PARALLEL SORTING

- where the I/P & O/P sequences are stored
- how comparisons are performed

- 1 element/processor:
  compare-Xchange step: $(t_s + t_w)$. As $t_s \gg t_w$, poor performance
- multiple elements/processor:
  compare split step: $(t_s + t_w \frac{n}{p})$
  For large blocks $\Theta(n/p)$ to merge 2 sorted blocks

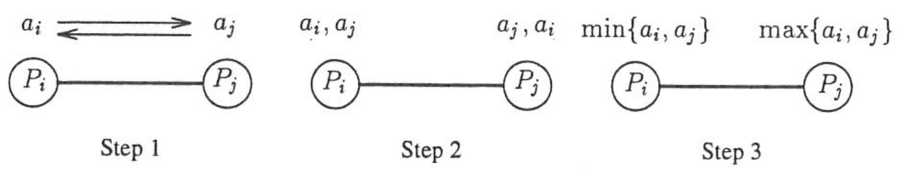


Figure 6.1 A parallel compare-exchange operation. Processors P_i and P_j send their elements to each other. Processor P_i keeps $\min\{a_i, a_j\}$, and P_j keeps $\max\{a_i, a_j\}$.

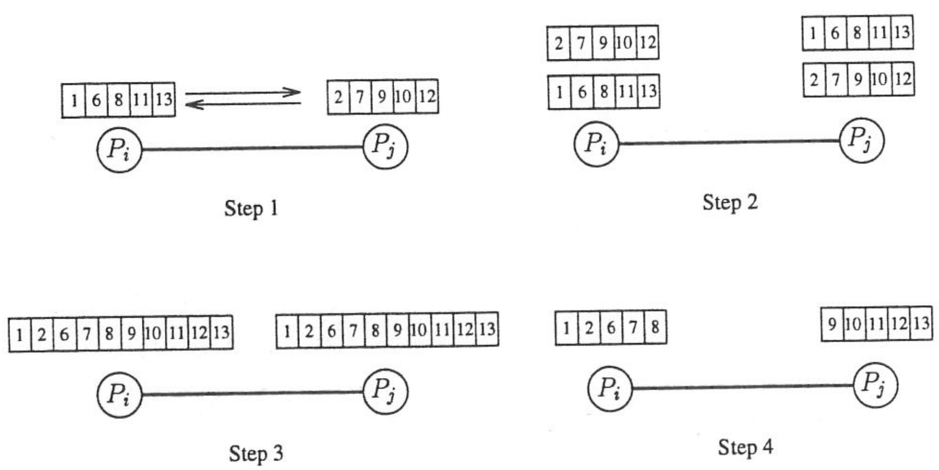


Figure 6.2 A compare-split operation. Each processor sends its block of size n/p to the other processor. Each processor merges the received block with its own block and retains only the appropriate half of the merged block. In this example, processor P_i retains the smaller elements and processor P_j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

SORTING NETWORKS

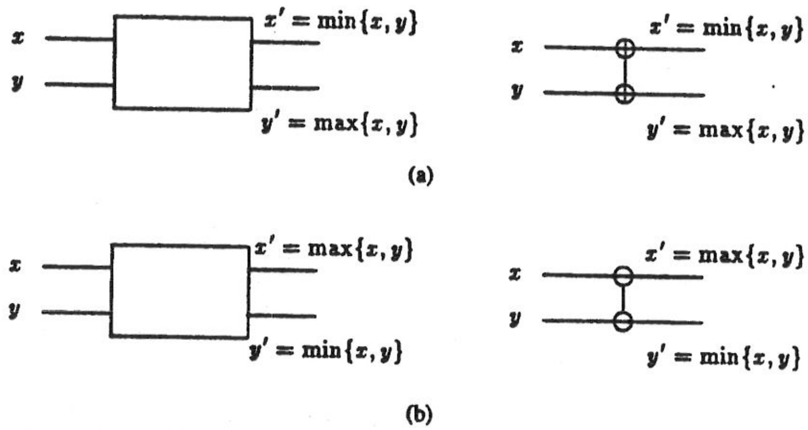


Figure 6.3 A schematic representation of comparators: (a) an increasing comparator, and (b) a decreasing comparator.

