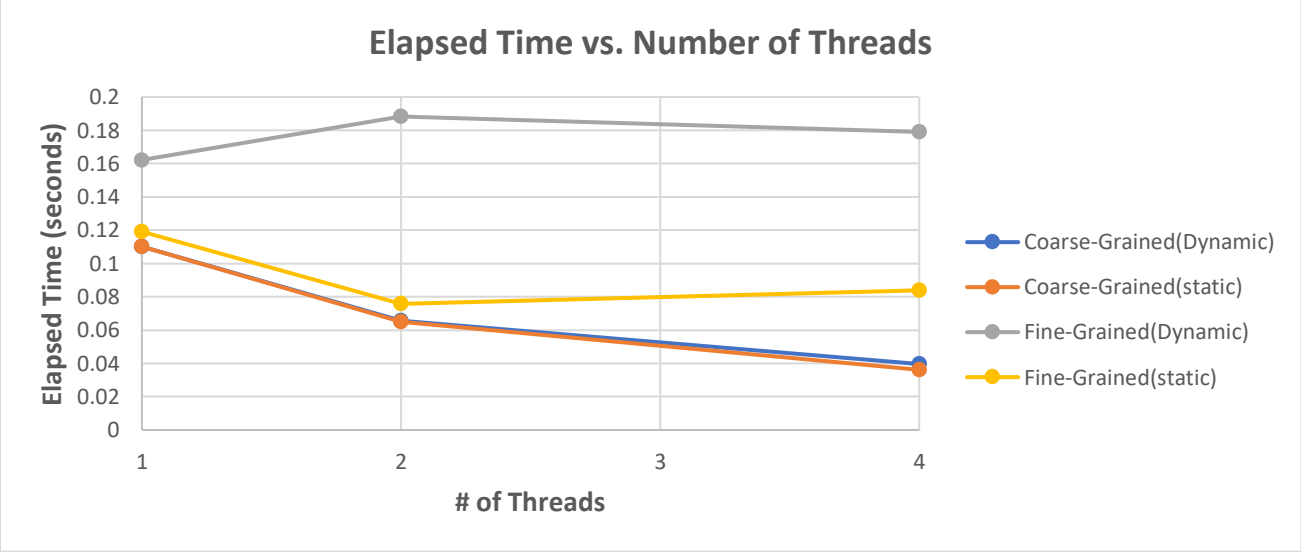
2. Table of Results

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type
1	18.162023	0.11012	Coarse-Grained	Dynamic
1	18.16076	0.110128	Coarse-Grained	Static
1	12.332644	0.162171	Fine-Grained	Dynamic
1	16.792353	0.119102	Fine-Grained	Static
2	30.524596	0.065521	Coarse-Grained	Dynamic
2	30.712769	0.065119	Coarse-Grained	Static
2	10.623219	0.188267	Fine-Grained	Dynamic
2	26.408464	0.075733	Fine-Grained	Static
4	50.489906	0.039612	Coarse-Grained	Dynamic
4	55.319683	0.036153	Coarse-Grained	Static
4	11.172537	0.17901	Fine-Grained	Dynamic
4	23.879773	0.083753	Fine-Grained	Static

3. Graph
   - **Performance** (MegaBodies Evaluations Per Second)



   - **Elapsed Time** (seconds)



4. What patterns are you seeing in the speeds?

Here we are grouping the data by the parallelism/thread type combination and then comparing by the # of threads used to determine the speedup. The patterns I'm seeing here for most cases are as you add in more threads to the program; the speedup ratio increases.

- **Coarse-Grained Dynamic Speedup**

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
1	18.162023	0.11012	Coarse-Grained	Dynamic	0
2	30.524596	0.065521	Coarse-Grained	Dynamic	1.68068253
4	50.489906	0.039612	Coarse-Grained	Dynamic	1.65406947

- **Coarse-Grained Static Speedup**

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
1	18.16076	0.110128	Coarse-Grained	Static	0
2	30.712769	0.065119	Coarse-Grained	Static	1.69118076
4	55.319683	0.036153	Coarse-Grained	Static	1.80120599

- **Fine-Grained Dynamic Speedup**

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
1	12.332644	0.162171	Fine-Grained	Dynamic	0
2	10.623219	0.188267	Fine-Grained	Dynamic	0.86138835
4	11.172537	0.17901	Fine-Grained	Dynamic	1.05171219

- **Fine-Grained Static Speedup**

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
1	16.792353	0.119102	Fine-Grained	Static	0
2	26.408464	0.075733	Fine-Grained	Static	1.57265657
4	23.879773	0.083753	Fine-Grained	Static	0.90424224

Here we are grouping the data by the number of threads used and then showing the difference in speedup between the thread/schedule type used. The patterns I'm seeing here are as you switch from dynamic to static for the scheduling/thread type while using the same number of threads, the speedup ratio increases. This means that static schedule/thread type is much better than dynamic performance wise, as it takes up less time.

