Lecture 12: Parallel Sorting

Shantanu Dutt
ECE Dept.
UIC
Acknowledgement

- Adapted from Chapter 9 slides of the text, by A. Grama w/ a few changes, augmentations and corrections
Topic Overview

• Issues in Sorting on Parallel Computers
• Bitonic Sorting Networks
• Bitonic Sorting on Hypercubes and 2-D Meshes
Sorting: Overview

- One of the most commonly used and well-studied kernels.
- Sorting can be *comparison-based* or *non-comparison-based*.
- The fundamental operation of comparison-based sorting is *compare-exchange*.
- The lower bound on any comparison-based sort of $n$ numbers is $\Theta(n \log n)$.
- We focus here on comparison-based sorting algorithms.
What is a parallel sorted sequence? Where are the input and output lists stored?

- We assume that the input and output lists are distributed.
- The sorted list is partitioned with the property that each partitioned list is sorted and each element in processor $P_i$'s list is less than that in $P_j$'s list if $i < j$. 
A parallel compare-exchange operation. Processes $P_i$ and $P_j$ send their elements to each other. Process $P_i$ keeps $\min\{a_i, a_j\}$, and $P_j$ keeps $\max\{a_i, a_j\}$. 
Sorting: Basics

What is the parallel counterpart to a sequential comparator?

- If each processor has one element, the compare exchange operation stores the smaller element at the processor with smaller id. This can be done in $t_s + t_w$ time.

- If we have more than one element per processor, we call this operation a compare split. Assume each of two processors have $n/p$ elements.

- After the compare-split operation, the smaller $n/p$ elements are at processor $P_i$ and the larger $n/p$ elements at $P_j$, where $i < j$.

- The communication time for a compare-split operation is $(t_s + t_w n/p)$, and assuming that the two partial lists were initially sorted, the computation time is $\Theta(n/p)$—essentially, the time to merge two sorted lists.
A compare-split operation. Each process sends its block of size $n/p$ to the other process. Each process merges the received block with its own block and retains only the appropriate half of the merged block. In this example, process $P_i$ retains the smaller elements and process $P_i$ retains the larger elements.
Sorting Networks

- Networks of comparators designed specifically for sorting.
- A comparator is a device with two inputs $x$ and $y$ and two outputs $x'$ and $y'$. For an increasing comparator, $x' = \min\{x,y\}$ and $y' = \min\{x,y\}$; and vice-versa for a decreasing comparator.
- We denote an increasing comparator by $\oplus$ and a decreasing comparator by $\ominus$.
- The speed of the network is proportional to its depth.
A schematic representation of comparators: (a) an increasing comparator, and (b) a decreasing comparator.
A typical sorting network. Every sorting network is made up of a series of columns, and each column contains a number of comparators connected in parallel.
A bitonic sorting network sorts $n$ elements in $\Theta(\log^2 n)$ time.

A bitonic sequence has two tones - increasing and decreasing, or vice versa. Any cyclic rotation of such sequences is also considered bitonic.

$\langle 1,2,4,7,6,0 \rangle$ is a bitonic sequence, because it first increases (1 to 7) and then decreases (7 to 0).
$\langle 8,9,2,1,0,4 \rangle$ is another bitonic sequence, because it is a cyclic shift of $\langle 0,4,8,9,2,1 \rangle$.

The kernel of the network is the rearrangement of a bitonic sequence into a sorted sequence.
Sorting Networks: Bitonic Sort

• Let $s = \langle a_0, a_1, ..., a_{n-1} \rangle$ be a bitonic sequence such that $a_0 \leq a_1 \leq \cdots \leq a_{n/2-1}$ and $a_{n/2} \geq a_{n/2+1} \geq \cdots \geq a_{n-1}$.

• Consider the following subsequences of $s$:

  - $s_1 = \langle \min\{a_0, a_{n/2}\}, \min\{a_1, a_{n/2+1}\}, \ldots, \min\{a_{n/2-1}, a_{n-1}\} \rangle$
  - $s_2 = \langle \max\{a_0, a_{n/2}\}, \max\{a_1, a_{n/2+1}\}, \ldots, \max\{a_{n/2-1}, a_{n-1}\} \rangle$

(1)

• Note that $s_1$ and $s_2$ are both bitonic and each element of $s_1$ is less than every element in $s_2$: Why? Need to prove—there can only be $\leq 1$ “break points” in these two sequences (if starting from two sorted seqs)

• We can apply the procedure recursively on $s_1$ and $s_2$ to get the sorted sequence.
**Sorting Networks: Bitonic Sort**

Merging a 16-element bitonic sequence through a series of $\log 16$ bitonic splits.

