Assignment p-threads:

Question #2: Why do you think some measurements are so erratic?

As we can see the graph output the speedup differs for iteration synchronization from thread level synchronization as it shows a no much speedup with increase in threads i.e. because of the way it differs from the way the mutexes are used in iteration than thread level. Whereas in the case of some graphs the graph plot seems distorted like for example for the graph with n values are comparatively low the graph shows a good speedup but as n increases (e.g. $n = 1000000$) the intensity increases the computations become more expensive and thus the speedup decreases for the thread level synchronization which only increases if we increase the number of threads that means as the work load gets distributed more it gives more speedup. There might be another reason for this which is hardware specific as the nodes on which some part of code or some threads work may perform different and may not give better performance than other nodes and thus these discrepancies are observed. Also, as we know the speedup is always the difference between the sequential runtime and the parallel run time, so it also depends on which node the sequential run was tested and on which the parallel test was run. If the sequential run was run on the fastest node and the parallel node on slower node then the parallel code execution time is not going to show much effect and is not going to make much difference.

Question #3: Why is the speedup of low intensity runs with iteration-level synchronization the way it is?

As observed from the plots, the iteration synchronization method speedup output for the low intensity is less than it is for the high intensity, this means as the intensity or the workload increases the speedup increases which goes with the Gustafson's law. According to Gustafson's law, if the sequential run remains one fixed overhead, the larger datasets takes the same time and gives more speedup as the parallelism increases i.e. number of threads (or processors) i.e. it gives speedup if the workload is more which is observed in the graphs with more intensity. In lower intensity graphs as the number of n is more the code gets more speedup than when the range of n is less.

Question #4: Compare the speedup of iteration-level synchronization to thread-level synchronization. Why is it that way?

For Thread level integration, for a particular value of intensity the code execution speeds up as the number of threads and n increases. Whereas, as observed for the integration level execution for each intensity the code execution speedup steeps down a little and then remains almost constant for increase in n. The probable reason for this could be the way the result variables are locked in both the approaches like at iteration level for each iteration the locks are acquired and released more frequently than as compared to in thread level as it is locked only after each for loop completion thus making thread level execution faster. As in, iteration every other thread waits for the locks to be released thus giving no much speedup. This goes according to the Amdahl's law as it says as the maximum part of the program gets more parallel than sequential, it increases the number of processors and thus speedsup the process.