Parallel Sorting
Chapter 9

Contents

• General issues
• Sorting network
  – Bitonic sort
• Bubble sort and its variants
  – Odd-even transposition
• Quicksort
• Other Sorting Algorithms
General Issues in Sorting

• Data location
  – Internal: data in memory
  – External: data on disk
  – Sequential sorting is usually internal, but could be external
  – Parallel sorting could have data in one process or distributed
  – We will focus on distributed data

• Comparisons or not
  – Sorting can be based on comparisons or not
  – Comparison based sorting requires $\Theta(n \log n)$ sequential time
  – Noncomparison based sorting can be faster
  – Fundamental operation is compareexchange (sort 2 elements)

Parallel Compare Exchange Operation

• Compare-Exchange
• One element per process
• Adjacent process exchange data
• One keeps minimum, and the other keeps maximum

![Diagram of compare-exchange operation]

**Figure 9.1** 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 $\max\{a_i, a_j\}$. 

Dr. Joe Zhang

PDC-14: Parallel Sorting
Parallel Compare Split Operation

- Compare-split
- Multiple elements per process
- Adjacent processes exchange data
- One keeps minimum half, and the other keeps maximum half

![Diagram of compare-split operation]

**Figure 9.2** 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 \( P_2 \) retains the smaller elements and process \( P_1 \) retains the larger elements.

Dr. Joe Zhang

PDC-14: Parallel Sorting

---

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 \( y' = \max\{x,y\} \); and vice-versa.
- We denote an increasing comparator by \( \oplus \) and a decreasing comparator by \( \ominus \).
Sorting Networks: Comparators

Figure 9.3 A schematic representation of comparators: (a) an increasing comparator, and (b) a decreasing comparator.

Sorting Networks

Figure 9.4 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.
**Bitonic Sequence**

- A bitonic sequence can be circularly shifted to yield an increasing subsequence followed by a decreasing subsequence.
  - \(\langle 1,2,4,7,6,0 \rangle\) is a bitonic sequence because it first increases and then decreases.
  - \(\langle 8,9,2,1,0,4 \rangle\) is another bitonic sequence, because it is a cyclic shift of \(\langle 0,4,8,9,2,1 \rangle\).

**Bitonic Split and Merge**

- Let \(s = \langle a_0, a_1, \ldots, a_{n-1} \rangle\) 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 \(s\):
    - \(s_1 = \langle \min\{a_0, a_{n/2}\}, \min\{a_1, a_{n/2+1}\}, \ldots, \min\{a_{n/2-1}, a_{n-1}\} \rangle\)
    - \(s_2 = \langle \max\{a_0, a_{n/2}\}, \max\{a_1, a_{n/2+1}\}, \ldots, \max\{a_{n/2-1}, a_{n-1}\} \rangle\)
  - Note that \(s_1\) and \(s_2\) are both bitonic and each element of \(s_1\) is less than every element in \(s_2\).
  - Bitonic split
    - The operation of splitting a bitonic sequence of size \(n\) into two bitonic sequences defined in (1)
  - Bitonic merge
    - The procedure of sorting a bitonic sequence using bitonic splits recursively
Merging Bitonic Sequence

Original sequence:

3 5 8 9 10 12 14 20 95 90 60 40 35 23 18 0

1st Split:

3 5 8 9 10 12 14 0 95 90 60 40 35 23 18 20

2nd Split:

3 5 8 0 10 12 14 9 35 23 18 20 95 90 60 40

3rd Split:

3 0 8 5 10 9 14 12 18 20 35 23 60 40 95 90

4th Split:

0 3 5 8 9 10 12 14 18 20 23 35 40 60 90 95

Figure 9.5  Merging a 16-element bitonic sequence through a series of $\log 16$ bitonic splits.

Bitonic Merging Networks

• 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 (depth $n$). Each column contains $n/2$ comparators and performs one step of the bitonic merge.

