CSCE 435 Spring 2020

HW 4: Parallel Programming with MPI – Hypercube Quicksort

Due: 11:59pm Thursday, April 9, 2020

You are provided with a program `qsort_hypercube.c` that implements the parallel quicksort algorithm on a hypercube using MPI. At the start of the algorithm, an unsorted list of elements is distributed equally across all processes. At the end of the algorithm, the sorted list is distributed across processors in order of their rank, i.e., the elements on processor P_j are no larger than elements on P_k, for k > j. In addition, the elements on a processor are themselves in a sorted list. It is not necessary for the sorted list to be distributed equally across the processors.

The `main` routine is missing **nine** calls to MPI routines that are labeled MPI-1, MPI-2, …, MPI-9. You are required to add these calls to the code to obtain a functioning program. The location of the missing MPI calls is marked by the following text:

```
// ***** Add MPI call here *****
```

The text is preceded by comments that describe what the call should do.

Once you complete the code, it can be compiled with the command:

```
mpiicc -o qsort_hypercube.exe qsort_hypercube.c
```

Setting VERBOSE to a value of 2 or 3 will result in additional output that could help in code development. To execute the program, use

```
mpirun –np <p> ./qsort_hypercube.exe <n> <type>
```

where <p> is the number of MPI processes, <n> is the size of the local list of each MPI process, and <type> is the method used to initialize the local list. The initialization routine is available in `qsort_hypercube.h`. The output of a sample run is shown below.

```
mpirun –np 8 ./qsort_hypercube.exe 6000000 0

[Proc: 0] number of processes = 8, initial local list size = 6000000,
hypercube quicksort time = 0.780582

[Proc: 0] Congratulations. The list has been sorted correctly.
```

1. (80 points) Complete the MPI-based code provided in `qsort_hypercube.c` to implement the parallel quicksort algorithm for a d-dimensional hypercube with $p=2^d$ processors. 60 points will be awarded if the code compiles and executes the following command successfully.

   ```
   mpirun -np  2 ./qsort_hypercube.exe 4 -1
   ```

   5 points will be awarded for each of the following tests that are executed successfully.

   ```
   mpirun -np  4 ./qsort_hypercube.exe 4 -2
   mpirun -np  8 ./qsort_hypercube.exe 4 -1
   mpirun -np  16 ./qsort_hypercube.exe 4 0
   mpirun -np  16 ./qsort_hypercube.exe 20480000 0
   ```

2. (5 points) *Weak Scalability Study*: Run your code to sort a distributed list of size np where n is the size of the local list on each process and p is the number of processes. For your experiments,

use n=20,480,000 and p = 1, 2, 4, 8, 16, 32, and 64. Set type=0. Plot the execution time, speedup, and efficiency of your code as a function of p. Use logarithmic scale for the x-axis.

3. (5 points) *Strong Scalability Study:* Now run your code with n=20,480,000/p where p = 1, 2, 4, 8, 16, 32, and 64. Set type=0. Plot the execution time, speedup, and efficiency of your code as a function of p. Use logarithmic scale for the x-axis.
4. (10 points) Modify the code to sort the list in descending order. Submit the modified code as `qsort_hypercube_descending.c`. 2 points will be awarded for each of the tests in Problem 1 that are executed successfully. (Note that the `check_list` routine in `qsort_hypercube.h` needs to be modified to verify descending order.)

Submission: You need to upload the following to eCampus as a **single zip file**:

1. Problem 1: Submit the file `qsort_hypercube.c`.
2. Problem 2-3: Submit a single PDF or MSWord document with your response.
3. Problem 4: Submit the file `qsort_hypercube_descending.c`.

Helpful Information:

1. Source file(s) are available on the shared Google Drive for the class.
2. You may use either ADA or TERRA for this assignment. Indicate which machine was used in your experiments.
3. Load the intel software stack prior to compiling your program. Use:
   ```
   module load intel/2017A
   ```
4. Compile MPI programs using `mpiicc`. For example, to compile `code.c` to create the executable `code.exe`, use
   ```
   mpiicc –o code.exe code.c
   ```
5. The run time of a code should be measured when it is executed in dedicated mode. Create a batch file and submit your code for execution via the batch system on the system.