Programming Assignment 4

1. Thread description

When the program runs the main function, it first begins by reading the data from the training data document and then proceeds to create the threads. The number of threads is passed by the arguments of the program execution, which the program uses to create that many threads accordingly. After creating the threads, the thread attaches a tree to the forest and after it is done, it checks if there is another tree available for creation. If there is, it works with it; otherwise it exits because its job is done.

The testing process works similar to the training process, in which threads are created and they work with trees until there's no other tree available to work it, where it would exit and join the main thread.

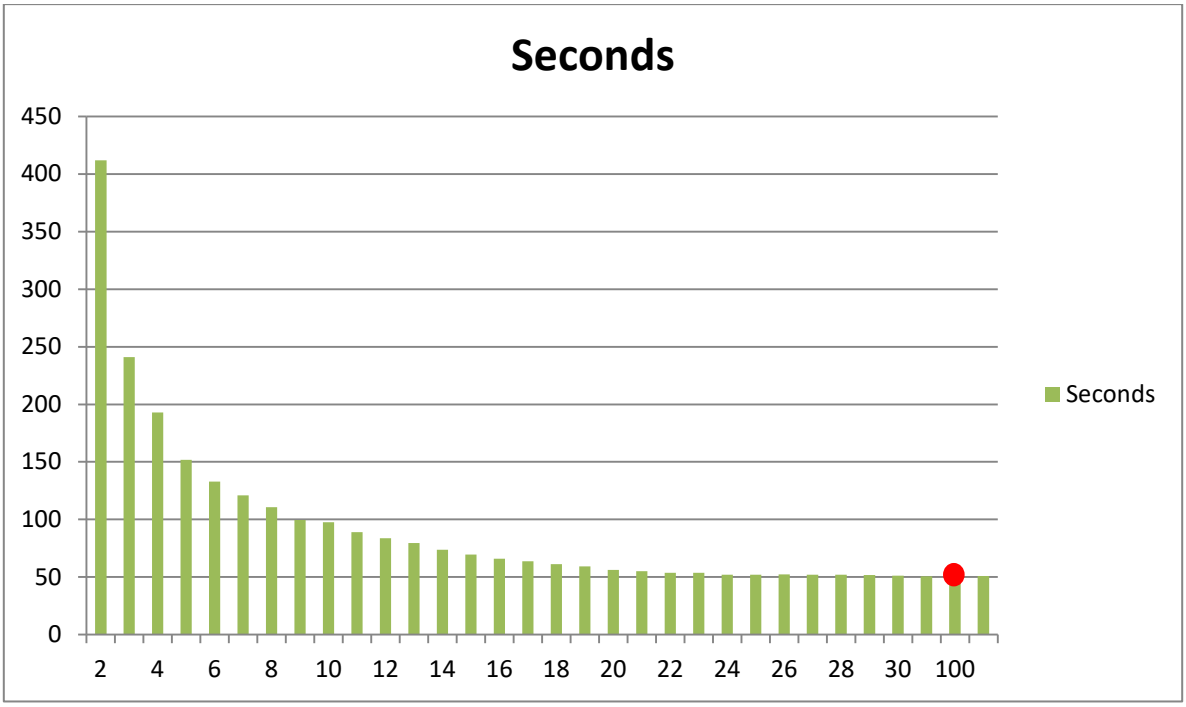
2. Thread Count Speed

Testing for this part of the report was made with 5000 trees and 120 items per tree. I began by trying increasing the number of threads one by one to get all the results using the command line showcased in the spec file. The figure of 5000 trees is for illustration purposes, and not the optimal number of trees.

Example testing:

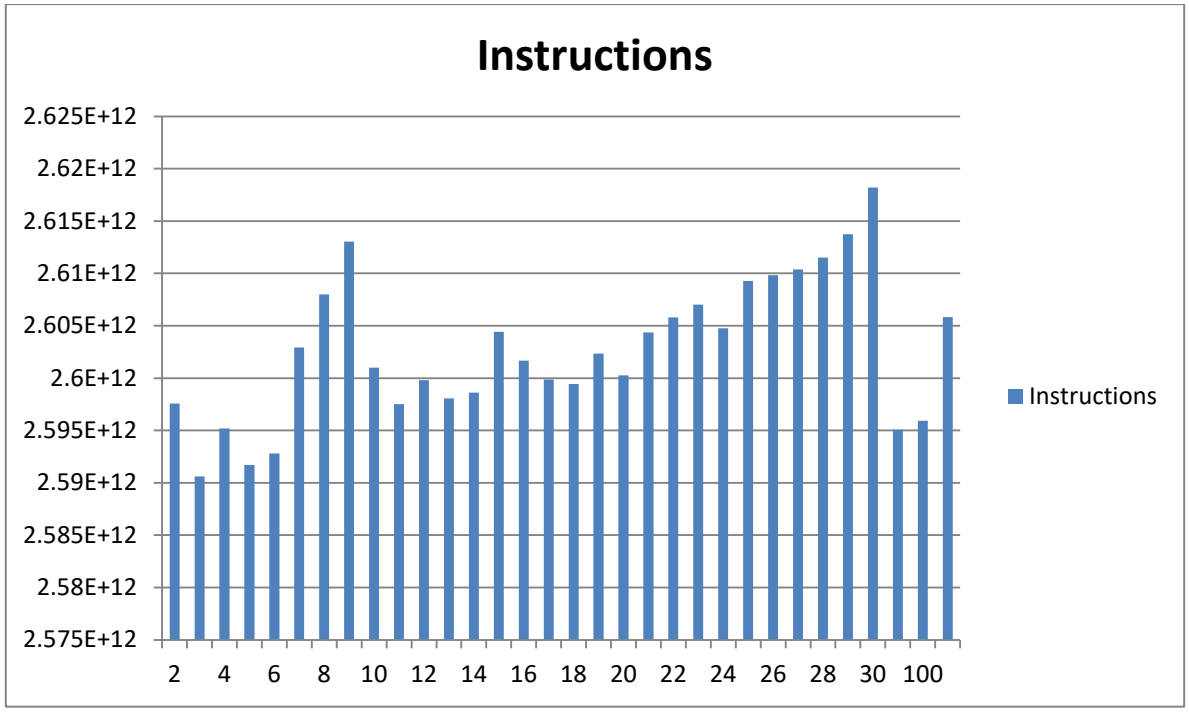
```
b06902102@linux5 [~/HW4/main] perf stat -e instructions:u -v ./hw4 -data ../data
 -output submission.csv -tree 5000 -thread 2
Using CPUID GenuineIntel-6-2D
alias offcore_response.corewb.any_response differs in field 'desc'
alias offcore_response.corewb.any_response differs in field 'long_desc'
instructions:u: 2597562036854 822408513879 822408513879

 Performance counter stats for './hw4 -data ../data -output submission.csv -tree
 5000 -thread 2':

 2,597,562,036,854      instructions:u


     411.717641196 seconds time elapsed

     820.612861000 seconds user
       0.288944000 seconds sys
```