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

• Replacing the $\oplus$ comparators by $\Theta$ comparators results in a decreasing output sequence; such a network is denoted by $\Theta BM[n]$. 
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.
**Bitonic Sorting Networks**

**Figure 9.7** A schematic representation of a network that converts an input sequence into a bitonic sequence. In this example, ⊕BM[k] and ⊖BM[k] denote bitonic merging networks of input size \( k \) that use \( ⊕ \) and \( ⊖ \) comparators, respectively. The last merging network (⊕BM[16]) sorts the input. In this example, \( n = 16 \).

---

**Bitonic Sorting Networks**

**Figure 9.8** The comparator network that transforms an input sequence of 16 unordered numbers into a bitonic sequence. In contrast to Figure 9.6, the columns of comparators in each bitonic merging network are drawn in a single box, separated by a dashed line.
Sorting Networks: Bitonic Sort

- The depth of a bitonic merging network with n input is $\Theta(\log n)$
- The depth of the sorting network is $\Theta(\log^2 n)$
  - Implemented in parallel
- Each stage of the network contains $n/2$ comparators.
- A serial implementation of the network would have complexity $\Theta(n \log^2 n)$.

Bubble Sort and its Variants

- The complexity of bubble sort is $\Theta(n^2)$.
- Bubble sort is difficult to parallelize since the algorithm has no concurrency.
- A simple variant, though, uncovers the concurrency.

1. procedure BUBBLE_SORT(n)
2. begin
3. for $i := n - 1$ downto 1 do
4. for $j := 1$ to $i$ do
5. compare-exchange($a_j, a_{j+1}$);
6. end BUBBLE_SORT
Odd-Even Transposition

Unsorted

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>3</th>
<th>8</th>
<th>5</th>
<th>6</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
</table>

Phase 1 (odd)

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>3</th>
<th>8</th>
<th>5</th>
<th>6</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
</table>

Phase 2 (even)

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>3</th>
<th>5</th>
<th>8</th>
<th>1</th>
<th>6</th>
<th>4</th>
</tr>
</thead>
</table>

Phase 3 (odd)

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>3</th>
<th>5</th>
<th>1</th>
<th>8</th>
<th>4</th>
<th>6</th>
</tr>
</thead>
</table>

Phase 4 (even)

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>3</th>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>6</th>
</tr>
</thead>
</table>

Phase 5 (odd)

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
</table>

Phase 6 (even)

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
</table>

Phase 7 (odd)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
</table>

Phase 8 (even)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
</table>

Sorted

Figure 9.13: Sorting $n = 8$ elements, using the odd-even transposition sort algorithm. During each phase, $n = 8$ elements are compared.

Sequential odd-even transposition sort algorithm.

Each phase of the algorithm requires $\Theta(n)$ comparisons.

Serial complexity is $\Theta(n^2)$.

Dr. Joe Zhang

PDC-14: Parallel Sorting
Parallel Odd-Even Transposition

1. procedure \textsc{Odd-Even.Par}(n)
2. begin
3. \hspace{1em} id := process’s label
4. \hspace{1em} for $i := 1$ to $n$ do
5. \hspace{2em} begin
6. \hspace{3em} if $i$ is odd then
7. \hspace{4em} if $id$ is odd then
8. \hspace{5em} compare-exchange\_min(id + 1);
9. \hspace{4em} else
10. \hspace{5em} compare-exchange\_max(id - 1);
11. \hspace{3em} if $i$ is even then
12. \hspace{4em} if $id$ is even then
13. \hspace{5em} compare-exchange\_min(id + 1);
14. \hspace{4em} else
15. \hspace{5em} compare-exchange\_max(id - 1);
16. end for
17. end \textsc{Odd-Even.Par}

Algorithm 9.4 The parallel formulation of odd-even transposition sort on an $n$-process ring.

Dr. Joe Zhang

PDC-14: Parallel Sorting

Parallel Odd-Even Transposition

- Consider the one item per processor case.
- There are $n$ iterations, in each iteration, each processor does one compare-exchange.
- The parallel run time of this formulation is $\Theta(n)$.
- This is cost optimal with respect to the base serial algorithm but not the optimal one.
Parallel Odd-Even Transposition

- Consider a block of \( n/p \) elements per processor.
- The first step is a local sort.
- In each subsequent step, the compare exchange operation is replaced by the compare split operation.
- The parallel run time of the formulation is

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

- The parallel formulation is cost-optimal for \( p = O(\log n) \).
- The isoefficiency function of this parallel formulation is \( \Theta(p^{2p}) \).

