Static Scheduler-

1. Static_sched_plots.pdf is included in the project folder.
2. Measurements are erratic because of the use of mutexes. It is impossible to predict accurately how the runtime behaviour of the program will be, especially with respect to how locks are acquired and released at runtime. Different runs of the program will acquire and release locks in different patterns, which inherently affects the runtime performance of the program differently during each run.
3. Low intensity runs with iteration level synchronization gives very poor results. This is especially noticed when value of n is large. It is mainly down to the fact that low intensity runs will require less time for computation. Because of this, locks get acquired quite frequently, and lots of threads will be in contention for the acquiring the lock. This results in a much lower performance than even sequential execution, mainly down to the huge contention over locks between threads. Context switches to threads wanting the locked shared resource will take away a lot of time.
4. It can be noticed that thread level synchronization has a vastly superior performance, especially when values of n are high. The speedup for thread level synchronization is almost proportional to the number of threads when values of n and intensity are high. The same is not true for iteration level synchronization. Iteration level mostly offer almost the same speed as a sequential performance, or is inferior to the sequential one. It mainly boils down to the huge number of locks acquired in an iteration level synchronized thread. An iteration level synchronized thread takes n locks, while a thread level synchronized one takes only locks equal to number of threads. That alone is the main reason why thread level synchronization is superior - a global shared resource is only locked once per thread in thread level synchronization.

Dynamic Scheduler-

1. Dynamic_sched_plots.pdf shows the pattern of speedup and time.
2. Comparing between chunk level synchronization and thread level synchronization for 16 threads, it can be noticed for granularity of 1 and high values of n, thread level synchronization gives better performance. This is because in chunk level synchronization, the synchronization overhead is high since chunk size is low. It can also be noticed that for reasonable chunk sizes (say 100 to 1000) and high values of n, chunk level synchronization gives slightly better performance. This is because the work is balanced and chunk level synchronization works best with balanced loads.

   The performance for thread and chunk level synchronization is similar in most cases. This is because unless value of granularity is very low (say 1), synchronization overhead will not affect performance of chunk level synchronization that much.

3. The speedup of thread level synchronization increases with the value of n. Here are some observations-
   A. The performance depends a lot on value of n.

B. For the same intensity, higher values of n tend to have higher performance. For high values of n (say n = 100000000), speedup is almost as high as 18.5. This is because there is more chance of performance improvement if the number of iterations are more, so that parallel operation will have more effect on performance. With the right granularity which reduces synchronization overhead, high performance can be achieved.

C. If we compare the same value of n (say 10000) and different intensities (1, 10, 100, 1000) for threads=16, we can see that higher intensities have better speedup performance. This is because in lower intensity counterparts, the individual chunks may get completed really fast and there will be contention for acquiring the lock for getNext().

D. As granularity increases to very high values, the performance degrades. The perfect granularity depends upon the amount of work (n). If value of n is low (say 10000) and granularity is high (say 5000), we cannot get good speedup because the work will be divided among two threads. On the contrary, if value of n is high (say 100000000) and granularity is very low (say 1), the performance will be bad because there will be a lot of synchronization overhead.