PARALLEL SORTING

- where the i/p & o/p sequences are stored
- how comparisons are performed

- 1 element/processor:
  compare-exchange step: \((t_S + t_w)\). As \(t_S \gg t_w\), poor performance

- multiple elements/processor:
  compare split step: \((t_S + \frac{t_w n}{p})\)
  For large blocks \(\Theta(n/p)\) to merge 2 sorted blocks

---

**Figure 6.1** A parallel compare-exchange operation. Processors \(P_i\) and \(P_j\) send their elements to each other. Processor \(P_i\) keeps \(\min\{a_i, a_j\}\), and \(P_j\) keeps \(\max\{a_i, a_j\}\).

Copyright (c) 1994 Benjamin/Cummings Publishing Co.

---

**Figure 6.2** A compare-split operation. Each processor sends its block of size \(n/p\) to the other processor. Each processor merges the received block with its own block and retains only the appropriate half of the merged block. In this example, processor \(P_i\) retains the smaller elements and processor \(P_j\) retains the larger elements.

Copyright (c) 1994 Benjamin/Cummings Publishing Co.
Figure 6.3  A schematic representation of comparators: (a) an increasing comparator, and (b) a decreasing comparator.

Copyright (c) 1994 Benjamin/Cummings Publishing Co.

Figure 6.4  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.

Copyright (c) 1994 Benjamin/Cummings Publishing Co.
BITONIC SORT: \( \Theta \left( \log^2 n \right) \)

- bitonic sequence: \( \uparrow \downarrow \) or \( \downarrow \uparrow \) or a cyclic shift thereof

Let \( s = \langle a_0, a_1, \ldots, a_{n/2-1}, a_{n/2}, \ldots, a_{n-1} \rangle \)

\[ s_1 = \langle \min \{a_0, a_{n/2}\}, \min \{a_1, a_{n/2+1}\}, \ldots, \min \{a_{n/2-1}, a_{n-1}\} \rangle \]

\[ s_2 = \langle \max \{a_0, a_{n/2}\}, \max \{a_1, a_{n/2+1}\}, \ldots, \max \{a_{n/2-1}, a_{n-1}\} \rangle \]

Claim: \( s_1 \) & \( s_2 \) are bitonic & each element of \( s_1 \) is \( < \) each element of \( s_2 \)

- BITONIC SPLIT: split bitonic seq of size \( n \) into 2 bitonic subseqs
- BITONIC MERGE: sort bitonic seq using bitonic splits

Use bitonic merging network

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 6.5 Merging a 16-element bitonic sequence through a series of \( \log 16 \) bitonic splits.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

- To sort \( n \) numbers, repeatedly merge bitonic sequences of \( \uparrow \) length
  (seq of 2 elements is bitonic)

\[ \Rightarrow \text{Each stage of network merges adjacent bitonic seqs in } \uparrow \text{ or } \downarrow \text{ order} \]

\[ \Rightarrow \text{seq obtained by concatenating these seqs is bitonic} \]

- last stage has depth \( \log n \) \([\text{for bitonic merge with } n \text{ inputs}]\)

\[ \text{Depth of network } d(n) = d \left( \frac{n}{2} \right) + \log n = \sum_{i=1}^{\log n} \left( \log^2 \frac{n}{2^i} + \log n \right) / 2 \]
Figure 6.6 A bitonic merging network for $n = 16$. The input wires are numbered $0, 1, \ldots, n - 1$, and the binary representation of these numbers is shown. Each column of comparators is drawn separately; the entire figure represents a $\oplus$BM[16] bitonic merging network. The network takes a bitonic sequence and outputs it in sorted order.

Copyright (c) 1994 Benjamin/Cummings Publishing Co.
Figure 6.7 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, $n = 16$.

Copyright © 1994 Benjamin/Cummings Publishing Co.

