Handing in your homework: Please hand in your homework as described in Assignment #1, i.e., by creating a git repository containing a Makefile, and by sending me the path to that repo. I will discuss the solution to Problem 1 in the class on March 30, so please submit your solutions before the class. The git repository https://github.com/NYU-HPC15/homework2.git contains the examples needed for this homework. For an overview over the MPI functions available in the MPI Standard 3.0, you can consult the official MPI documentation at http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf.

1. Finding MPI bugs. The above mentioned repository contains the files mpi_bug1.c, mpi_bug2.c, ..., mpi_bug7.c. These example codes contain bugs, resulting in hangs or other undesirable behavior. Try to find these bugs and fix them. Add a short comment to the code describing what was wrong and how you fixed the problem. Add the solutions to your repository using the naming convention mpi_solved{1,2,...}.c. Each problem should be run with 4 MPI tasks.

2. Parallel sample sort. Each of the $P$ processors creates an $N$-vector of random numbers. The target is to sort the union of all these distributed vectors; this union, let’s call it $v$, consists of $PN$ numbers and is assumed to be too large to fit into the memory of a single processor—thus, a serial sort algorithm cannot be used. The goal is to sort $v$ such that every processor roughly holds about $N$ sorted numbers, say $v_i$, and that the elements on the processor with rank $i$ are all smaller than those on the processor with rank $i + 1$ (for all $i = 0, 1, \ldots, P - 2$). The above repository contains a stub called ssort.c, which also contains an outline of the algorithm. For a summary of the sample sort algorithm, see the Wikipedia entry\(^1\) for sample sort, as well as the pages linked under “References” from there. The main steps of sample sort are:

- **Select local samples:** Each of the $P$ processors selects a set of $S$ random entries and communicates these entries to the root processor, who sorts the resulting $SP$ entries and determines $P - 1$ splitters $\{S_1, \ldots, S_{P-1}\}$, which are broadcasted.

- **Distribute to buckets:** Each processor determines the “buckets” to which each of its $N$ elements belong; for instance, the first bucket contains all the numbers $\leq S_1$, the second one are all the entries that are in $(S_1, S_2]$ and so on. The numbers contained in each bucket are then communicated; processor 0 receives every processor’s first bucket, processor 1 gets processor’s second bucket, and so on.

- **Local sort:** Each processor uses a local sort and writes the result to disc.

Include the MPI rank in the filename (see the example pingpong_array.c example file). Run your implementation of the sorting algorithm on at least 64 cores of Stampede, and present timings depending on the number of $N$ of elements to be sorted per processor\(^2\).

---

\(^1\)http://en.wikipedia.org/wiki/Samplesort

\(^2\)I will show how to submit jobs to Stampede in our next class, and add an example jobscript to the repository.
3. **Pitch your final project.** Summarize your final project in 1-2 paragraphs. I assume that you have already talked to me about your plans and have gotten my OK for your plans. Detail *what* you are planning to do, and with *whom* you will be cooperating.