PHYS405

Advanced Computational Physics
Parallel Computing

Assignment No. 3
Due: Monday, October 31, 2016

Purpose: Learn to sort small and large list of objects using appropriate algorithms for sequential and parallel computers.

Reference: The course website. Google.

Extract from the course website the sequential codes that implement the Bubble sort and the Quick-Sort methods, as well as the skeleton of a method to merge two ordered lists to yield a single compound ordered (large) list.

Part A

Find the wall time dependence on the size of list of the Bubble sort and Quick-Sort methods. To do so, implement the MPIWtime() in these codes. Use lists (arrays filled with random numbers) of sizes: 4,000, 8,000, 15,000, 30,000, 60,000, 80,000, 120,000, 250,000, 500,000, 1 million objects. Do a fit of wall time as a function of the sizes of the lists.

Part B

To sort very large lists requires the use of parallel computers in order to satisfy the very large requirements in memory and CPU.

The first step consists in dividing the (large) list in sub-lists of about the same size that reside on the slave nodes.

The division in sub-lists of somewhat the same size will help balance the resources needed to divide and store the sub-lists. Homogeneous computers are easier to use here.

The sub-lists are then ordered, each one being ordered on/by its local node.

The individual sub-lists are then merged, two (2) sub-lists being merged simultaneously. In this way, the sub-lists are recombined, producing ever larger sub-lists. Eventually, the entire list is rebuild to its full size.

It is recommended to use a Master/Slave algorithm is this approach. Node 0 is then the master node that sets-up the problem (for instance the generation of the original list), administers the number of nodes, the overall logic, the flow of data, the use of the sequential ordering of the sub-lists in the nodes, and finally the ordered merging of the sub-lists to rebuild the original list. There must be an even number of slave nodes to load balance the merging.

You are asked to write the missing blocks of code to produce a working code for each of the sequential methods, the "Bubble sort" and the "Quick-sort" algorithm. Be careful to validate your parallel codes.

Part C

Repeat the calculations done in *Part A* for the same list sizes using your parallel codes.

Calculate a gain factor, i.e., the ratio of the wall time ratios between the sequential and the parallel codes. This factor measures the speed gain (or lost) of the parallel codes to the sequential implementations of the same methods.