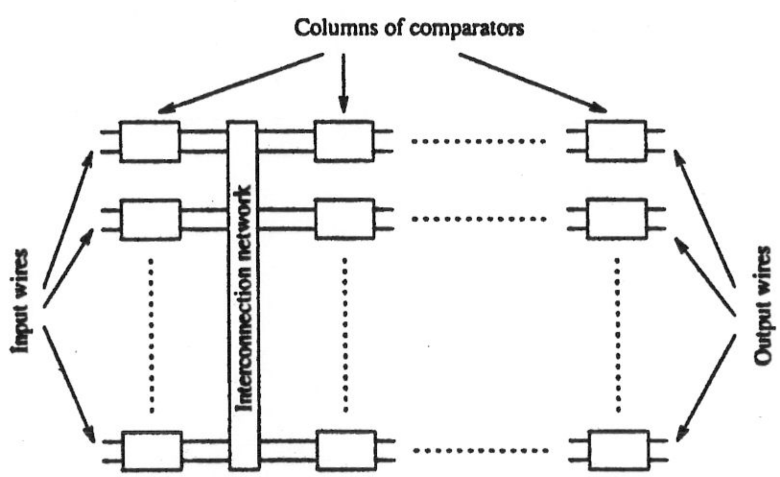


Figure 6.4 A typical sorting network. Every sorting network is made up of a series of columns, and each column contains a number of comparators connected in parallel.

BITONIC SORT : $\Theta(\log^2 n)$

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

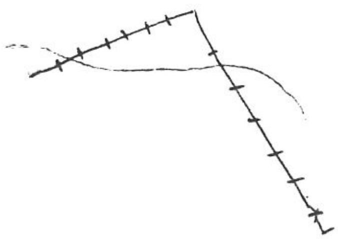
- let $s = \langle \underbrace{a_0\ a_1 \cdots a_{n/2-1}}_{\uparrow}\ \underbrace{a_{n/2} \cdots a_{n-1}}_{\downarrow} \rangle$

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

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. [RECURSION log n steps]

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

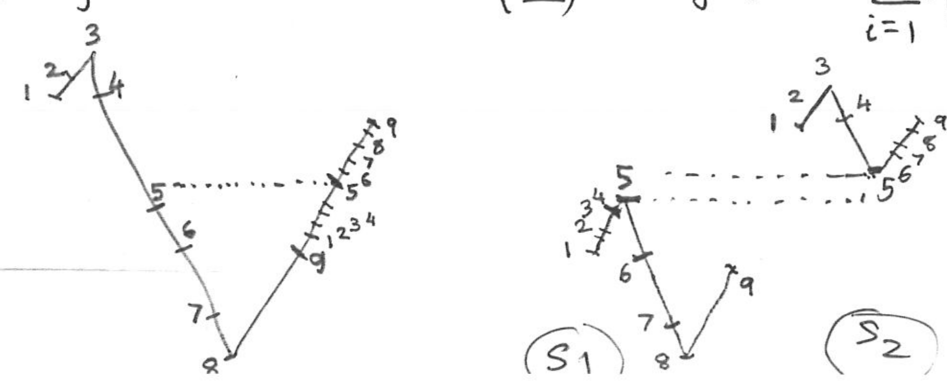
Figure 6.5 Merging a 16-element bitonic sequence through a series of $\log 16$ bitonic splits.

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

$\rightarrow$ Each stage of network merges adjacent bitonic seqs in $\uparrow$ or $\downarrow$ order

$\Rightarrow$ seq. obtained by concatenating these seqs is bitonic

- last stage has depth $\log n$ [for bitonic merge with n inputs]
- Depth of network $d(n) = d\left(\frac{n}{2}\right) + \log n = \sum_{i=1}^{\log n} i = (\log^2 n + \log n)/2$



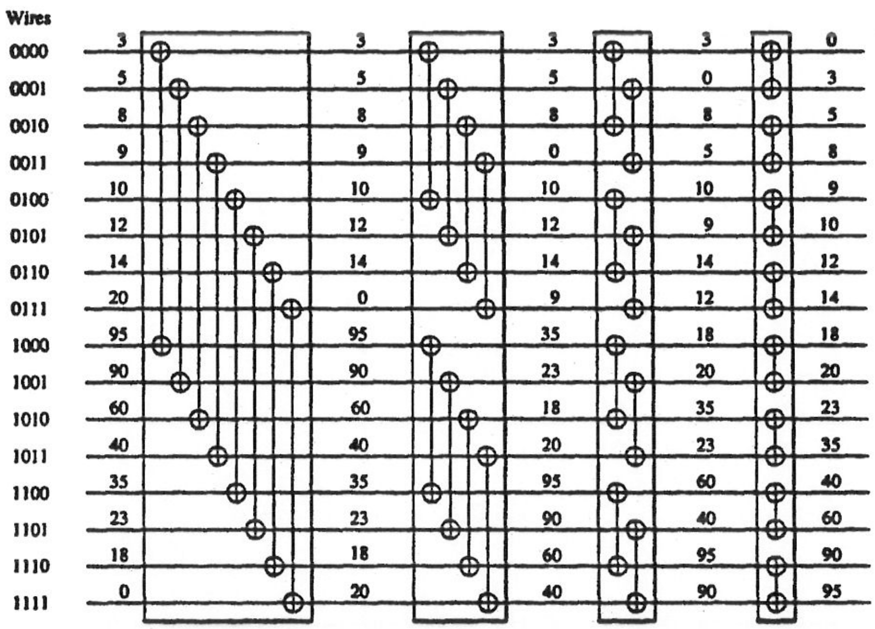
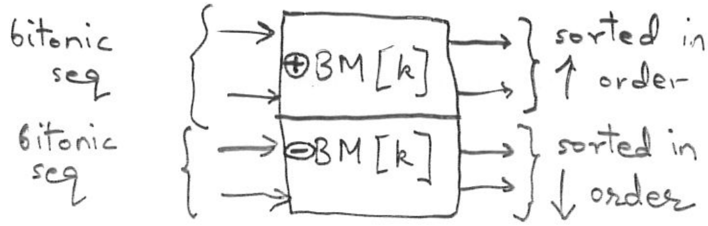