Graph of the results



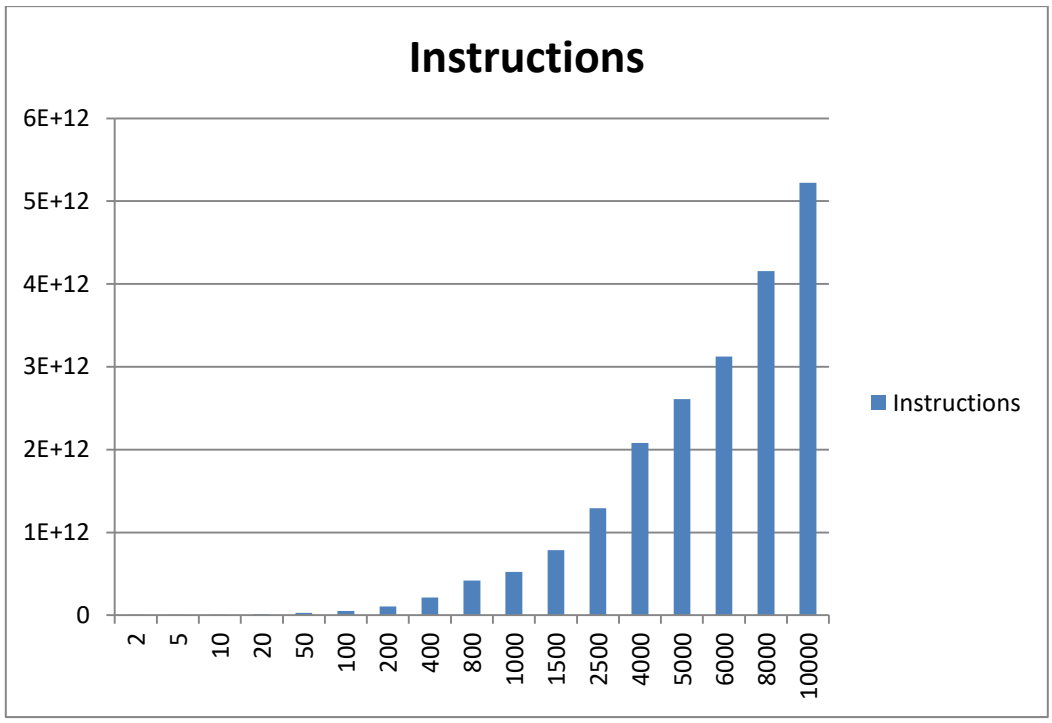
We can see by the graph that the time decreases significantly after 6 threads and just at over 14 threads, changes in time are not significant enough to seriously impact the performance of the program. Minimum time is at 100 threads, the last test was made at 250 threads and it increased for a bit, so it might hint that performance is best between 100 and 250, but nothing definitive.

3. Threads and instructions



With the same settings as in the previous graph, I also registered the number of instructions per iteration of the code. Number of instructions varies between relatively small ranges, so we can note that the number of instructions doesn't change that much when we only change the number of threads.

4. Instructions and Trees



In the graph we can clearly see that the more trees we utilize in the program, the more instructions it'll end up executing. This can also show in the execution time, which also increases in the same fashion as the number of trees increases.

5. Remarks

- I tried running the program with 400 threads, but it was remarkably slow (never finished if I used 500 threads). It was not of any help anyways, since time stopped dropping significantly before I reached that number.
- Given the nature of randomization involved, the accuracy rate was not affected by a huge increase in the number of trees above 1800, but the program did get slower and slower.
- Trees with a number of entries greater than 60 were faster and faster, but the accuracy was just relatively the same, so increasing them significantly would not make much of a difference (besides, I settled for 120 since it seemed a good balance between accuracy and speed).