MPI Homework (Due 11:59PM ET, 3/27)

Sorting and Matrices. This homework focuses on parallelization strategies and implementation. First you will do a quicksort using a hypercube. You can easily find MPI cheat sheets online to help you with this. Next you implement the Google PageRank algorithm with MPI. PageRank from your implementation point of view, is merely a matrix-vector multiplication. Just a very-very large matrix. As before, your submission must be your own (Your codes are checked on a neat online plagiarism checker for originality). Finally, I'm going to spend about 15mins on Mon the 18th discussing how you should report performance.

1. Quicksort using MPI (40 points)

1.1 Generate a list of random numbers (either on head node, or on all nodes)

1.2 Assume an n-D hypercube and sort them.

1.3 Make sure your code has testing/debugging modes so I have an easy way to determine that your code works. Possible strategies could include a debug mode that only works with 100 numbers and checks the 100 numbers to confirm that the list is sorted. You need to have such strategies if you were doing this in a production environment

1.4 Organize your code as follows with the following checks:

```
PreSortList = Create_Random_List(N_list)
PreSortSum = Sum(PreSortList)
T_start = time()
     SortedList = ParallelQuickSort(PreSortList, N_proc)
T_stop = time()
PostSortSum = Sum(PostSortList)
SortCheck = checkPostSort(PostSortList)
ReportPerformance(N_list, N_proc, T_start, T_stop)
```

checkPostSort → Walk through your sorted list and confirm that List[N] <= List[N+1]

2 Parallelize the PageRank algorithm with MPI (60 points)

2.1 Use the web-graphs from the Stanford Large Network Dataset Collection (http://snap.stanford.edu/data/index.html#web)

- *You can have the head node read and parse the file and initialize your matrices. Alternatively, you can write a separate program (in say python) to read and transform your data, create the various transition matrices, create the sparse matrix representations and write those to file. Then your parallelized version can work with this file.*

2.2 Realize that your matrices are very sparse, so be smart about it. CSR maybe?? Also, do you think it makes sense to communicate the entire PageRank vector.

2.3 Compute the PageRank until they converge OR run about 50 iterations

2.4 For evaluation, I would suggest implementing a serial version (will also help you understand PageRank) and running a small dataset on it. If it helps, you can do this in python.

2.5 Your submission will be your code and a **report**[*] that includes:

- Name of the dataset that you worked on
- Your parallelization strategy and optimizations
- Your testing and evaluation strategy.
- List your Top100 nodes in a file called **top100.txt**
- Report your performance — Think Scalability, speedup, number of processors and dataset size.
- Finally, while you can use the entire cluster, I would suggest no more than 64cores. However, if the entire class can self-organize, then you can use more nodes. But wait until the last few days to do such large runs. By then you should have a good idea about the time your job will take, as well as it's efficiency.

Useful Links for PageRank

- http://www.ams.org/samplings/feature-column/fcarc-pagerank
- https://towardsdatascience.com/graphs-and-paths-pagerank-54f180a1aa0a
- https://www2.cs.duke.edu/csed/principles/pagerank/
  - Something weird going on with this page where it has a hard time rendering the last three pages. Not important pages since they just describe eigenvalues and eigenvectors. But here is a pdf (without the interactivity https://www.dropbox.com/s/3i1bqrv2c4shta4/PageRank%20Explained%20with%20Javascript.pdf?dl=0)
- http://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm

[*] Format: pdf; Length: 2 pages; Single-spaced; Font Arial; Size 11-14; 0.5” margins