Sergi Aliaga

Northeastern University

Homework 4

Question 2

Develop a parallel histogramming program using C/C++ and OpenMPI. A histogram is used to summarize the distribution of data values in a data set. The most common form of histogramming splits the data range into equal-sized bins. For each bin, the number of data values in the data set that falls into that class are totaled. Your input to this program will be integers in the range 1-1,000,000 (use a random number generator that first generates the numbers first). Your input data set should contain 2 million integers. You will vary the number of bins. You should have as many OpenMPI processes as bins. You are suggested to use the sample batch script provided on Canvas for specifying your OpenMPI configuration and running your program. Make sure to use the express partition, versus the short partition.

a) *Assume there are 100 bins. Perform binning across nodes and processes using OpenMPI, and then perform a reduction on the lead node, combining your partial results. Run this on 2 and 4 nodes on Discovery. Your program should print out the number of values that fall into each bin. Compare the performance between running this on 2 and 4 nodes. Comment on the differences*

The code used in this question is included in files "Q2a.c". To run the script with the different node configuration, 2 and 4, the scripts "Q2a_2nodes.script" and "Q2a_4nodes.script" have been used, respectively. To test that the code is working properly, some initial results have been obtained from histogramming 2000 integers ranging from 0 to 100, in 4 bins (that is using 2 nodes and 2 tasks per node, using the bash script in "Q2a_test.script"). Results are reported in Figure 1.

```
[aliaga.s@login-00 Question2]$ more Q2a_2nodes.out
RESULTS from process 1 at node c0161:
[1.000000 - 25.750000)  [25.750000 - 50.500000) [50.500000 - 75.250000) [75.250000 - 100.000000)
125     110     134     131
RESULTS from process 2 at node c0162:
[1.000000 - 25.750000)  [25.750000 - 50.500000) [50.500000 - 75.250000) [75.250000 - 100.000000)
132     123     134     111
RESULTS from process 0 at node c0161:
[1.000000 - 25.750000)  [25.750000 - 50.500000) [50.500000 - 75.250000) [75.250000 - 100.000000)
122     121     135     122
RESULTS from process 3 at node c0162:
[1.000000 - 25.750000)  [25.750000 - 50.500000) [50.500000 - 75.250000) [75.250000 - 100.000000)
134     110     120     136
GLOBAL RESULTS:
[1.000000 - 25.750000)  [25.750000 - 50.500000) [50.500000 - 75.250000) [75.250000 - 100.000000)
513     464     523     500
```

Figure 1: Histogram results for 2000 elements in 4 processes.

For evaluating the performance of the code, a set of 10 experiments for each configuration, 2 and 4 nodes respectively, have been performed. Results are reported in Figure 2. We observe that the code is significantly faster in the 4 node configuration with respect to the 2 node one, with a **41% speedup** in average. This might be caused by the fact that the bottleneck is in the computation of the histogram, rather than the communication between nodes and, therefore, using more nodes is still faster despite the communication overhead between them.

