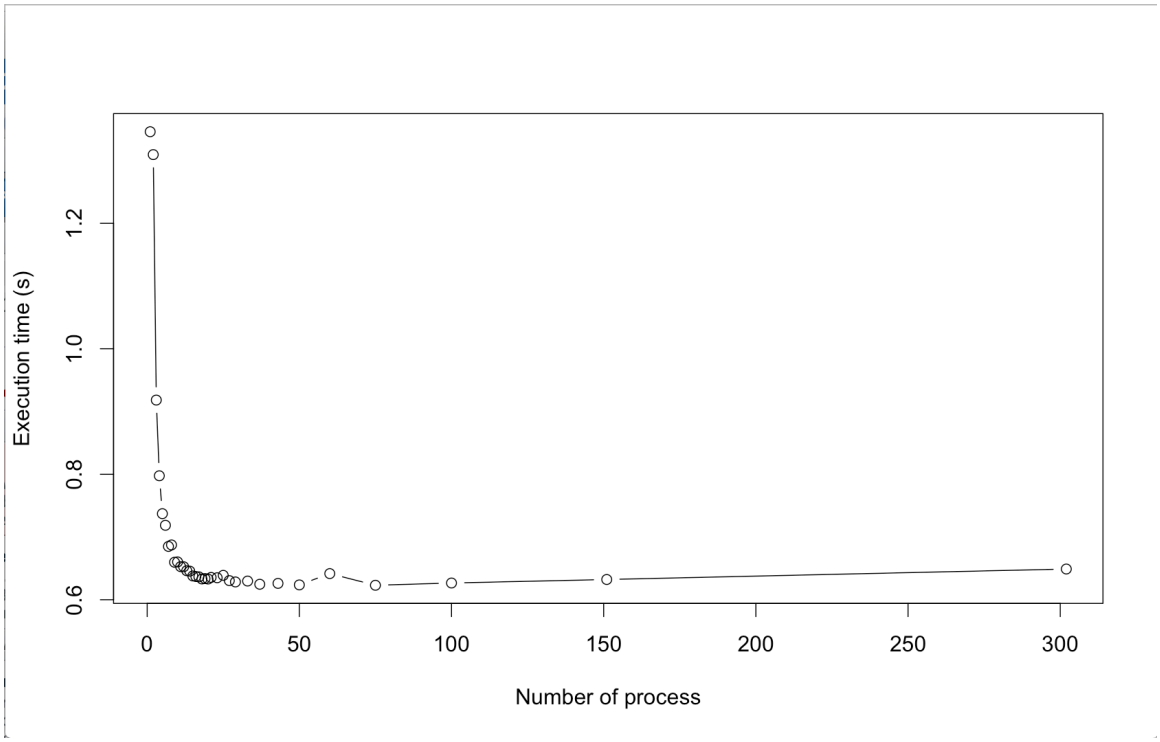
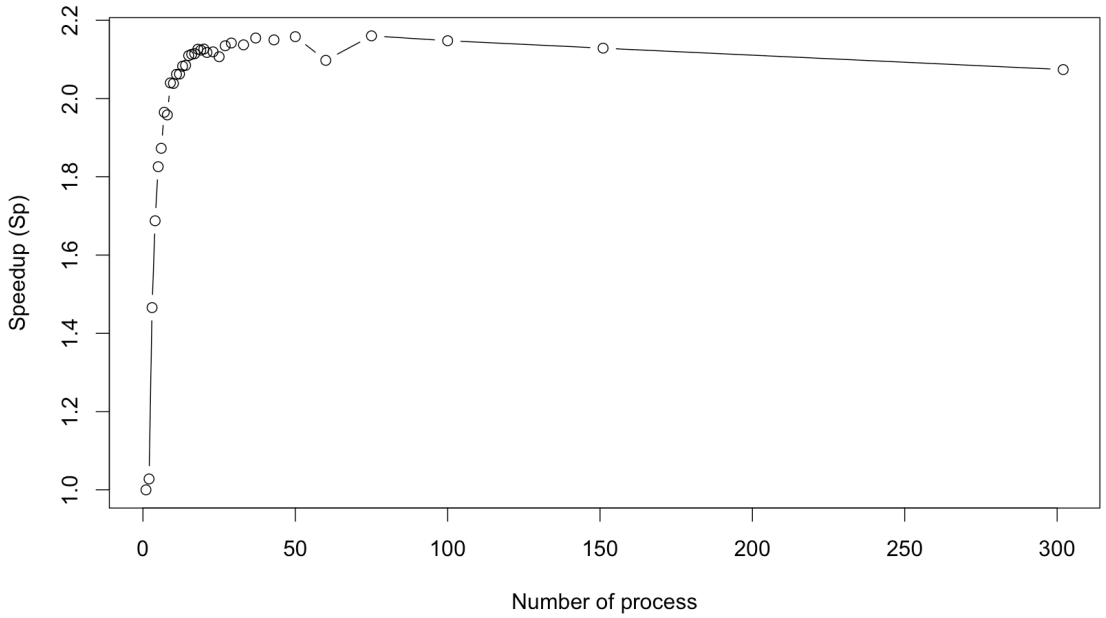
Experiment report

