Exam 2: Applying Foster's Methodology

Author: Gabriel Hofer

Course: CSC-410 Parallel Programming

Instructor: Dr. Karlsson

Due: November 23, 2020

Computer Science and Engineering
South Dakota School of Mines and Technology

Buidling the Histogram

Partitioning

We will partition the array into $n/chunk\_size$ subsections where n is the size of the array of floats and the parameter $chunk\_size$ is the number of floats per subsection of the original array. If n is not evenly divisible by $chunk\_size$, the remaining subsection will have fewer elements.

Each subsection of the array is assigned a task. And, each task corresponds to a unique bin in the histogram. Moreover, task i is assigned to bin j such that $i = j$. See Figure 1.

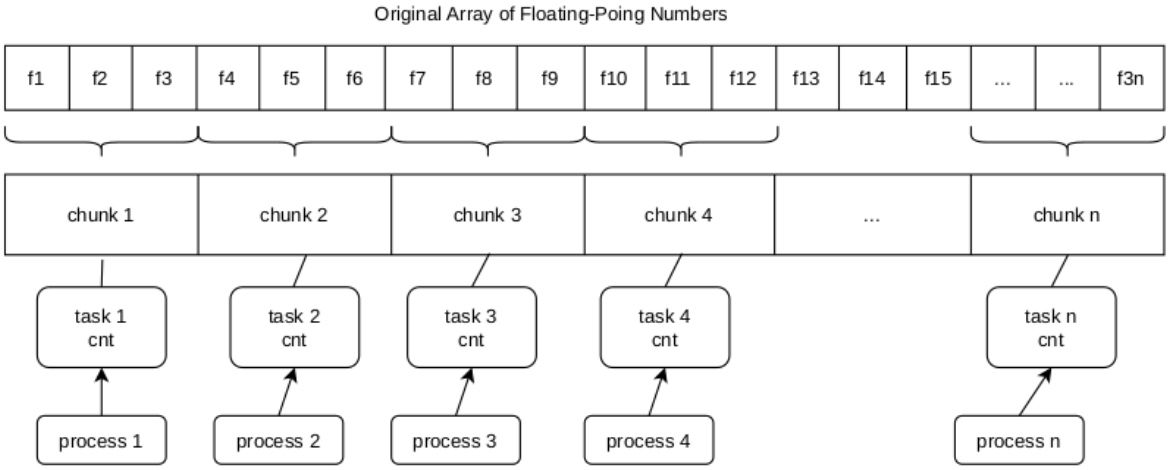

Figure 1: Data Partitioning
An arbitrary number of floating point numbers in the original array are grouped together to form a chunk. Each task has a chunk of the original array assigned to it. Finally, Each processor has its own task.

Before processing the data, each task has a local variable, bin_cnt, which is initialized to zero. The local variable bin_cnt in the i^{th} task will store the number of floating-point numbers in the i^{th} bin.

When the original array of all floating-point numbers is initially partitioned into smaller subsections, the floating-point numbers in an array subsection don't necessarily 'belong' in the bin associated with their task number. This is because, the floats are initially, unsorted and unorganized. And the floating-point numbers are still unsorted after partitioning. Therefore, we need communication in order to 'move' the floating-point numbers to the right tasks/bins.

Communication

After the partitions have been created, each task will iterate through its own subsection. For any floating-point number f in the subsection, the task calculates which bin it belongs to. The correct bin is calculated by dividing the floating-point number by the bin_width using integer division:

$$target\_bin = f/bin\_width$$

where

$$bin\_width = (max\_float - min\_float)/(n/chunk\_size)$$

Basically, the bin_width is determined by the number of array chunks/partitions and the range of the floats in the global array. After traversing through the whole sub-array, the task sends a message to all bins. The messages contain the amount that each non-local task should update their local bin_cnt variable. See Figure 2.

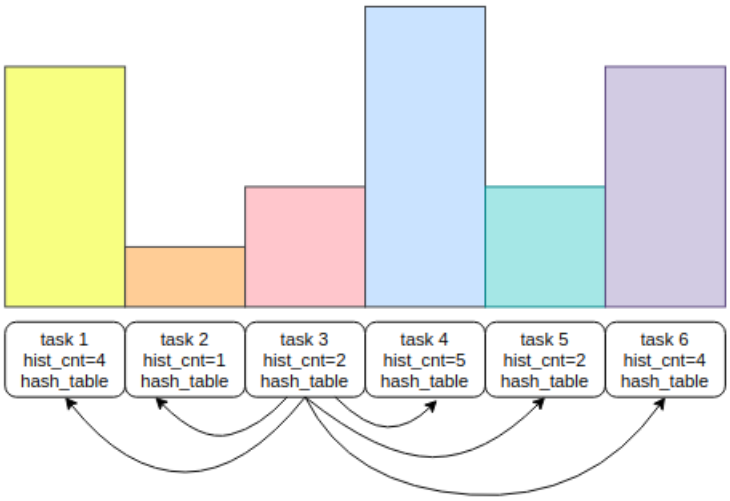

Figure 2
This figure illustrates how the cnt variable in each task is used to construct a histogram. Secondly, the arrows are used to show that tasks send messages to every other task to tell them how much to add to their count (cnt) variable. In the figure, only the send-messages from task 3 are shown.