Quicksort

- Quicksort
  - One of the most common sorting algorithms for sequential computers because of its simplicity, low overhead, and optimal average complexity.
- How does it work?
  - Quicksort selects one of the entries in the sequence to be the pivot and divides the sequence into two - one with all elements less than the pivot and other greater.
  - The process is recursively applied to each of the sublists.
- Performance
  - The performance of quicksort depends critically on the quality of the pivot.
  - The average complexity of quicksort is \( O(n \log n) \).
Quicksort

For a sequence $A[q, \ldots, r]$

1. procedure QUICKSORT($A, q, r$)
2. begin
3. if $q < r$ then
4. begin
5. $x := A[q]$;
6. $s := q$;
7. for $i := q + 1$ to $r$ do
8. if $A[i] \leq x$ then
9. begin
10. $s := s + 1$;
11. swap($A[s], A[i]$);
12. end if
13. end if
14. swap($A[q], A[s]$);
15. QUICKSORT($A, q, s$);
16. QUICKSORT($A, s + 1, r$);
17. end QUICKSORT

Algorithm 9.5 The sequential quicksort algorithm.

Dr. Joe Zhang PDC-14: Parallel Sorting

Quicksort

(a) \[
\begin{array}{ccccccc}
3 & 2 & 1 & 5 & 8 & 4 & 3 & 7 \\
\end{array}
\]

(b) \[
\begin{array}{ccccccc}
1 & 2 & 3 & 5 & 8 & 4 & 3 & 7 \\
\end{array}
\]

(c) \[
\begin{array}{ccccccc}
1 & 2 & 3 & 3 & 4 & 5 & 8 & 7 \\
\end{array}
\]

(d) \[
\begin{array}{ccccccc}
1 & 2 & 3 & 3 & 4 & 5 & 7 & 8 \\
\end{array}
\]

(e) \[
\begin{array}{ccccccc}
1 & 2 & 3 & 3 & 4 & 5 & 7 & 8 \\
\end{array}
\]

Figure 9.15 Example of the quicksort algorithm sorting a sequence of size $n = 8$. 

Dr. Joe Zhang PDC-14: Parallel Sorting
QuickSort Algorithm on a Hypercube

1. procedure HYPERCUBE.QUICKSORT (B, n)
2. begin
3.   id := process’s label;
4.   for i := 1 to d do
5.     begin
6.       x := pivot;
7.       partition B into B1 and B2 such that B1 \leq x < B2;
8.       if i\textsuperscript{th} bit is 0 then
9.         begin
10.          send B2 to the process along the i\textsuperscript{th} communication link;
11.          C := subsequence received along the i\textsuperscript{th} communication link;
12.          B := B1 \cup C;
13.       end
14.     else
15.         send B1 to the process along the i\textsuperscript{th} communication link;
16.         C := subsequence received along the i\textsuperscript{th} communication link;
17.         B := B2 \cup C;
18.       endelse
19.     endfor
20.   sort B using sequential quicksort;
21. end HYPERCUBE.QUICKSORT

Algorithm 9.9 A parallel formulation of quicksort on a d-dimensional hypercube. B is the n/p-element subsequence assigned to each process.
Quicksort on a Hypercube

Figure 9.21  The execution of the hypercube formulation of quicksort for $d = 3$. The three splits – one along each communication link – are shown in (a), (b), and (c). The second column represents the partitioning of the $n$-element sequence into subcubes. The arrows between subcubes indicate the movement of larger elements. Each box is marked by the binary representation of the process labels in that subcube. $A = \ldots$ denotes that all the binary combinations are included.

Dr. Joe Zhang  PDC-14: Parallel Sorting

Summary

• General issues
  – Location
  – Comparison or not

• Sorting network
  – Bitonic sort

• Bubble sort and its variants
  – Odd-even transposition

• Quicksort
  – Average performance $n\log(n)$
  – Dependent on pivot selection

• Other Sorting Algorithms*
CSC630/CSC730
Parallel & Distributed Computing

Questions?