In order to experiment with process number, we run program ‘phist’ with all possible n (from 1 to 302, since there are only a total of 302 files) and record the execution time and corresponding process number n. We call this one experiment. Since for a given process number n, the running time may vary slightly each time we run it. Therefore, we conduct several experiments. Since the resulting graph are similar and the observation are the same. We use the data we obtained from the 1st experiment in this report.

After we run ‘phist’ with all possible n and adjust the data to reflect implementing of parallel computing in the code (for details see appendix), we get the following graph (Execution time versus number of process).



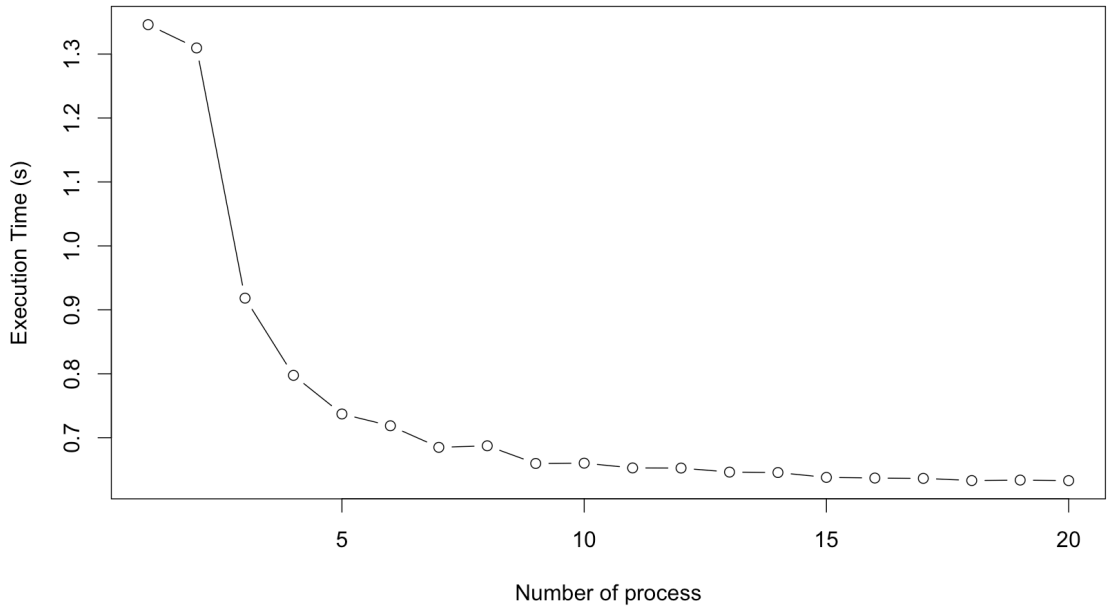
Plotting speedup against number of process, we have

