159.735 Assignment 2 - Semester 2, 2018

Parallel Bucket Sort

Write your own parallel implementation of the bucket sort routine. The purpose of this assignment is to study the partitioning strategy for parallelizing a problem and to analyze the performance. This is also an exercise in using MPI collective communication routines.

The master process takes as user input (from the command line) the size of the array of data numbers to sort and then creates an array and populates it with a set of random numbers. You should only use the master process to initialize the array. The master then needs to partition this array and use an appropriate collective communication routine to send each partition to the correct slave process. Each slave will maintain a set of its own buckets covering the full range of possible numbers and will implement the bucket sort routine on its own set of numbers. After this, the appropriate collective communication routine will then need to be called to send each small bucket to the large bucket on the corresponding process. Finally, the large buckets need to be sorted, and the contents gathered into the master process.

To get started, copy the program `bucket.c` from `mpi_examples`. This program provides a sequential implementation of bucket sort. The directory also has some example programs showing how to use the `scatter`, `gather`, and `alltoall` collective communication routines.

High marks will be awarded for those who use the collective communication routines for variable data buffer sizes, i.e.: `Gatherv`, `Alltoallv`, etc. Also high marks will be earned for those who perform a complete performance analysis of their program, and address Gustaffson's Law.

Submission

Please submit your C or C++ source code and provide a brief report of the performance you see.

Investigate the scalability of your program by running it on different numbers of processors. Do this in a manner that provides a way testing Gustaffson's Law. Present your results in an appropriate manner. Do you get the results you expect from Gustaffson's Law? If not, what is causing this? Are there any parts of the program that do obey Gustaffson's Law?

Due date: September 14, 2018.
This assignment is worth 20% of your final grade.

Appendix

Quicksort

The bucket sort algorithm essentially takes a set of values, partitions them, and then applies some sorting algorithm on each partition. It can be thought of as a "meta algorithm" in that it provides a way of partition data to use for any given sequential implementation of a sorting algorithm. A popular sorting algorithm is "quicksort" which is available in the C/C++ standard library as `qsort()`. For sorting N numbers, quicksort has an average time complexity of $O(N \log N)$. Compare this with the $O(N^2)$ time complexity of the "bubble sort" algorithm!

Note: you are not being asked to provide a parallel implementation of the quicksort algorithm. For this assignment you may use the library `qsort()` on your partitions.

Sequential bucket sort

The first phase in implementing bucket sort is to distribute them into the p buckets. This involves taking each number and dividing by p. The time complexity is $O(N)$. The second phase is to apply the sorting algorithm to the set of values in each partition. If quicksort is used, then the complexity on each partition is $O((N/p) \log(N/p))$. If each partition is treated sequentially then the complexity of this phase is then $O(N \log(N/p))$. This can be compared to $O(N \log N)$ for quicksort on the full set of values.

One obvious way of parallelizing the bucket sort algorithm is to partition according to the number of processors, and then assigning one processor per bucket. Why is this a bad idea?

For this assignment, you should implement the small bucket/large bucket approach. See the textbook slides on "Embarassingly parallel, partitioning, and divide-and-conquer".