First 3 stages in Fig 6.8
Last stage in Fig 6.6

$$d(n) = d\left(\frac{n}{2}\right) + \log n$$

$$= \sum_{i=1}^{\log_2 n} i = \frac{\log_2 n \cdot (\log_2 n + 1)}{2} = \Theta\left(\log^2 n\right)$$
<table>
<thead>
<tr>
<th>Wires</th>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>0100</th>
<th>0101</th>
<th>0110</th>
<th>0111</th>
<th>1000</th>
<th>1001</th>
<th>1010</th>
<th>1011</th>
<th>1100</th>
<th>1101</th>
<th>1110</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>20</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>9</td>
<td>5</td>
<td>20</td>
<td>14</td>
<td>8</td>
<td>14</td>
<td>14</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>20</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>14</td>
<td>12</td>
<td>8</td>
<td>20</td>
<td>12</td>
<td>12</td>
<td>14</td>
<td>3</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>10</td>
<td>14</td>
<td>14</td>
<td>8</td>
<td>3</td>
<td>90</td>
<td>10</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>90</td>
<td>90</td>
<td>40</td>
<td>40</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>35</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>23</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6.8  The comparator network that transforms an input sequence of 16 unordered numbers into a bitonic sequence. In contrast to Figure 6.6, the columns of comparators in each bitonic merging network are drawn in a single box, separated by a dashed line.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.
HC: one element/processor

→ in any step, compare-exchange performed bet. 2 wires if labels differ in 1 bit

→ last step of each stage: labels differing in LSB
→ last 3 stages, 2nd last step: labels differing in 2nd LSB.

→ labels differing in \(i\text{th}\) LSB: perform compare-exchange \((\log n - i + 1)\) times

On HC: \(i\text{th}\) step of final stage: communicate along \((d - (i - 1))\text{th}\) dimension

\((1 + \log n)(\log n)/2\) steps \(\Rightarrow\) cost-optimal worst. sequential implementation of bitonic sort

\[ T_s = o(n \times \log^2 n) \]

Figure 6.9 Communication during the last stage of bitonic sort. Each wire is mapped to a hypercube processor; each connection represents a compare-exchange between processors.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

1. procedure BITONIC-SORT (label, d)
2. begin
3.   for \(i := 0\) to \(d-1\) do
4.     for \(j := i\) downto 0 do
5.       if \((i + 1)\text{st}\) bit of label \(\neq\) \(j\text{th}\) bit of label then
6.         comp-exchange-max(j);
7.       else
8.         comp-exchange-min(j);
9. end BITONIC-SORT
Figure 6.10  Communication characteristics of bitonic sort on a hypercube. During each stage of the algorithm, processors communicate along the dimensions shown.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.
- Mesh: impossible to map such that each compare-exchange between neighbors
- Row-major shuffled mapping: processors that do c-e are on square subsections whose size is inversely related to frequency of c-e
- Wires that differ in i-th LSB mapped to mesh processors
  \[ 2 \log(i-1)/2 \] links away
- Communication overhead = \[ \sum_{i=1}^{\log n} \sum_{j=1}^{(i-1)/2} 2 \approx 7\sqrt{n} = \Theta(\sqrt{n}) \]

* In each of \( \Theta(\log^2 n) \) steps, each processor performs max. 1 comparison
  \( \Rightarrow \) Parallel run time \( T_p = \Theta(\log^2 n) + \Theta(\sqrt{n}) \)
  \( \Rightarrow \) Processor-time product = \( \Theta(n\sqrt{n}) \) \( \Rightarrow \) not cost-optimal!
- Run time of sorting on mesh bounded by \( n \sqrt{n} \). Why?
  \( \Rightarrow \) parallel formulation is asymptotically optimal for mesh architecture.
$n$ numbers, $p \leq n$ processors

**Block of elements per processor**

Hypercube: $T_p = \Theta \left( \frac{n}{p} \log \frac{n}{p} \right) + \Theta \left( \frac{n}{p} \log^2 p \right) + \Theta \left( \frac{n}{p} \log^2 p \right)$

Local sort

Comparisons

Communication

Mesh: $T_p = \Theta \left( \frac{n}{p} \log \frac{n}{p} \right) + \Theta \left( \frac{n}{p} \log^2 p \right) + \Theta \left( \frac{n}{p} \log^2 p \right)$

// log $p$ communication steps of size $\frac{n}{p}$, $\Rightarrow \frac{n}{p} \log p$ comparisons

// MESH: the communication steps need $\Theta \left( \sqrt{p} \right)$ substeps

**Stage 4**

---

**Figure 6.12** The last stage of the bitonic sort algorithm for $n = 16$ on a mesh, using the row-major shuffled mapping. During each step, processor pairs compare-exchange their elements. Arrows indicate the pairs of processors that perform compare-exchange operations.

