**Static Scheduler**

**Question: Why do you think some measurements are so erratic?**

There seems to be a common trend of erraticism when the number ‘n’ is low. They tend to stabilize as the intensities increase. I believe what happens here is that having multiple threads with creation and locking overheads for a small problem is expensive.

**Question: Why is the speedup of low intensity runs with iteration-level synchronization the way it is?**

As intensities increase, the locking overhead is overshadowed by the repetitive iteration of the function due to the intensities. At low intensities, these locking overheads are more prominent which is why the speedup of lower intensity runs is the way it is.

**Question: Compare the speedup of iteration-level synchronization to thread-level synchronization. Why is it that way?**

The iteration level locks the shared resource after every computation while the thread level locks the resource at the end of its total computation. This implies iteration level locks way more which makes it very expensive as opposed to the thread level that just locks the resource ‘number of threads’ times. Hence the speed up for iteration level is lower.

**Dynamic Scheduler**

**Question: Compare performance at 16 threads across the different synchronization mode. Why are the speedup this way?**

It appears as if the thread level and chunk level are very comparable across all the runs, thread minutely outperforming chunk because of the fewer locking of the shared resource. It is also notable that as the granularity tends to get closer to n, the runs become very close to sequential and that is expected. Hence, we see a drop, in speed up. And an increase in the opposite case.

The older bench.sh file portrayed iteration as well and that was outperformed by the other 2 drastically which was because of overuse of locks.

**Question: For thread level synchronization, compare the performance at 16 threads of different n and intensity. Why are the plots this way?**

Speed up increases with n. But as granularity increases, speed up becomes 1. Speed up is hence optimal for small granularity because that way the distribution of tasks across the threads is fairly even.

Considering intensities, as mentioned above, more the intensity, more optimal the runs due to fewer locks.