Report

Introduction:

The aim of this project is to evaluate which algorithm is best between sequential and parallel algorithm in terms of speed up. Parallel algorithms are optimal if the time it takes to do a process is low. A significant amount of work is being done to develop efficient parallel algorithms for a variety of parallel architectures compared to the development of serial algorithms. We expect time efficiency to be better in parallel algorithms as we increase the number of threads to do a process relative to serial algorithm.

Methods:

The approach used to find a better solution in terms of speed up was to parallelize the serial algorithm, to allow division of tasks. The method used in this regard was Divide and Conquer algorithm using the Fork/Join Framework. This method, divides the tasks into smaller subtasks, allows threads to do left subtasks and the right subtasks are computed manually, at some point the algorithm will become serial, specifically when the difference between high and low bounds are at most the SEQUENTIAL_CUTOFF, then the solution is returned.

Approach used to test the correctness of the program was to take a series of input from a file, also changing the filter size and comparing the results between the SerialMedianFilter program and the ParallelMedianFilter program. The results must be the same, thus in that way, the program is correct. In general to test if both programs computed the correct output, I took a set of finite numbers and worked out the solution manually i.e. the expected solution using the paper based approach, then ran both the programs to observe if the solution is identical.

Method used to time my algorithm was the System.currentTimeMillis, but converted into seconds. Time is recorded after a completion of processes.

Input sizes ranging between 300 and 10000 and a change in SEQUENTIAL_CUTOFF were used. Time was taken for the same input sizes

between Serial and Parallel algorithms. Speed up was calculated by taking serial time dividing it by time on parallel algorithm, for every input size.

Machine architectures which I tested my code on were hp, core i3, Desktop and Lenovo, core i3 laptop. The findings were different, for instance when the program ran on the Desktop it was a bit faster than compared to when it was run on the laptop, hence it became a bit of a challenge to obtain the same speed up using these machines. The serial version was even slower on the laptop.

Results and Discussion:

Graphs and results are attached in Graphs.xls.

The results are evident that it is worth parallelizing the median filter problem as it appears that parallel algorithms tend to be efficient.

Between 500- 7000 data sizes, and corresponding filter sizes 13-21 the algorithm works well.

Maximum speed-up obtainable: 0.847112.

The expected speed up was below, approximately 3. The observed is less than the predicted but pretty close.

Optimal sequential cutoff= 1000.

Both Pc and Desktop have 4 processors, the optimal number of threads in this regard is approximately at most 2 threads per core.

Conclusion:

After doing all these tests I have discovered that parallelization of an algorithm can be useful in terms of performance. Not all the time, this assumption holds true because it may occur that a parallel algorithm runs slower than the serial algorithm for the same problem.

Another discovery made from this project is that the speed up of parallel algorithm depends mostly on the number of threads available, as well as cores on the machine.

My results are also evident that the parallel algorithm is efficient relative to the serial algorithm. The results obtained are fair, and were obtained by testing a multiple times, thus we can conclude that they are fair.