Figure 6.6 A bitonic merging network for $n = 16$. The input wires are numbered $0, 1 \ldots, n-1$, and the binary representation of these numbers is shown. Each column of comparators is drawn separately; the entire figure represents a $\oplus$BM[16] bitonic merging network. The network takes a bitonic sequence and outputs it in sorted order.



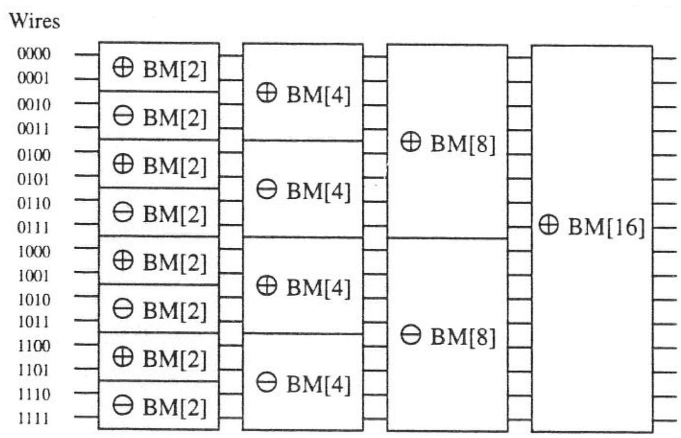


Figure 6.7 A schematic representation of a network that converts an input sequence into a bitonic sequence. In this example, ⊕BM[k] and ⊖BM[k] denote bitonic merging networks of input size k that use ⊕ and ⊖ comparators, respectively. The last merging network (⊕BM[16]) sorts the input. In this example, $n = 16$.

First 3 stages in Fig 6.8

Last stage in Fig 6.6

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

$$= \sum_{1}^{\log n} i = \frac{\log^2 n + \log n}{2} = \theta\left(\log^2 n\right)$$

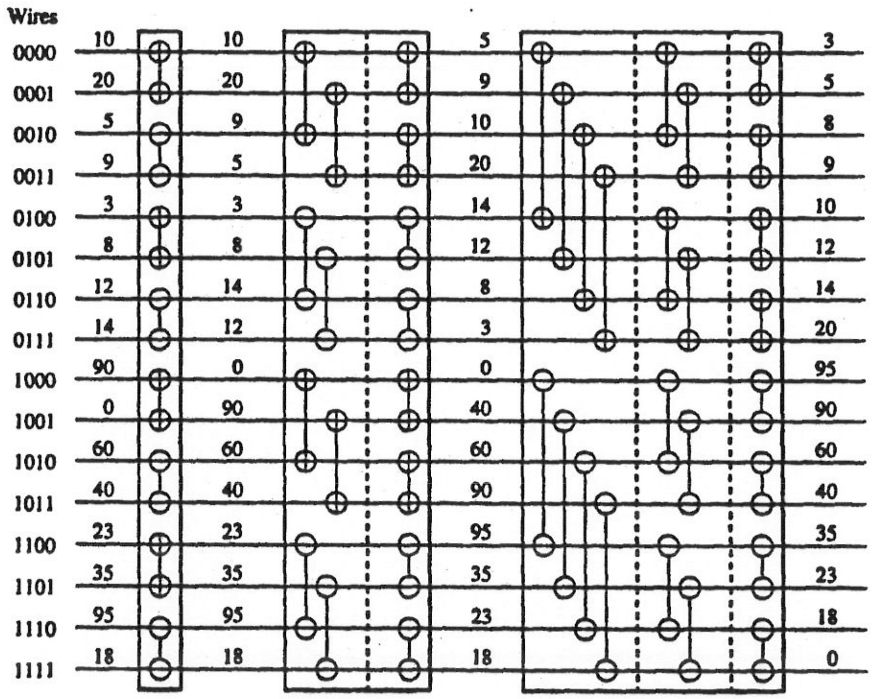


Figure 6.8 The comparator network that transforms an input sequence of 16 unordered numbers into a bitonic sequence. In contrast to Figure 6.6, the columns of comparators in each bitonic merging network are drawn in a single box, separated by a dashed line.

