# CIS 5930-04 Parallel Computing

**Spring 2007**

## Assignment 2

**Due: 6 Apr 2007**

**Problem:** In this assignment, you will implement four parallel sorting algorithms using MPI.

**Parallelization:** Implement the following four parallel sorting algorithms using MPI for communication.

1. *Parallel Quicksort* from section 14.3.
2. *Hyperquicksort* from section 14.4.
3. *Parallel Sorting by Regular Sampling* from section 14.5.
4. One other (comparison based) parallel sorting algorithm – you are free to choose this.

**Files:** You should create a directory called *hw2* on the p690 at NCSA, containing the following files:

1. *sort1.c*, implementing the code for #1 above.
2. *sort2.c*, implementing code for #2 above.
3. *sort3.c*, implementing code for #3 above.
4. *sort4.c*, implementing the code for #4 above. The top few lines of this code should have a comment specifying the name of the algorithm used, and the reference where you found it. You may make changes to the algorithm given in your reference, in order to make it run faster, especially on large output.
5. *sort.c*, implementing the `main` function. This code should read command line arguments provided by the user, call the appropriate sorting routine (as explained below), and print the desired output (exactly as specified – there should be no extra output).
6. *Makefile*, which is the makefile for this assignment. Typing `make` should create an executable called *sort*, which is created by linking *sort.o, sort1.o, sort2.o, sort3.o*, and *sort4.o*.
7. *sort*, which is the executable that I will run.

**Running the code:** Code will be run as follows:

```
poe sort <a> <n> <s> [<optional list of integers>] –procs <np>
```

In the above line, <*np*> refers to the number of processors. The algorithm to use for sorting will be specified by <*a*>; <*a*> = *1* for the method *1*, *2* for method *2*, *3*, for method *3*, and *4* for method *4*. The total number of (long) integers to sort is specified by <*n*>, which will be at least *1*. You should generate the random integers as follows. Use `srand48` to seed the random number generator using <*s*> (which is a long integer) as the seed. Then generate <*n*> random integers using `lrand48`. (Remember to include the header file *stdlib.h*.) The <*n*> integers should initially be divided by having each processor get *floor((double)*

<*n*>/<*np*>*)* or *ceil((double)* <*n*>/<*p*>*)* integers each. Then call the sorting algorithm, and time its execution using `MPI_Wtime`. Then output the time taken by your algorithm, in seconds, using "*%g*" in the output format. If an optional list of integers is provided in the command line, then each integer will be in the range *[0, <n>-1]*. Let those integers be $x_1$, $x_2$, ..., $x_k$. You should output the $x_1$ th smallest integer in the input, the $x_2$ th smallest integer in the input, ..., the $x_k$ th smallest integer in the input, each on a separate line, in the order specified.

Example: `poe sort 1 5 8 2 1 3 –procs 2`

The random numbers generated by `srand48` will be as follows, using *8* as the seed: *294734626, 830272356, 8452830, 1276961749, 1036785092*. The program will run parallel quicksort on two processors. The output will be as follows (the time taken may differ – everything else should be exactly as given here).

```
Time = 2.86102e-5 s
294734626
8452830
830272356
```

**Code organization:** The basic code (in *sort.c*) should be organized as follows.

- Read command line arguments.
- Call `srand48(<s>)`
- Generate the desired samples. You may either generate them on one process, and send them to others, or generate them directly on different processes. You need to be careful in the latter case, to ensure that the correct random numbers are generated on each process.
- Call `MPI_Barrier`
- Call `MPI_Wtime`
- Run the desired parallel sorting algorithm
- Call `MPI_Barrier`
- Call `MPI_Wtime`
- Output the time taken on process *0*.
- Perform any output required by the optional command line arguments.

**Turning in your assignment:** All your files should have a time stamp before midnight, 6 Apr 2007. Please have read and execute permissions unset until at least midnight 8 Apr 2007. Read permission should be set on by noon 9 Apr 2007.

**Grading:** You will be graded on the following (i) correctness of your codes, (ii) performance of your codes, especially relative to those of others in the class, and (iii) coding style. Good performance on method *4* is especially important. In the other methods, your implementation should not perform considerably worse than is possible. In method *4*, your grade for performance will depend on how it compares with the best in the class, for large input sizes.