Copyright (c) 1994 Benjamin/Cummings Publishing Co.

---

**Hypercube (HC):**

$S = \frac{\Theta(n \log n)}{\theta \left( \frac{n}{p} \log \left( \frac{n}{p} \right) \right) + \Theta \left( \frac{n}{p} \log^2 p \right)}$

$E = \frac{1}{1 - \theta \left( \log p \right) / \log n + \Theta \left( \log^2 p \right) / \log n}$

**Mesh:**

$S = \frac{\Theta(n \log n)}{\theta \left( \frac{n}{p} \log \frac{n}{p} \right) + \Theta \left( \frac{n}{p} \log^2 p \right) + \Theta \left( \frac{n}{\sqrt{p}} \right)}$

$E = \frac{1}{1 - \theta \left( \log p \right) / \log n + \Theta \left( \log^2 p \right) / \log n + \Theta \left( \frac{n}{\sqrt{p}} \right) / \log n}$
Bubble-sort $\Theta(n^2)$ inherently sequential; use odd-even transposition sort which has $n$ phases, each w/ $\Theta(n)$ comparisons
Use RING, processor-time product is $\Theta(n^2)$ → NOT cost-optimal.
For cost optimality, each proc has $n/p$ elements.

$$T_p = \Theta\left(\frac{n}{p} \log \frac{n}{p}\right) + \Theta(n) + \Theta(n)$$

// local sort, comparisons, communication // $p$ phases of $\frac{n}{p}$ data at each

<table>
<thead>
<tr>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 2 3 8 5 6 4 1</td>
<td>1 2 3 3 4 5 6 8</td>
</tr>
<tr>
<td>1 2 3 3 8 5 6 1 4</td>
<td>Phase 1 (odd)</td>
</tr>
<tr>
<td>2 3 3 5 8 1 6 4</td>
<td>Phase 2 (even)</td>
</tr>
<tr>
<td>2 3 3 5 1 8 4 6</td>
<td>Phase 3 (odd)</td>
</tr>
<tr>
<td>2 3 3 1 5 4 8 6</td>
<td>Phase 4 (even)</td>
</tr>
<tr>
<td>2 3 1 3 4 5 6 8</td>
<td>Phase 5 (odd)</td>
</tr>
<tr>
<td>2 1 3 3 4 5 6 8</td>
<td>Phase 6 (even)</td>
</tr>
<tr>
<td>1 2 3 3 4 5 6 8</td>
<td>Phase 7 (odd)</td>
</tr>
<tr>
<td>1 2 3 3 4 5 6 8</td>
<td>Phase 8 (even)</td>
</tr>
</tbody>
</table>

Figure 6.13: Sorting $n = 8$ elements, using the odd-even transposition sort algorithm. During each phase, $n = 8$ elements are compared.

Copyright (c) 1994 Benjamin/Cummings Publishing Co.

$$S = \frac{\Theta(n \log n)}{\Theta\left(\frac{n}{p} \log \frac{n}{p}\right) + \Theta(n)}$$

$$E = \frac{1}{1 - \Theta\left(\frac{(\log p)}{\log n}\right)} + \Theta\left(p/\log n\right)$$

- cost optimal when $p = O\left(\log n\right)$
- Isoefficiency function is $\Theta\left(p^{2/p}\right)$
Figure 3.8  The quicksort task-dependency graph based on recursive decomposition for sorting a sequence of 12 numbers.
QuickSort

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. swap \( A[q], A[s] \);
14. QUICKSORT \( A, q, s \);
15. QUICKSORT \( A, s + 1, r \);
16. end if
17. end QUICKSORT

**Algorithm 9.5** The sequential quicksort algorithm.

**Figure 9.15** Example of the quicksort algorithm sorting a sequence of size \( n = 8 \).

- denotes \( s \)’s position before swap.
QUICKSORT: Worst case $T(n) = T(n-1) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)$