HC : one element/processor

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

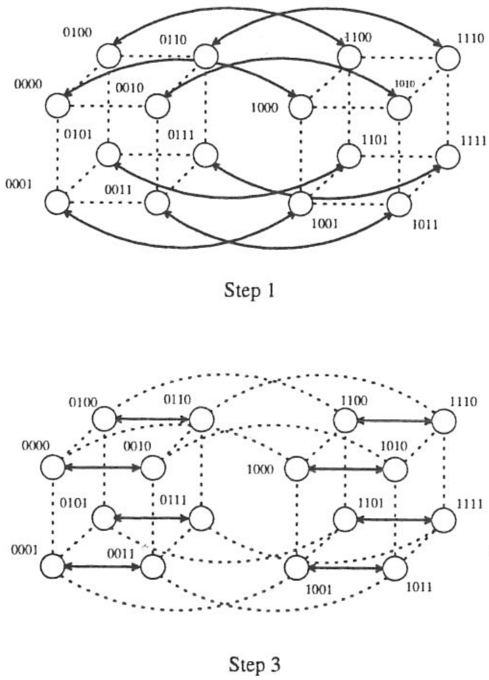
→ Last step of each stage : labels differing in LSB

→ Last 3 stages, 2^{nd}-last step : labels differing in 2^{nd} LSB.

⇒ Labels differing in i^{th}-LSB : perform compare-Xchange $(\log n - i + 1)$ times

• On HC : i^{th} step of final stage : communicate along $(d-(i-1))^{st}$ dimension

- $(1+\log n)(\log n)/2$ steps ⇒ cost-optimal w.r.t. 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.


```
procedure BITONIC_SORT(label, d)
begin
    for i := 0 to d-1 do
        for j := i downto 0 do
            if (i+1)st bit of label ≠ jth bit of label then
                comp_exchange_max(j);
            else
                comp_exchange_min(j);
end BITONIC_SORT
```

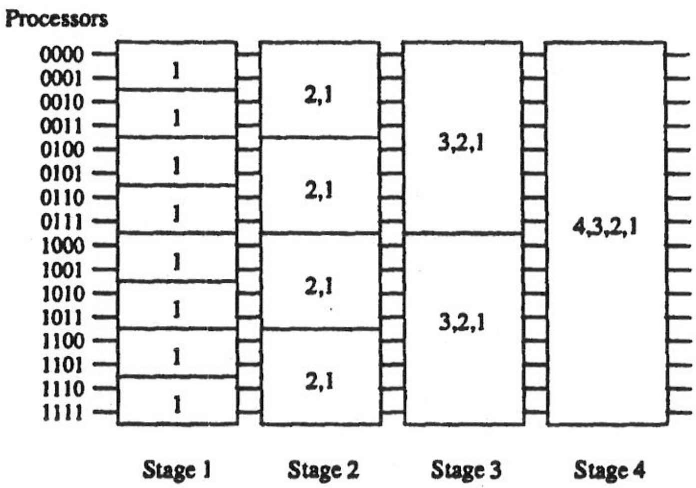
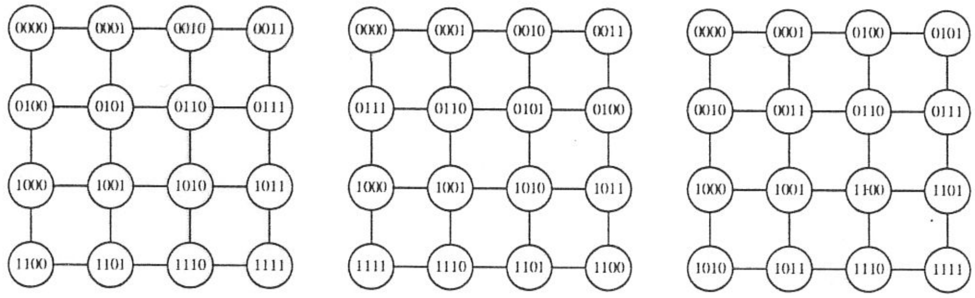


Figure 6.10 Communication characteristics of bitonic sort on a hypercube. During each stage of the algorithm, processors communicate along the dimensions shown.

- Mesh: impossible to map such that each compare-Xchange between neighbors
- Row-major shuffled mapping: processors that do c-e are on square subsections whose size is inversely related to frequency of c-es
- [Wires that differ in i^{th} LSB mapped to mesh processors $2^{\lfloor (i-1)/2 \rfloor}$ links away
- Communication overhead performed per processor $= \sum_{i=1}^{\log n} \sum_{j=1}^{i} 2^{\lfloor (j-1)/2 \rfloor} \approx 7\sqrt{n} = \Theta(\sqrt{n})$



(a) (b) (c)

Figure 6.11 Different ways of mapping the input wires of the bitonic sorting network to a mesh of processors: (a) row-major mapping, (b) row-major snakelike mapping, and (c) row-major shuffled mapping.