- **Dynamic Vs. Static Speedup**

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
1	18.162023	0.11012	Coarse-Grained	Dynamic	0
1	18.16076	0.110128	Coarse-Grained	Static	0.99992736
1	12.332644	0.162171	Fine-Grained	Dynamic	0
1	16.792353	0.119102	Fine-Grained	Static	1.36161441

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
2	30.524596	0.065521	Coarse-Grained	Dynamic	0
2	30.712769	0.065119	Coarse-Grained	Static	1.00617331
2	10.623219	0.188267	Fine-Grained	Dynamic	0
2	26.408464	0.075733	Fine-Grained	Static	2.48593084

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
4	50.489906	0.039612	Coarse-Grained	Dynamic	0
4	55.319683	0.036153	Coarse-Grained	Static	1.09567671
4	11.172537	0.17901	Fine-Grained	Dynamic	0
4	23.879773	0.083753	Fine-Grained	Static	2.13735627

Here we are grouping the data by the number of threads used and then showing the difference in speedup between the parallelism type used. The patterns I'm seeing here are as you switch from fine-grained to coarse-grained for the parallelism type while using the same number of threads, the speedup ratio increases. This means that the parallelism type, coarse-grained, is better performance wise, as it takes up less time.

- **Coarse-Grained Vs. Fine-Grained Speedup**

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
1	12.332644	0.162171	Fine-Grained	Dynamic	0
1	18.162023	0.11012	Coarse-Grained	Dynamic	1.47267526
1	16.792353	0.119102	Fine-Grained	Static	0
1	18.16076	0.110128	Coarse-Grained	Static	1.081487

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
2	10.623219	0.188267	Fine-Grained	Dynamic	0
2	30.524596	0.065521	Coarse-Grained	Dynamic	2.87338411
2	26.408464	0.075733	Fine-Grained	Static	0
2	30.712769	0.065119	Coarse-Grained	Static	1.1629939

# of Threads	maxMegaBodies	Elapsed Time	Parallelism Type	Thread Type	Speedup
4	11.172537	0.17901	Fine-Grained	Dynamic	0
4	50.489906	0.039612	Coarse-Grained	Dynamic	4.51908513
4	23.879773	0.083753	Fine-Grained	Static	0
4	55.319683	0.036153	Coarse-Grained	Static	2.31662656

5. Why do you think it is behaving this way?

# of Threads

In the charts above in #4 where the data is grouped by parallelism/thread type combinations and then compared by the number of threads used, I'm seeing that as you use increase the number of threads used in your program, the overall performance of your program also increases for data sets of the same size. This is because the equation for calculating performance is measured by the amount of work executed in your program (maxMegaBodies) divided by the amount of time it took to do that work. So, as you add more threads to tackle a problem with the same input size, the execution time also goes down which causes the overall performance to increase – which can be seen with all different combinations of parallelism type and thread type tested.

Dynamic Vs. Static

The static schedule/thread type results in better performance than dynamic schedule/thread type because for dynamic – the work is assigned to each thread in the pool during runtime, which creates overhead compared to static.

Fine-Grained Vs. Coarse-Grained

The coarse-grained parallelism results in better performance than fine-grained parallelism because of a cache-miss problem. Coarse-grained parallelism is set to create the team of threads to divide up the 1st inner for loop passes among those threads. This loop goes through each body once. This does not violate spatial coherence, as you do access some of the memory locations of the array around the index of the Planet(bodies) you are currently on, so performance is not hurt here. Fine-grained parallelism is set to create the team of threads to divide up the 2nd inner for loop passes among those threads. Temporal coherence is being violated here because the cache line is being flushed each time a thread completes its work, and then whichever next thread picks up the work next has the cache line reloaded from memory.