<table>
<thead>
<tr>
<th>Original sequence</th>
<th>3</th>
<th>5</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>20</th>
<th>95</th>
<th>90</th>
<th>60</th>
<th>40</th>
<th>35</th>
<th>23</th>
<th>18</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Split</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>0</td>
<td>95</td>
<td>90</td>
<td>60</td>
<td>40</td>
<td>35</td>
<td>23</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>2nd Split</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>0</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>9</td>
<td>35</td>
<td>23</td>
<td>18</td>
<td>20</td>
<td>95</td>
<td>90</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td>3rd Split</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>10</td>
<td>9</td>
<td>14</td>
<td>12</td>
<td>18</td>
<td>20</td>
<td>35</td>
<td>23</td>
<td>60</td>
<td>40</td>
<td>95</td>
<td>90</td>
</tr>
<tr>
<td>4th Split</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>18</td>
<td>20</td>
<td>23</td>
<td>35</td>
<td>40</td>
<td>60</td>
<td>90</td>
<td>95</td>
</tr>
</tbody>
</table>
Sorting Networks: Bitonic Sort

• We can easily build a sorting network to implement this bitonic merge algorithm.

• Such a network is called a *bitonic merging network*.

• The network contains $\log n$ columns. Each column contains $n/2$ comparators and performs one step of the bitonic merge.

• We denote a bitonic merging network with $n$ inputs by $\oplus\text{BM}[n]$.

• Replacing the $\oplus$ comparators by $\ominus$ comparators results in a decreasing output sequence; such a network is denoted by $\ominus\text{BM}[n]$.
A bitonic merging network for $n = 16$. The input wires are numbered 0, 1, ..., $n - 1$, and the binary representation of these numbers is shown. Each column of comparators is drawn separately; the entire figure represents a $\oplus\text{BM}[16]$ bitonic merging network. The network takes a bitonic sequence and outputs it in sorted order.
Sorting Networks: Bitonic Sort

How do we sort an unsorted sequence using a bitonic merge?

- We must first build a single bitonic sequence from the given sequence.
- A sequence of length 2 is a bitonic sequence.
- A bitonic sequence of length 4 can be built by sorting the first two elements using $\oplus BM[2]$ and next two, using $\ominus BM[2]$.
- This process can be repeated to generate larger bitonic sequences.
A schematic representation of a network that converts an input sequence into a bitonic sequence. In this example, $\oplus\text{BM}[k]$ and $\ominus\text{BM}[k]$ denote bitonic merging networks of input size $k$ that use $\oplus$ and $\ominus$ comparators, respectively. The last merging network ($\oplus\text{BM}[16]$) sorts the input. In this example, $n = 16$. 
The comparator network that transforms an input sequence of 16 unordered numbers into a bitonic sequence.
Sorting Networks: Bitonic Sort

- The depth of the network is $\Theta(\log^2 n)$.
- Each stage of the network contains $n/2$ comparators. A serial implementation of the network would have complexity $\Theta(n \log^2 n)$. 
Mapping Bitonic Sort to Hypercubes

- Consider the case of one item per processor. The question becomes one of how the wires in the bitonic network should be mapped to the hypercube interconnect.

- Note from our earlier examples that the compare-exchange operation is performed between two wires only if their labels differ in exactly one bit!

- This implies a direct mapping of wires to processors. All communication is nearest neighbor!
Mapping Bitonic Sort to Hypercubes

Communication during the last stage of bitonic sort. Each wire is mapped to a hypercube process; each connection represents a compare-exchange between processes.
Communication characteristics of bitonic sort on a hypercube. During each stage of the algorithm, processes communicate along the dimensions shown.
Mapping Bitonic Sort to Hypercubes

Parallel formulation of bitonic sort on a hypercube with \( n = 2^d \) processes.

```plaintext
procedure BITONIC_SORT(label, d)
begin
    Augment d-bit labels to (d+1)-bit by concatenating a 0 at (d+1)-bit posn. of all d-bit labels;

    for i := 0 to d - 1 do
        for j := i downto 0 do
            if \((i + 1)^{st}\) bit of label \(!=\) \(j^{th}\) bit of label then
                comp_exchange_max(j);
            else
                comp_exchange_min(j);

end BITONIC_SORT

/* The above condition stems from the more basic condition: if \((i+1)\)'th bit is 0 (1) then need to perform increasing (decreasing) order sort, and thus if my j’th bit (current exchange dimension) is 1, I need to keep the max (min), and if 0, need to keep min (max) */
```
Mapping Bitonic Sort to Hypercubes