- In each of $\Theta(\log^2 n)$ steps, each processor performs max. 1 comparison

$\Rightarrow$ Parallel run time $T_P = \overbrace{\Theta(\log^2 n)}^{\text{compare}} + \overbrace{\Theta(\sqrt{n})}^{\text{exchange}}$

$\Rightarrow$ Processor-time product $= \Theta(n\sqrt{n})$ $\Rightarrow$ not cost-optimal !

- Run time of sorting on mesh bounded by $\Omega(\sqrt{n})$. Why ?

$\therefore$ parallel formulation is asymptotically optimal for mesh architecture

n numbers; $p\ (<n)$ processors

Block of elements per processor

Naive emulation: on HC, $T_p = \frac{n \log^2 n}{p}$

$E = \frac{1}{\log n}$ ∴ not cost optimal. $\quad pT_p = \Theta(n \log^2 n)$

Hypercube: $$T_p = \underbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}_{\text{local sort}} + \underbrace{\Theta\left(\frac{n}{p}\log^2 p\right)}_{\text{comparisons}} + \underbrace{\Theta\left(\frac{n}{p}\log^2 p\right)}_{\text{communication}}$$

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

// $\log^2 p$ communication steps, of size $\frac{n}{p}$, $\Rightarrow \frac{n}{p}\log^2 p$ COMPARISONS

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

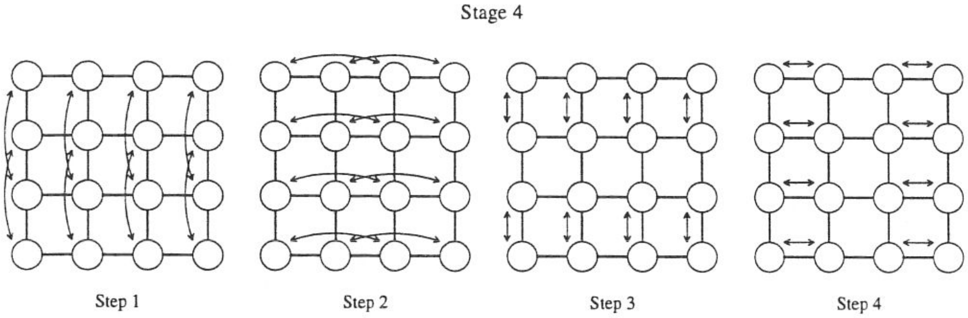


Figure 6.12 The last stage of the bitonic sort algorithm for $n = 16$ on a mesh, using the row-major shuffled mapping. During each step, processor pairs compare-exchange their elements. Arrows indicate the pairs of processors that perform compare-exchange operations.

$W = KT_o \Rightarrow pT_p - T_s$

HC: $$S = \frac{\Theta(n \log n)}{\Theta((n/p \log (n/p))) + \Theta((n/p \log^2 p))}$$

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

For cost optimality $(\log^2 p)/\log n = O(1) \Rightarrow$

$P = \Theta\left(2^{\sqrt{\log n}}\right)$ procs

$n = \Theta(2^{\log^2 p})$ ∴ $W = n \log n \Rightarrow$

Isoefficiency function $= \Theta\left(p^{\log p} \log^2 p\right)$

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

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

For cost optimality, $\sqrt{p}/\log n = O(1) \Rightarrow$

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

Isoefficiency function $= \Theta\left(2^{\sqrt{p}}\sqrt{p}\right)$

• Bubble-sort $\Theta(n^2)$ inherently sequential ∴ use odd-even Xposition sort which has n phases, each w/ $\Theta(n)$ comparisons

Use RING. Processor-time product is $\Theta(n^2) \Rightarrow$ NOT cost-optimal

• For cost optimality, each proc has n/p elements

$$T_p = \underbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}_{\text{local sort}} + \underbrace{\Theta(n)}_{\text{comparisons}} + \underbrace{\Theta(n)}_{\text{communication}}$$

