Josh Huff, huffj@oregonstate.edu
Professor Bailey
CS 475 – Spring 2017
May 1, 2017

Project 2: Commentary

---

1. Tell what machine you ran this on

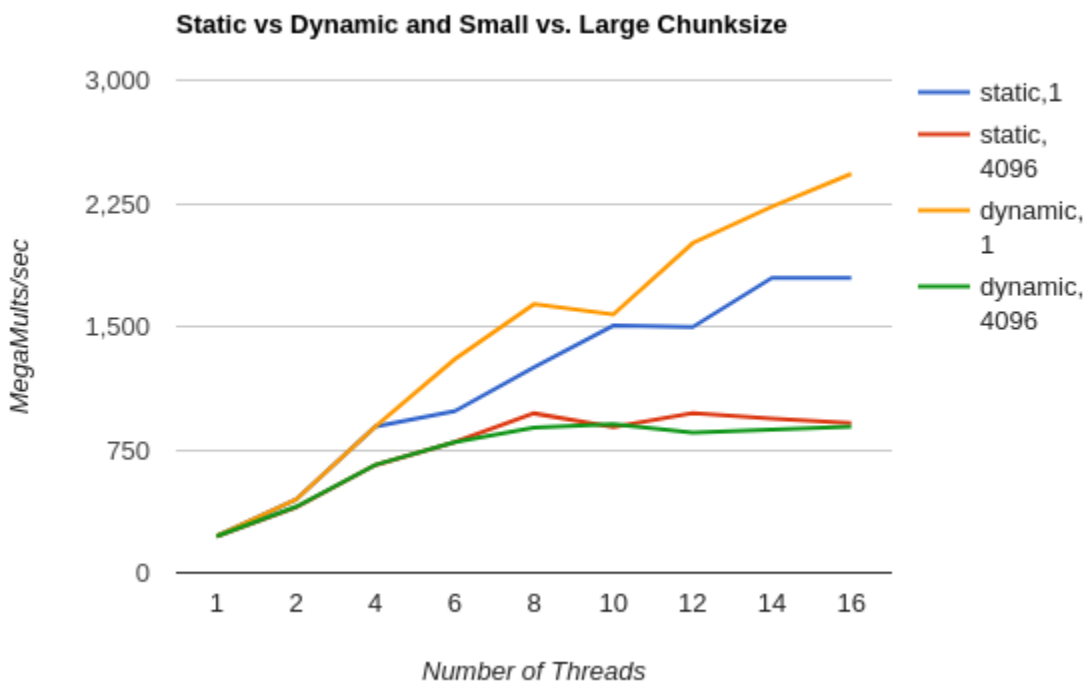
My main workhorse, a home-built machine running Ubuntu 16.04 LTS (64-bit). The processor is Intel® Core™ i5-4690K CPU @ 3.50GHz × 4 and it has 15.6 GiB of usable RAM.

The difference between peak and average performance was minimal, and the load averages were very low (less than 1) across the board. However, the results from running the program locally were suspect and flip2 was used for this report. At the time, the load averages were in the low 5s.

2. Create a table with your results

# of Threads	Static-1	Static-4096	Dynamic-1	Dynamic-4096
1	222.83	223	222.98	222.99
2	445.62	398.1	445.75	401.68
4	891.04	656.15	891.04	657.09
6	983.84	796.27	1300.27	794.71
8	1252.75	970.84	1635.92	883.6
10	1504.78	887.69	1573.13	905.99
12	1495.51	972.61	2007.48	853.71
14	1795.44	939.01	2230.42	870.62
16	1795.32	910.78	2430.33	888.73

3. Draw a graph. The X axis will be the number of threads. The Y axis will be the performance.



4. What patterns are you seeing in the speeds?

With one thread, the speeds are fairly uniform. This is to be expected, because there is no parallelism.

With two threads, the smaller chunksizes outperform the larger chunksizes by about 11-12%. There seems to be no difference between static and dynamic scheduling.

With four threads, the smaller chunksizes have about a 36% increase in their advantage over large chunksizes. There seems to be no difference between static and dynamic scheduling.

With six threads, the lines representing larger chunksizes are virtually identical. Those lines and the static-1 line barely see any improvement. On the other hand, the dynamic-1 line sees a dramatic 46% improvement over its performance with four threads.

With eight threads up to sixteen threads, the large chunksizes plateau and have comparable performances regardless of their scheduling. There is a significant improvement in the static-1 line until it reaches 14 threads, at which point it also begins to plateau. The dynamic-1 line continues to trend upward significantly, and likely would continue to rise with the number of threads.

5. Why does chunksize 1 vs. 4096 matter like this?

A chunksize of 1 is the default because it creates the best probability that the iterations in a loop will be distributed among threads in such a way that the workload will be evenly shared (in theory).

A chunksize of 4096 is substantially larger than the default, and it increases the chances that some threads will be assigned more computationally intensive iterations—which means some threads will work harder than others, and the parallelism suffers as a result. This is reflected in the red and green lines (Static-4096 and Dynamic-4096) reaching a plateau at eight threads. The additional threads make no difference, because the workload is being poorly distributed in large chunks.

6. Why does static vs. dynamic matter like this?

Static scheduling does not care about allocating work after runtime, so if one thread is overburdened and another is given a light load to handle, the overall performance will be limited to the overburdened thread; a team is only as strong as its weakest link. Conversely, dynamic scheduling allows the "master" to assign more work to the "slacker" threads that were given light workloads to begin with.

At low thread levels, the difference between static and dynamic scheduling is negligible. As the number of threads grows, the dynamic scheduling becomes more advantageous. While declaring a small chunksize distributes work to maximize the probability that the threads shoulder their even share of the burden, it does not perfectly take advantage of a larger number of threads. Dynamic scheduling seeks to ensure that threads do not remain idle while there is work to be done, and so the disparity is further minimized.