Brent Clapp, Donivan Anderson

[Email address]

Project 2

Quick Sort

Running PQsort

1. Log in to the school’s system
2. Type “/usr/lib64/openmpi/bin/mpicc name
3. Then type “/usr/lib64/openmpi/bin/mpiexec -n X a.out
   1. X is equal to the number of processors that would like to use.
4. Now just wait until the program finishes. The result of PQsort will be displayed.

PQsort function

To start off, PQsort starts at the root and we determine how many elements each processor receives. We also load the sizes of each processor into send\_count and then the displacement of each processor is stored in displacements. Both send\_count and displacements need to have storage allocated to them with a size determined by how many processors there are.

MPI\_Scatter is executed as MPI\_Scatter(send\_count, 1, MPI\_INT, &recCnt, 1, MPI\_INT, root, MPI\_COMM\_WORLD) so we can receive the size of each processor (returned as recCnt). With that data we allocate space for our new array localarray and it has the size of recCnt (receive count). Now we can call MPI\_Scatterv(elements, send\_count,displacements, MPI\_INT, localArray, recCnt, MPI\_INT, 0, MPI\_COMM\_WORLD) so that we can give each process the (possibly varying) data. Afterwards, we can select the pivot by choosing a random process and from within that process we choose a random number and broadcast that number to all processors. With the pivot and the array distributed, we can begin the partitioning. Each process calls the function partition and its job is to locate any digit less than or equal to the pivot by keeping an index value moving through the array and one index value at the end of the numbers that are less than or equal to the pivot. When partition does locate a target, the function swap is called to simply swap the two index values so that we keep the smaller values on the left end of the array.

Testing

There are a few base cases that we used to test the function. They will show how the program partitions the given array as well as can it sort the array.

Let processors = 5.

Try processors = elements (1 to 1). This will show that the program can separate the arrays onto the Li and Si.

Try 2\*processors = elements (2 to 1). This will show that the program can separate the arrays as well as the processors are partitioning their arrays.

Try 3\*processors = elements (3 to 1). This further show that the program can separate the arrays as well as the processors are partitioning their arrays.

Now to prove that PQsort works, we will use an array of size 100 with processor count of 1, 4, 16, and 64.

1 Processor

4 Processors

16 processors

64 processors