// p phases of $\frac{n}{p}$ data each

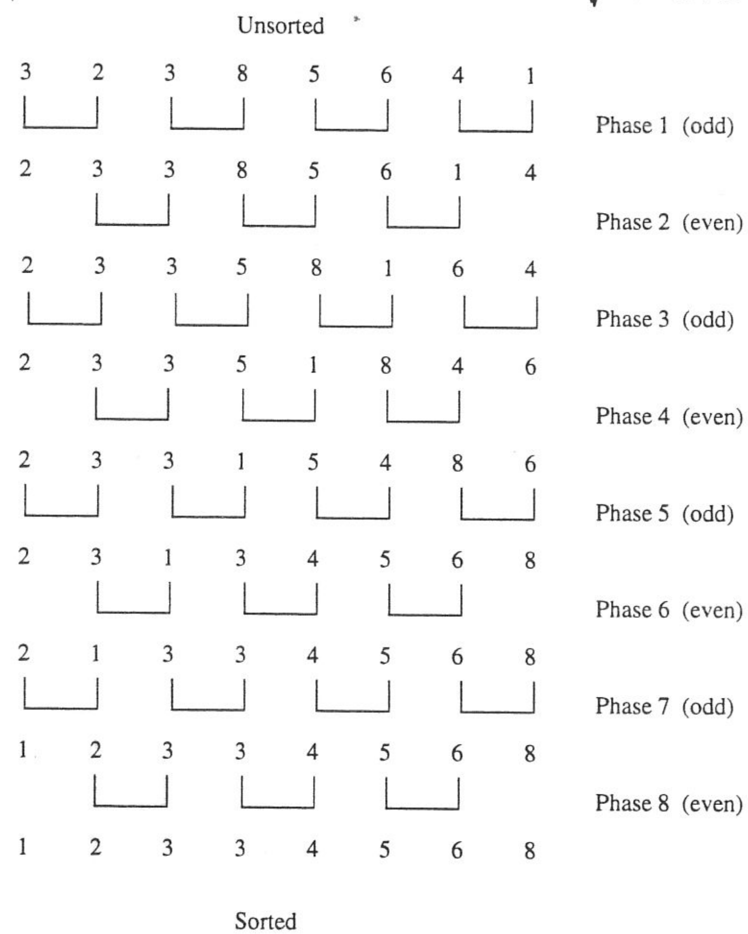


Figure 6.13 Sorting $n = 8$ elements, using the odd-even transposition sort algorithm. During each phase, $n = 8$ elements are compared.

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

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

• cost optimal when $p = O(\log n)$

• Isoefficiency function is $\Theta(p\,2^p)$

// $n = \Theta(2^p)$ ∴ $W = n\log n = \Theta(p\,2^p)$

BUBBLE SORT

for $i = n-1$ downto 1
  for $j = 1$ to i
    compare-Xchg(a_j, a_{j+1})

Analyze: $W = KT_o$
$= K[pT_p - T_s]$
ie, when $T_s \approx \Theta(pT_p)$

$n\log n \approx KT_o \approx K\left[\left(n\log\frac{n}{p} + np\right) - n\log n\right]$

∴ $\log n \propto p - \log p$

Example: Quicksort

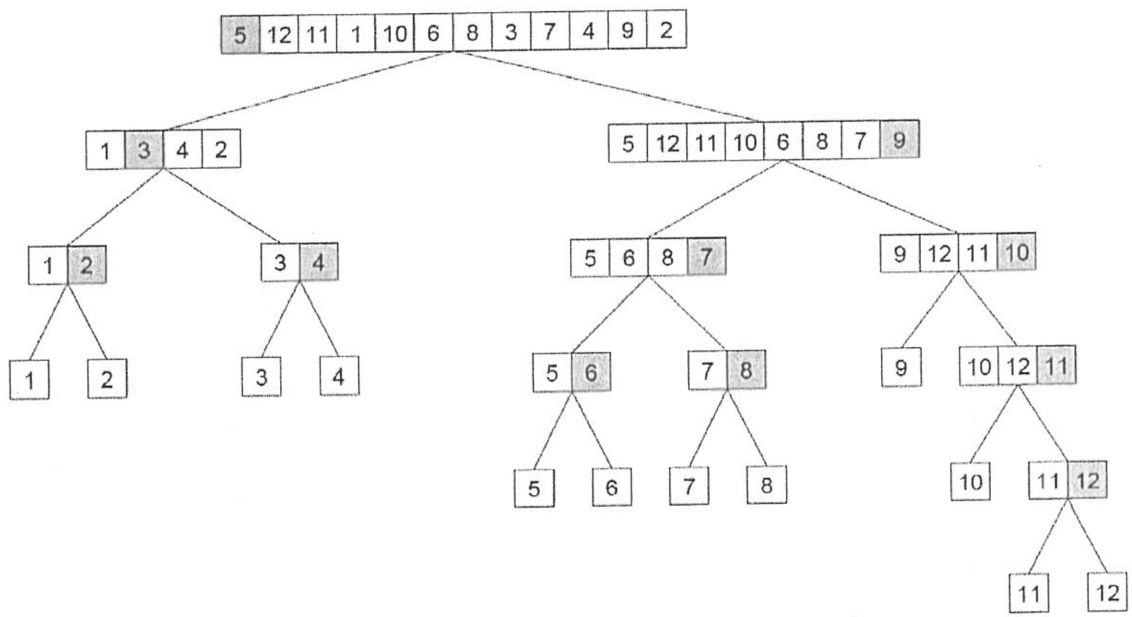


Figure 3.8 The quicksort task-dependency graph based on recursive decomposition for sorting a sequence of 12 numbers.