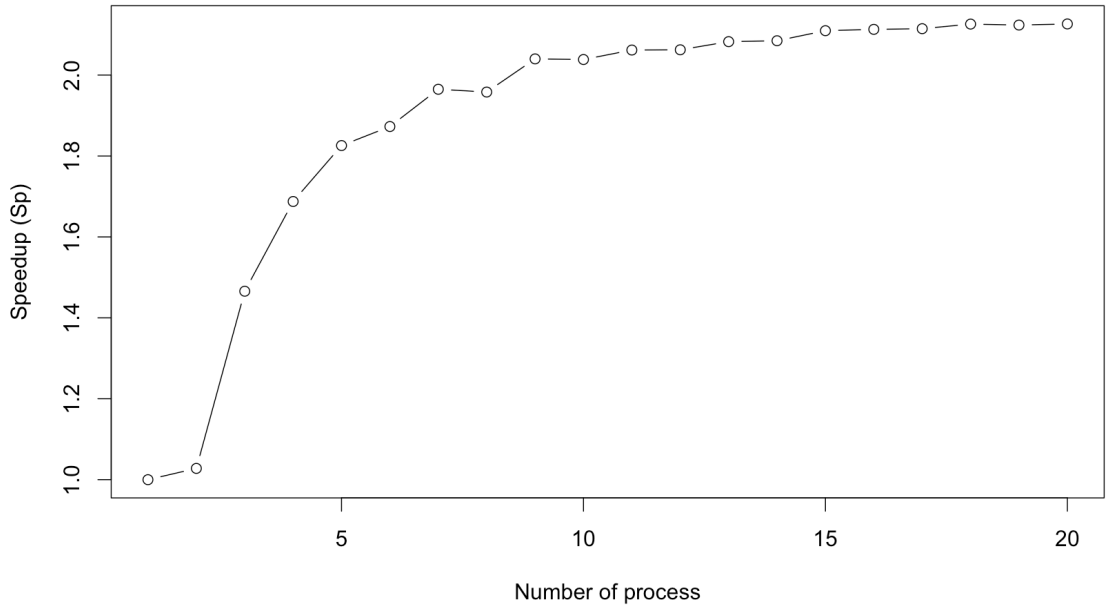


According to the graphs above, the execution time of the program first decreases sharply between 1 to 15 processes and it then exhibits a rather gradual upward tendency. The opposite is true for speedup versus number of processes.

Between 1 and 15, the speedup increases dramatically, which is within our expectation. Using multi-processes, we divide our problems into smaller pieces and pieces may be simultaneously processed by multiple processing units. Therefore, multi-processes can fully utilize the power of multi-processor of the computer. Since different processes can be simultaneously processed, the execution time is thus reduced. Even with single processor, multi-processes can increase the utilization of the CPU. For example, since the program are divided into multiple processes to work on smaller problems, when one processes are waiting for data from a file, then processor can give CPU resources to processes in need. Therefore, the running time will be improved. These are the main reason why execution time decreases and speedup increase drastically between 1 and 15.

Below are the close-up of the above graph between process number of 1 and 20.





From graph, we can see that after number of processes increase to 10, there is little change in the value of speedup and execution time. As we mentioned above, the execution time even exhibit an upward tendency later. The reason for this may be the number of processors in the computer is limited (to 2 in my case), the number of memory

space is limited and the I/O speeds are bounded. Creating processes is very expensive and takes time. For example, each process need separate memory space. Therefore, more processes increase the costs in this aspect. In order to complete the full task, processes need communication within themselves, which consumes times and resources. More processes means more communication costs. Additionally, usually switching between processes is necessary but also consumes time. When processes are greater than the number of processors, more processes will cause the processes fighting over CPU resources. Because processes will block each other, and the total time will be more than without parallel processes.

These costs in time and resources all increases as number of processes increase, which may cause running time increases. Therefore, the above gives the explanation of the behavior of the execution time and speedup in our graph. After 10 processes are introduced, more processes won't lead to more improvement of the running time. The execution time even tends to increase and speedup tends to decrease as the number of processes increase. It is also possible to have speedup below one due to the argument above, when time costs introduced by creating more processes are larger than the time it may save. However, this is cannot be seen in our graph. Since the maximum number of processes we can create is not large enough to cause slowdown of the program, the speedup is never below 1 in our case, but in reality such case can happen.

Appendix

When we run program with all possible n, we get the rough data and plot the following graph. The graph exhibits a clear pattern within each region, which is the execution decrease with number of processes. By analysis, we found that each region have the same integral value of number of files/ number of processes. This is caused by how the division of the files to each process is implemented in the code. In our code, when number of files doesn't divide the number of the processes, we take the floor of the quotient and assign the floor of the quotient number of files to each process except last one. The last one take however many number of files left. Hence, when we increase files in the region that has the same integral value of the quotient of number of files and number of processes. Even when the number of the processes only increased by one, the number of files processed by last processes decreases dramatically. Therefore, the runtime decreased dramatically due to this reason and conceal the effect of the process number increase, hence add noises to the data. Therefore, since the last number of processes in each region represent the n that divides most evenly across the process, if two number have the same quotient after dividing the number of files, we takes the larger one and its data in plotting the graph in the report.