- During each step of the algorithm, every process performs a compare-exchange operation (single nearest neighbor communication of one word).
- Since each step takes $\Theta(1)$ time, the parallel time is

$$T_p = \Theta(\log^2 n)$$

(2)

- This algorithm is cost optimal w.r.t. its serial counterpart, but not w.r.t. the best sorting algorithm.
Mapping Bitonic Sort to Meshes

- The connectivity of a mesh is lower than that of a hypercube, so we must expect some overhead in this mapping.

- Consider the row-major shuffled mapping of wires to processors.
Different ways of mapping the input wires of the bitonic sorting network to a mesh of processes: (a) row-major mapping, (b) row-major snakelike mapping, and (c) row-major shuffled mapping.
Mapping Bitonic Sort to Meshes

The last stage of the bitonic sort algorithm for $n = 16$ on a mesh, using the row-major shuffled mapping. During each step, process pairs compare-exchange their elements. Arrows indicate the pairs of processes that perform compare-exchange operations.
Mapping Bitonic Sort to Meshes

• In the row-major shuffled mapping, wires that differ at the $i^{th}$ least-significant bit are mapped onto mesh processes that are $2^{\lfloor (i-1)/2 \rfloor}$ communication links away.

• The total amount of communication performed by each process is $\sum_{i=1}^{\log n} \sum_{j=1}^{i} 2^{\lfloor (j-1)/2 \rfloor} \approx 7\sqrt{n}$, or $\Theta(\sqrt{n})$. The total computation performed by each process is $\Theta(\log^2 n)$.

• The parallel runtime is:

$$T_P = \Theta(\log^2 n) + \Theta(\sqrt{n}).$$

• This is not cost optimal.
Block of Elements Per Processor

- Each process is assigned a block of \( \frac{n}{p} \) elements.
- The first step is a local sort of the local block.
- Each subsequent compare-exchange operation is replaced by a compare-split operation.
- We can effectively view the bitonic network as having \( (1 + \log p)(\log p)/2 \) steps.
Block of Elements Per Processor: Hypercube

- Initially the processes sort their $n/p$ elements (using merge sort) in time $\Theta((n/p)\log(n/p))$ and then perform $\Theta(\log^2 p)$ compare-split steps.

- The parallel run time of this formulation is

\[ T_P = \Theta\left(\frac{n}{p} \log \frac{n}{p}\right) + \Theta\left(\frac{n}{p} \log^2 p\right) + \Theta\left(\frac{n}{p} \log^2 p\right). \]

- Comparing to an optimal sort, the algorithm can efficiently use up to $p = \Theta(2^{\sqrt{\log n}})$ processes (derived by obtaining $p$ as a function of $n$ for a constant efficiency).

- The isoefficiency function due to both communication and extra work is $\Theta(p \log p \log^2 p)$ (derived by obtaining $n$ as a function of $p$ for a constant efficiency).
Block of Elements Per Processor: Mesh

- The parallel runtime in this case is given by:
  \[ T_P = \Theta \left( \frac{n}{p} \log \frac{n}{p} \right) + \Theta \left( \frac{n}{p} \log^2 p \right) + \Theta \left( \frac{n}{\sqrt{p}} \right) \]

- This formulation can efficiently use up to \( p = \Theta(\log^2 n) \) processes.
- The isoefficiency function is \( \Theta(2\sqrt{p} \sqrt{p}) \).
Performance of Parallel Bitonic Sort

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Maximum Number of Processes for $E = \Theta(1)$</th>
<th>Corresponding Parallel Run Time</th>
<th>Isoefficiency Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypercube</td>
<td>$\Theta(2^{\sqrt{\log n}})$</td>
<td>$\Theta(n/(2^{\sqrt{\log n}}) \log n)$</td>
<td>$\Theta(p^{\log p} \log^2 p)$</td>
</tr>
<tr>
<td>Mesh</td>
<td>$\Theta(\log^2 n)$</td>
<td>$\Theta(n/ \log n)$</td>
<td>$\Theta(2^{\sqrt{p}} \sqrt{p})$</td>
</tr>
<tr>
<td>Ring</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(2^p p)$</td>
</tr>
</tbody>
</table>

The performance of parallel formulations of bitonic sort for $n$ elements on $p$ processes.