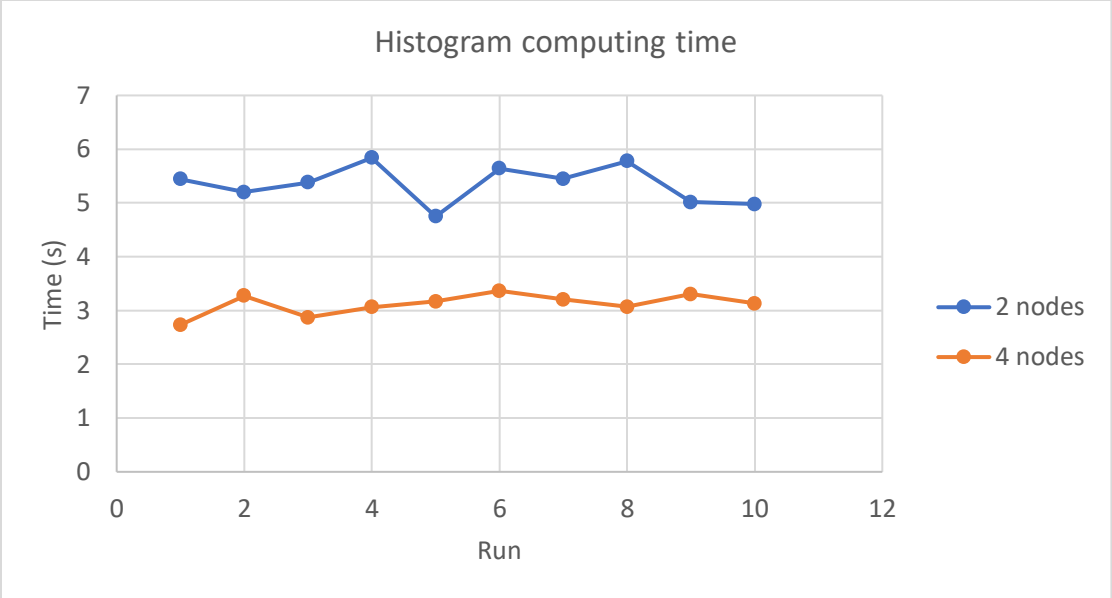

Figure 2: Computing time for calculating the histogram of 2e6 numbers between 1 and 1e6 with 100 bins (and 100 processes).

b) *For this part, assume you have 10 bins. Perform binning on each process using OpenMPI, and then perform a reduction on the lead node, combining your partial results. Run this on 2 and 4 nodes on Discovery. Your program should print out the number of values that fall into each bin. Compare the performance between running this on 2 and 4 nodes. Comment on the differences.*

For this section of the assignment the code in "Q2a.c" has been slightly modified and saved in a new file called "Q2b.c". This version of the code creates the histogram with just 10 bins. It also includes some control sequences in case the number of processes are higher than the number of bins (in the case of 4 nodes, the minimum required tasks per node are 3, which results into 12 processes. Two of these processes could stop the program if not handled correctly, in this case, staying idle while the rest of the processes compute the histogram). A similar test with just 2000 numbers between 1 and 100 has been performed to confirm the code works. Results are reported in Figure 3.

```
[aliaga.s@login-00 Question2]$ more Q2b_4nodes.out
RESULTS from process 1 at node c0161:
[1.0 - 10.9)    [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
15      20      23      22      23      12      29      19      19      18
RESULTS from process 2 at node c0161:
[1.0 - 10.9)    [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
14      17      16      19      19      28      28      19      24      16
RESULTS from process 3 at node c0162:
[1.0 - 10.9)    [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
22      21      23      16      17      21      24      18      18      20
RESULTS from process 4 at node c0162:
[1.0 - 10.9)    [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
16      30      19      20      15      18      24      21      23      14
RESULTS from process 5 at node c0162:
[1.0 - 10.9)    [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
20      22      23      19      22      16      24      12      23      19
RESULTS from process 6 at node c0163:
[1.0 - 10.9)    [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
25      21      14      24      14      26      14      30      15      17
RESULTS from process 7 at node c0163:
[1.0 - 10.9)    [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
23      14      22      19      21      25      18      24      13      21
RESULTS from process 8 at node c0163:
[1.0 - 10.9)    [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
17      27      RESULTS from process 0 at node c0161:
22      RESULTS from process 9 at node c0164:
10[1.0 - 10.9)  [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
18      22      20      16      15      20      19      22      21      27
        [1.0 - 10.9)20          [10.9 - 20.8)10         24[20.8 - 30.7)         16[30.7 - 40.6)         29[40.6 - 50.5)         25[50.5 - 60.4)
[60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
21      29      18      11      21      24      17      20      19      20
GLOBAL RESULTS:
[1.0 - 10.9)    [10.9 - 20.8)   [20.8 - 30.7)   [30.7 - 40.6)   [40.6 - 50.5)   [50.5 - 60.4)   [60.4 - 70.3)   [70.3 - 80.2)   [80.2 - 90.1)   [90.1 - 100.0)
191     223     200     176     187     200     221     201     204     197
Total computing time: 0.116280
```

Figure 3: Histogram results for 2000 elements in 10 processes.

