CSE 6220 Introduction to High Performance Computing
Spring 2019
Homework 3 solutions

1. Draw a 8-element bitonic sorting circuit to sort elements in non-increasing order. Use horizontal lines for the numbers and vertical lines to denote comparison operations. Label the comparison operations as ↑ or ↓ where the direction of the arrowhead indicates the destination of the smaller element. Illustrate how the input $15, 2, 7, 4, 17, 3, 9, 1$ is sorted using the diagram.

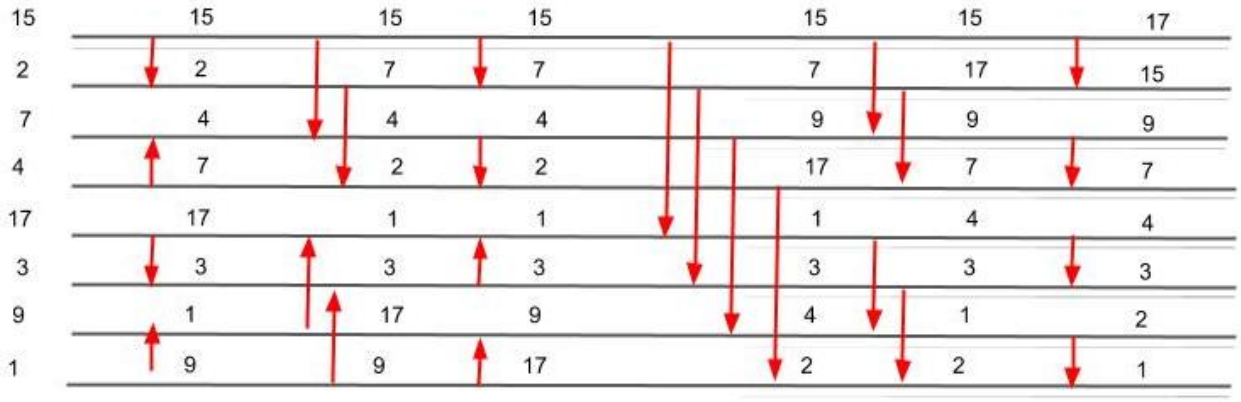

Figure 1: Bitonic Sort Diagram

2. The *Bitonic Split* operation defined in class assumes that the bitonic sequence has an even length. Extend this operation to bitonic sequences of odd length. Now, show how the bitonic sequence $5, 3, 4, 7, 10, 14, 17, 13, 8$ can be converted into a sorted sequence using repeated bitonic split operations.

Solution:

The idea is that whenever we have an odd length bitonic sequence to split, we can duplicate the last element and append it to the current sequence. We also mark the appended element so that it is easily removable later. We now have an even length sequence which is still bitonic, and hence a split operation can be performed. We can remove the marked element after the split has been performed. The given sequence can be sorted as follows:

5, 3, 4, 7, 10, 14, 17, 13, 8
↓
5, 3, 4, 7, 10, 14, 17, 13, 8, **8**
↓ split
5, 3, 4, 7, **8**, 14, 17, 13, 8, 10

$\downarrow$

5, 3, 4, 7 | 14, 17, 13, 8, 10

$\downarrow$

5, 3, 4, 7 | 14, 17, 13, 8, 10, **10**

$\downarrow$ split

4, 3, 5, 7 | 8, 10, **10**, 14, 17, 13

$\downarrow$

4, 3 | 5, 7 | 8, 10, | 14, 17, 13

$\downarrow$

4, 3 | 5, 7 | 8, 10, | 14, 17, 13, **13**

$\downarrow$ split

3, 4 | 5, 7 | 8, 10, | 13, **13**, 14, 17

$\downarrow$

3 | 4 | 5 | 7 | 8 | 10 | 13 | 14, 17

$\downarrow$ split

3 | 4 | 5 | 7 | 8 | 10 | 13 | 14 | 17

Note that this problem asks you to extend *Bitonic Split* operation that works on bitonic sequence. You cannot add a very large/small number like INT_MAX/INT_MIN or infinity to the sequence since after adding that element the sequence is no longer bitonic.

3. Give an algorithm to merge two sorted sequences of lengths m and n, respectively.You may assume that the input is an array of length $m + n$ with one sequence followed by the other, distributed across processors such that each processor has a subarray of size $\frac{m+n}{p}$. What are the computation and communication times for your algorithm? (You may assume the use of any permutation-style communication, not necessarily hypercubic.)

   Solution:

   Note that reversing the second sequence will result in a bitonic sequence. This bitonic sequence can be sorted using bitonic merge. Reversing the second sequence would take $O(\tau + \mu n/p)$ time. Bitonic merging of a sequence of length $m + n$ using p processors takes $O((\tau + \mu(m + n)/p) \log p)$ time.

4. Let **k-to-all broadcast** be a communication operation in which k processors have a message of size m each and each message should be broadcast to all p processors. Give an algorithm for this operation that runs in $O\left(\tau \log p + \mu mk \left(\log\left(\frac{p}{k}\right) + 1\right)\right)$ time. Prove the running time. Assume that k is a power of 2, $1 \leq k \leq p$. The k processors that have messages to broadcast can be any arbitrary processors.

Solution:

Given p processors, we can think of k of them sending a message of size m, and $p-k$ of them sending a message of size 0. With that, an `All-Gather` operation can achieve k**-to-all broadcast**. Note that the maximum message size at any stage of the All-Gather operation will not exceed $k \times m$.

Therefore, the runtime of the `All-Gather` operation in this case is bounded by

$$\begin{aligned}
&\mathcal{O}\left(\tau \log p + \mu m \underbrace{(1+2+2^2+\ldots+\frac{k}{2})}_{\log k \text{ values}} + \mu m \underbrace{(k+k+\ldots+k)}_{\log p - \log k \text{ values}}\right) \\
=&\mathcal{O}\left(\tau \log p + \mu m(k) + \mu m k(\log p - \log k)\right) \\
=&\mathcal{O}\left(\tau \log p + \mu m k(\log\left(\frac{p}{k}\right)+1)\right)
\end{aligned}$$

5. Consider a tree of n nodes and having a bounded degree (i.e., the number of children per node is bounded by a constant). The tree is stored in an array A of size n. Each node also contains the indices at which its parent and children are stored in the array. The array is distributed among processors using block decomposition, i.e., P_i contains $A[i\frac{n}{p}], A[i\frac{n}{p}+1], \ldots, A[(i+1)\frac{n}{p}-1]$. For each of the following operations, which communication primitive will you use and what is the runtime?

   (a) Each node in the tree has $O(1)$ sized data that should be sent to its parent.

   (b) Each node in the tree has $O(1)$ sized data that should be sent to all its children.

Solution:

Use `many-to-many` communication primitive in both cases. Each processor has $\frac{n}{p}$ tree nodes. Let d denote the maximum number of children per tree node, s and r denote the maximum possible size of a message that'd be sent and received by a processor.

(a) $s \leq \frac{n}{p}$, $r \leq d\frac{n}{p}$

(b) $s \leq d\frac{n}{p}$, $r \leq \frac{n}{p}$

In both cases, runtime would be the same:

$$\text{Hypercubic Permutation} = O\left(\tau \log p + \mu \frac{n}{p} \log p\right)$$

$$\text{Arbitrary Permutation} = O\left(\tau p + \mu \frac{n}{p}\right)$$