Agglomeration

We could have created a task/process for each floating-point number in the array. This is actually possible if the $chunk\_size$ is set to 1. However, it's preferable to choose a larger value for bin_width because it will reduce the number of messages sent between processes. Notice that there will be $O(chunk\_size^2)$ messages sent between tasks for updating each task's bin_cnt. Also, the point of the histogram is to combine data points to show groups of similar data. A histogram with so many bins is less likely to be useful.

Mapping

Since the number of tasks and the number of processors are the same, we simply assign task t to a process p such that $t = p$. This mapping was illustrated in Figure 1.

Combining Data in the Root Process

The root task will construct a plot from the bin_cnt of each task. For this to happen, the data needs to be retrieved and sent to the root. Once, each task has iterated through its own array, they will ping the root message, indicating that the task has processed all of its floating point numbers and that all messages to other processors have been received. Once the root process has been pinged this message from all of the processors/tasks, then the root will send back a pong to all of the tasks. The purpose of this pong is to tell every task to send their 'bin_cnt' variable to the root task. The reason we need to do a ping-pong before each task simply sends its bin_cnt variable to the root is that we need to make sure that every message has been completed (i.e. synchronized) before we send the bin_cnt to the root.

Extending our approach to the Stem-and-Leaf Plot

We will extend the method for parallelizing the histogram to the stem-and-leaf plot. We use the same parameter $chunk\_size$ to partition the array into smaller subsections. However, there are two major modifications.

Hash Tables

First, each task will maintain a hash-table to store its partition/subsection of the stem-and-leaf plot. The hash table will map keys (stems) to a list of values (leaves). The keys are the integer part of the floating-point numbers while the list of values contain the fractional part of the floating-point numbers.

Communication Revisited

Secondly, each task will send messages to every other task. But, instead of only sending an integer amount to add to non-local bin_cnt variables, each message will contain a list of the actual floating-point numbers, so that each non-local task can process the numbers they receive from other tasks and add them to their own local hash-table. (We could done this (sent the floats themselves) with the histogram, but it was more memory efficient to just send an integer that would update the bin_cnt.) In the Stem-and-Leaf plot a single task may have between 0 and n unique stems. A stem will only exist if there is at least one leaf. See Figure 3.

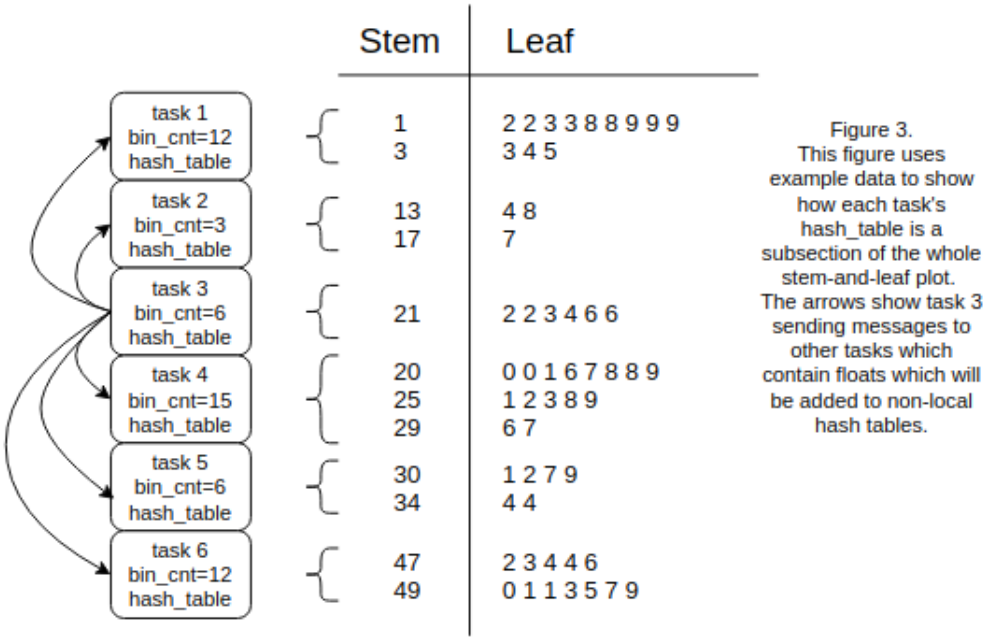

Figure 3. This figure uses example data to show how each task's hash_table is a subsection of the whole stem-and-leaf plot. The arrows show task 3 sending messages to other tasks which contain floats which will be added to non-local hash tables.

Conclusion

Now, after writing this all up, I realize that it would have been much easier to extend the method we used to make the Stem-and-Leaf plot to the Histogram. Once all of the floats have been sent to their appropriate tasks, we simply have to count the number of floats in a task to calculate bin_cnt for a bin in the histogram. Furthermore, we would convert a Stem-and-Leaf plot to a Histogram by putting all of the stems/leafs in the same task into the same bin.