$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$ \( \Rightarrow \) depends on pivot selection

\[
\text{QUICKSORT} \quad (A,q,r) \\
\text{if} \quad q < r \text{ then} \\
\quad x := A[q] \\
\quad s := q \\
\quad \text{for} \quad i := q + 1 \text{ to } r \text{ do} \\
\quad \quad \text{if} \quad A[i] \leq x \text{ then} \\
\quad \quad \quad s := s + 1 \\
\quad \quad \quad \text{swap}(A[s], A[i]) \\
\quad \text{swap}(A[q], A[s]) \\
\quad \text{QUICKSORT} \quad (A,q,s) \\
\quad \text{QUICKSORT} \quad (A,s+1,r) \\
\]

Naive parallel formulation: give a subproblem to another processor

\text{Drawback: partitioning done by single processor} \Rightarrow \Omega(n) \\
\text{important to do partitioning in parallel} \\
T_P = \Omega(n^2) \quad \rho T_P = \Omega(n^2) \neq O(n \log n) \\
\text{not cost-optimal.}

\[\text{Figure 6.15} \quad \text{Example of the quicksort algorithm sorting a sequence of size } n = 8.\]

Copyright (c) 1994 Benjamin/Cummings Publishing Co.
Parallel Formulation for CRCW PRAM

→ construct a binary tree; sorted sequence is its in-order traversal

Figure 6.16 A binary tree generated by the execution of the quicksort algorithm. Each level of the tree represents a different array-partitioning iteration. If pivot selection is optimal, then the height of the tree is $\Theta(\log n)$, which is also the number of iterations.

Copyright (c) 1994 Benjamin/Cummings Publishing Co.
A[\text{root}] is used as pivot pivot = copied into local parent = Processors with elements < A[parent_i] write their labels into Lc[parent_i] = etc.

Algorithm continues until n pivot elements are selected
Processor exits when its element becomes pivot
In each iteration, 1 level of tree is constructed in $\Theta(1)$ time
Avg complexity of binary tree building = $\Theta(\log n)$
To traverse tree $\Theta(\log n)$ on n-processor PRAM
Figure 6.17  The execution of the PRAM algorithm on the array shown in (a). The arrays leftchild and rightchild are shown in (c), (d), and (e) as the algorithm progresses. Figure (f) shows the binary tree constructed by the algorithm. Each node is labeled by the processor (in square brackets), and the element is stored at that processor (in curly brackets). The element is the pivot. In each node, processors with smaller elements than the pivot are grouped on the left side of the node, and those with larger elements are grouped on the right side. These two groups form the two partitions of the original array. For each partition, a pivot element is selected at random from the two groups that form the children of the node.

Copyright (c) 1994 Benjamin/Cummings Publishing Co.
QuickSort on HC:

1. Select a pivot
2. BC the pivot
3. Exchange lower/higher elements along dimension i

\[ T_p = \frac{\Theta(n \log \frac{n}{p})}{\text{local sort}} + \frac{\Theta(n \log p)}{\text{communication}} + \frac{\Theta(\log^2 p)}{\text{pivot broadcasting}} \]

Hypercube

Sequence of Elements

(a) Split along the third dimension. Partitions the sequence into two big blocks—one smaller and one larger than the pivot.

(b) Split along the second dimension. Partitions each subblock into two smaller subblocks.

(c) Split along the first dimension. The elements are sorted according to the global ordering imposed by the processors’ labels onto the hypercube.

Figure 6.18 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 processor labels in that subcube. A * denotes that all the binary combinations are included.

Copyright © 1994 Benjamin/Cummings Publishing Co.

\[
S = \frac{\Theta(n \log n)}{\Theta((\frac{n}{p}) \log (\frac{n}{p})) + \Theta((\frac{n}{p}) \log p) + \Theta(\log^2 p)}
\]

For cost optimality,

\[
\left( \frac{p \log^2 p}{(n \log n)} \right) = O(1)
\]

\[
p = \Theta(n / \log n)
\]

\[
\text{Efficiency} = \Theta(p \log^2 p)
\]