Bucket Sort Parallel Performance

Phase 1 - Partitioning

Original unsorted array needs to be partitioned accorded to the number of processes. Each partition will have n/p data numbers

Two ways of partitioning the array across all the processes

- Partition array on the master process, and send each partition using a `scatter`
$$t_{\text{comm1}} = p\left(t_{\text{startup}} + \left(\frac{n}{p}\right) t_{\text{data}}\right)$$
- Broadcast all numbers and let each processor make the partition
$$t_{\text{comm1}} = t_{\text{startup}} + nt_{\text{data}}$$

The first method incurrs additional communication overhead of pt_{startup}, but the second method requires additional memory on the processes.

Phase 2 - Small bucket distribution

Each process will have n/p numbers. Each value is examined and put into the appropriate small bucket.

$$t_{\text{comp2}} = \frac{n}{p}$$

Phase 3 - Put in large buckets

- No computation here, just communication by `alltoall`
- Each small bucket will have, on average, n/p^2 numbers to be sent to each other process
- Computation time for one process to send each of its small buckets
$$t_{\text{comm,ideal}} = (p-1)\left(t_{\text{startup}} + \left(\frac{n}{p^2}\right) t_{\text{data}}\right)$$
- This is the deal case is where all processes can do this in parallel.
- Worst case is that all processes must do this sequentially

- Actual communication time will depend on things like the network topology, MPI implementation, architecture etc.
- Actual communication time will be somewhere between the lower and upper limits

$$t_{\text{comm,ideal}} \leq t_{\text{comm3}} \leq p t_{\text{comm,ideal}}$$

Phase 4 - Sort large buckets

All processes sort their respective large buckets in parallel. If quicksort is used:

$$t_{\text{comp4}} = \left(\frac{n}{p}\right) \log \left(\frac{n}{p}\right)$$