Once the proper operation of the code is confirmed, the same 10 run test for each configuration has been performed, using files “Q2b_2nodes.script” and “Q2b_4nodes.script” respectively. Results are reported in Figure 4. This time not only we observe a similar performance for both configurations but even a slightly worse performance in the case of 4 nodes. In this scenario, each process only has to go through at most 10 bins for each of the 2 milion values to find its bin. This means that the computational burden on each process is much lower and the bottleneck this time is in the communications between nodes. This bottleneck is reflected in an average **7% difference** in computing time for the 4 node case.

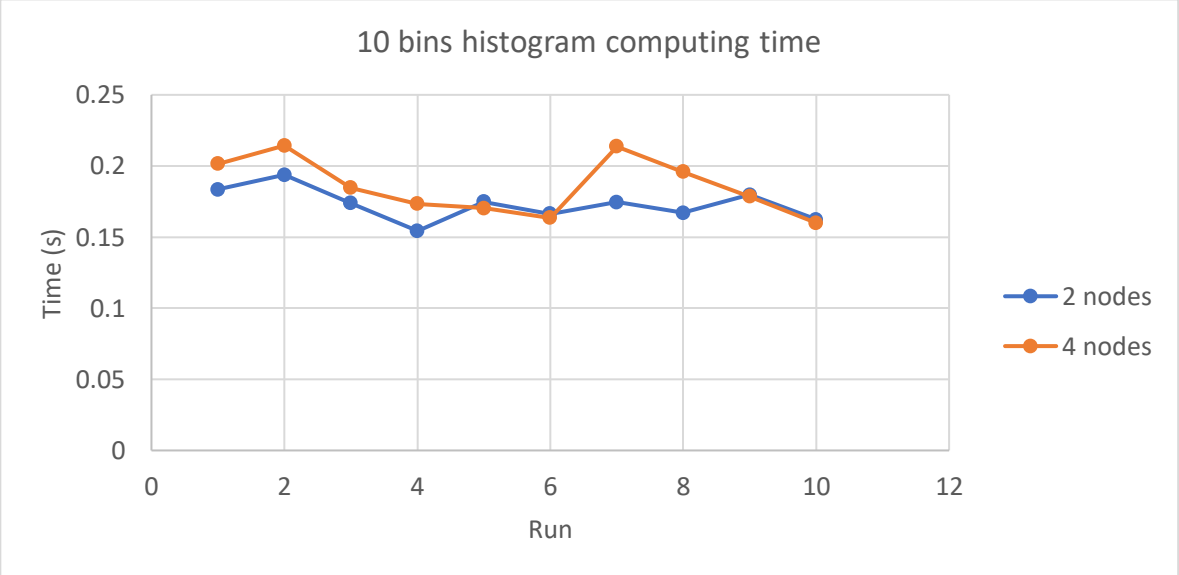

Figure 4: Computing time for calculating the histogram of 2e6 numbers between 1 and 1e6 with 10 bins (and 10 processes).

c) *Compare the performance measured in parts a.) and b.). Try to explain why one is faster than the other and run additional experiments to support your claims.*

As explained, in part a) each process has to find the bin for a smaller amount of numbers, since there are more processes than in part b). However, finding the bin of each number means checking at most 100 bins, which results in a larger computational burden than the 10 bin scenario. For this reason, in part a) we can observe that the bottleneck is in the actual binning at each process and therefore, more nodes result into a significantly better performance. On the other hand, in part b) the bottleneck resides in the communication between nodes since the computational burden is much lower for 10 bins. This results into a better performance in the case of 2 nodes.