Quicksort

3 7 8 4 2 1 9 5 $\boxed{5}$
3 4 2 7 8 1 9 5 $\boxed{5}$
3 4 2 1 5 7 9 8 $\boxed{5}$
3 4 2 1 5 5 9 8 7

```
procedure QUICKSORT (A, q, r)
begin
    if q < r then
    begin
        x := A[q];
        s := q;
        for i := q + 1 to r do
            if A[i] ≤ x then
            begin
                s := s + 1;
                swap(A[s], A[i]);
            end if
        swap(A[q], A[s]);
        QUICKSORT (A, q, s);
        QUICKSORT (A, s + 1, r);
    end if
end QUICKSORT
```

Algorithm 9.5 The sequential quicksort algorithm.

(a)	3	2	1	5	8	4	3	7
(b)	1	2	3	5	8	4	3	7
(c)	1	2	3	3	4	5	8	7
(d)	1	2	3	3	4	5	7	8
(e)	1	2	3	3	4	5	7	8

Pivot

Final position

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

s = 9

3 9 8 1 2 7 0 4 2
i i i

3 1 8 9 2 7 0 4 2

3 1 2 9 8 7 0 4 2

0 8 7 9 4 2

$\boxed{3}$ 1 2 0 2 7 9 4 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

```
QUICKSORT (A, q, r)
if q < r then
    x := A[q]
    s := q
    for i := q+1 to r do
        if A[i] ≤ x then
            [s := s+1
            [swap (A[s], A[i])
    swap(A[q], A[s])
    QUICKSORT (A, q, s)
    QUICKSORT (A, s+1, r)
```

Naive parallel formulation: give a subproblem to another processor

Drawback: partitioning done by single processor $\Rightarrow \Omega(n)$

$\therefore$ important to do partitioning in parallel

$T_p = \Omega(n)$ $\therefore$ $pT_p = \Omega(n^2) \neq O(n \log n)$

$\therefore$ not cost-optimal.

$[\underbrace{q \cdots s}_{\leq} \quad \underbrace{s+1 \cdots r}_{>}]$ $A[q]$

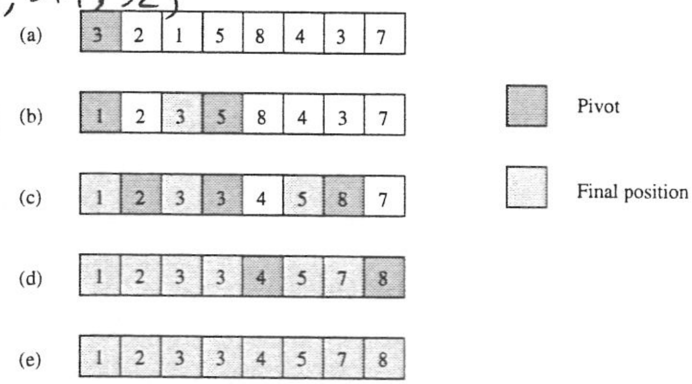


Figure 6.15 Example of the quicksort algorithm sorting a sequence of size $n = 8$.

q, s   i

x =

3 8 9 4 1 7 2 0 6 5

1 8

2 9

0 4

0 1 2 3 8 7 9 4 6 5

Parallel Formulation for CRCW PRAM

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

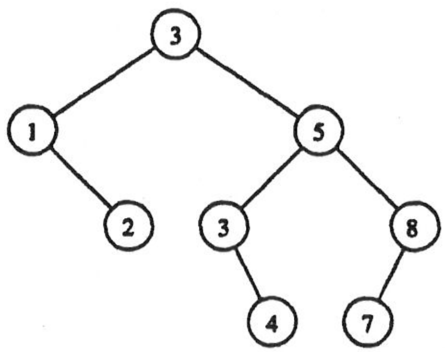
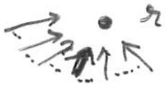


Figure 6.16 A binary tree generated by the execution of the quicksort algorithm. Each level of the tree represents a different array-partitioning iteration. If pivot selection is optimal, then the height of the tree is $\Theta(\log n)$, which is also the number of iterations.



root

root ← i

$parent_i$ ← root

A N

leftchild

rightchild

```
procedure BUILD_TREE (A[1...n])
begin
    for each processor i do
    begin
        root := i;
        parent_i := root;
        leftchild[i] := rightchild[i] := n + 1;
    end for
    repeat for each processor i ≠ root do
    begin
        if (A[i] < A[parent_i]) or
            (A[i] = A[parent_i] and i < parent_i) then
        begin
            leftchild[parent_i] := i;
            if i = leftchild[parent_i] then exit
            else parent_i := leftchild[parent_i];
        end for
        else
        begin
            rightchild[parent_i] := i;
            if i = rightchild[parent_i] then exit
            else parent_i := rightchild[parent_i];
        end else
    end repeat
end BUILD_TREE
```

Lc[$parent_i$]   rc[$parent_i$]

Program 6.6 The binary tree construction procedure for the CRCW PRAM parallel quicksort formulation.

A[root] is used as first pivot
→ copied into local $parent_i$
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

(a) A

	1	2	3	4	5	6	7	8
A	33	21	13	54	82	33	40	72

(b) root = 4

(c)

	1	2	3	4	5	6	7	8
leftchild				1				
rightchild				5				

(d)

	1	2	3	4	5	6	7	8
leftchild	2			1	8			
rightchild	6			5				

(e)

	1	2	3	4	5	6	7	8
leftchild	2	3		1	8			
rightchild	6			5		7		

(f)

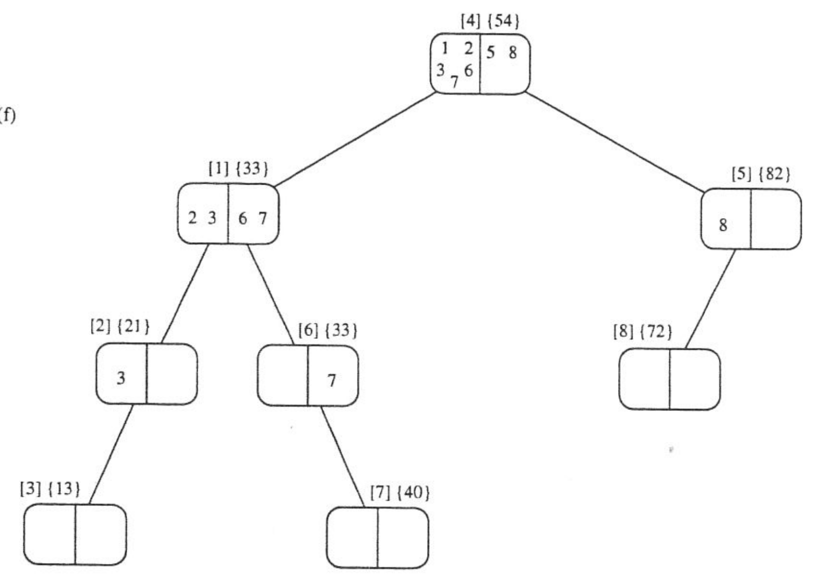
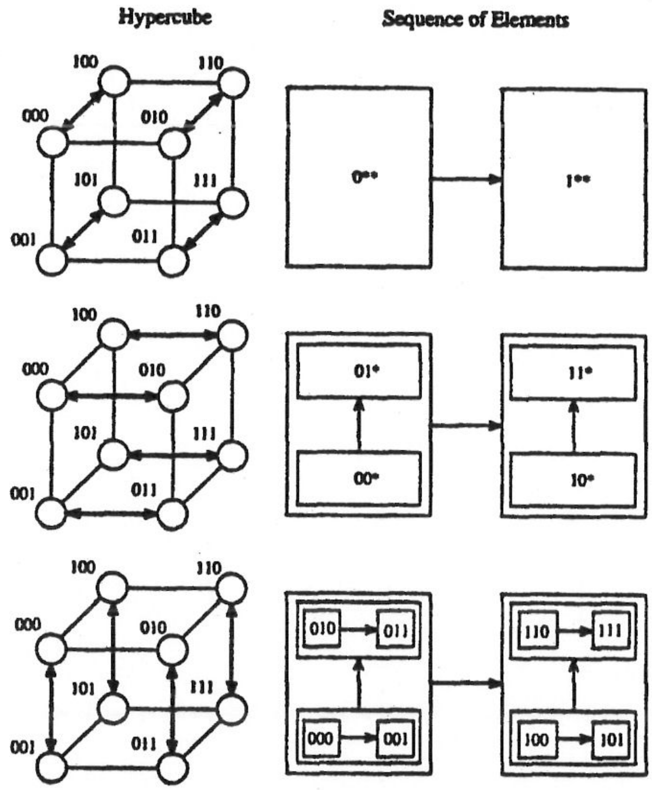


Figure 6.17 The execution of the PRAM algorithm on the array shown in (a). The arrays *leftchild* and *rightchild* are shown in (c), (d), and (e) as the algorithm progresses. Figure (f) shows the binary tree constructed by the algorithm. Each node is labeled by the processor (in square brackets), and 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.

Quicksort on HC:

d steps: $1 \leq i \leq d$
$$\left[\begin{array}{l}\text{select a pivot} \\ \text{BC the pivot} \\ \text{Xchange lower/higher elements along dimension } i\end{array}\right]$$
pivot selection is critical!

$$T_p = \underbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}_{\text{local sort}} + \underbrace{\Theta\left(\frac{n}{p}\log p\right)}_{\text{communication}} + \underbrace{\Theta\left(\log^2 p\right)}_{\text{pivot broadcasting}}$$



(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.

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

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

For cost optimality

$(p \log^2 p)/(n \log n) = O(1)$

$\Rightarrow \quad p = \Theta(n/\log n)$

Isoefficiency $= \Theta(p \log^2 p)$