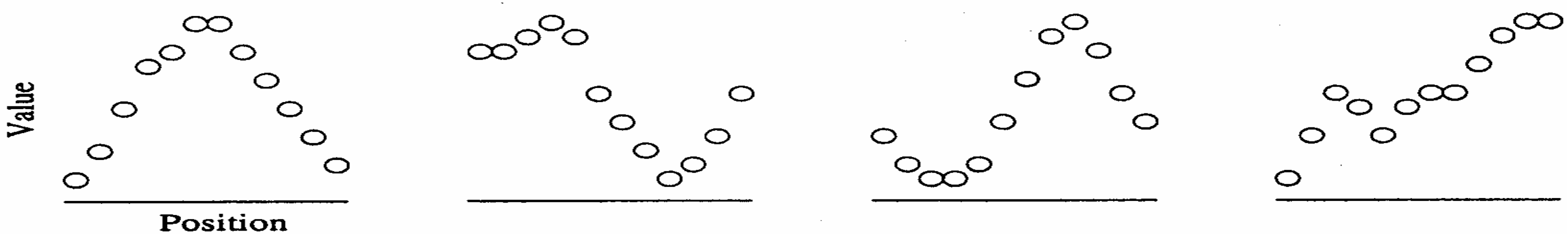
Parallel Sorting Algorithms

– Bitonic Merge

Definition 10.1. A **bitonic sequence** is a sequence of values $a_0, \ldots, a_{n-1}$, with the property that (1) there exists an index i, where $0 \leq i \leq n-1$, such that a_0 through a_i is monotonically increasing and a_i through a_{n-1} is monotonically decreasing, or (2) there exists a cyclic shift of indices so that the first condition is satisfied.

The first three sequences are bitonic sequences; the last sequence is not.



Lemma 10.1. If n is even, then $n/2$ comparators are sufficient to transform a bitonic sequence of n values,

$$a_0, a_1, a_2, \ldots, a_{n-2}, a_{n-1}$$

into two bitonic sequences of $n/2$ values,

$$\min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}), \ldots, \min(a_{n/2-1}, a_{n-1})$$

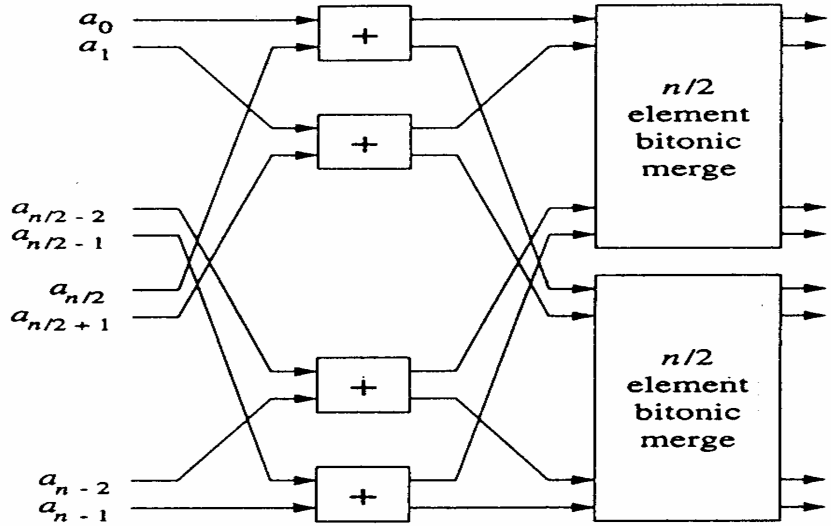
and

$$\max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}), \ldots, \max(a_{n/2-1}, a_{n-1})$$

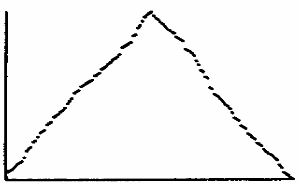
such that no value in the first sequence is greater than any value in the second sequence.

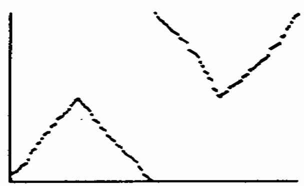
Parallel Sorting Algorithms

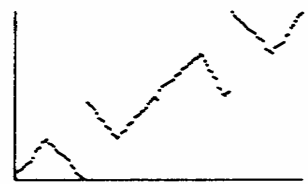
– Bitonic Merge

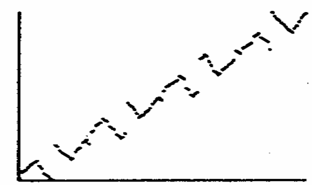


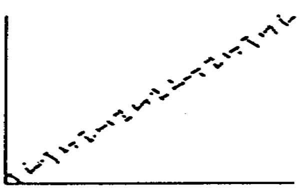
The recursive nature of bitonic merge. Given a bitonic sequence, a single compare-exchange step divides the sequence into two bitonic sequences of half the length. Applying this step recursively yields a sorted sequence, which can be thought of as half of a bitonic sequence of twice the length.

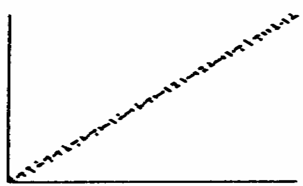

After sixth iteration


After first merge

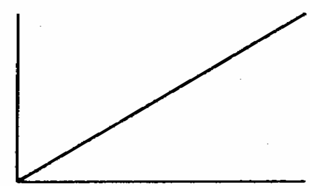

After second merge


After third merge


After fourth merge

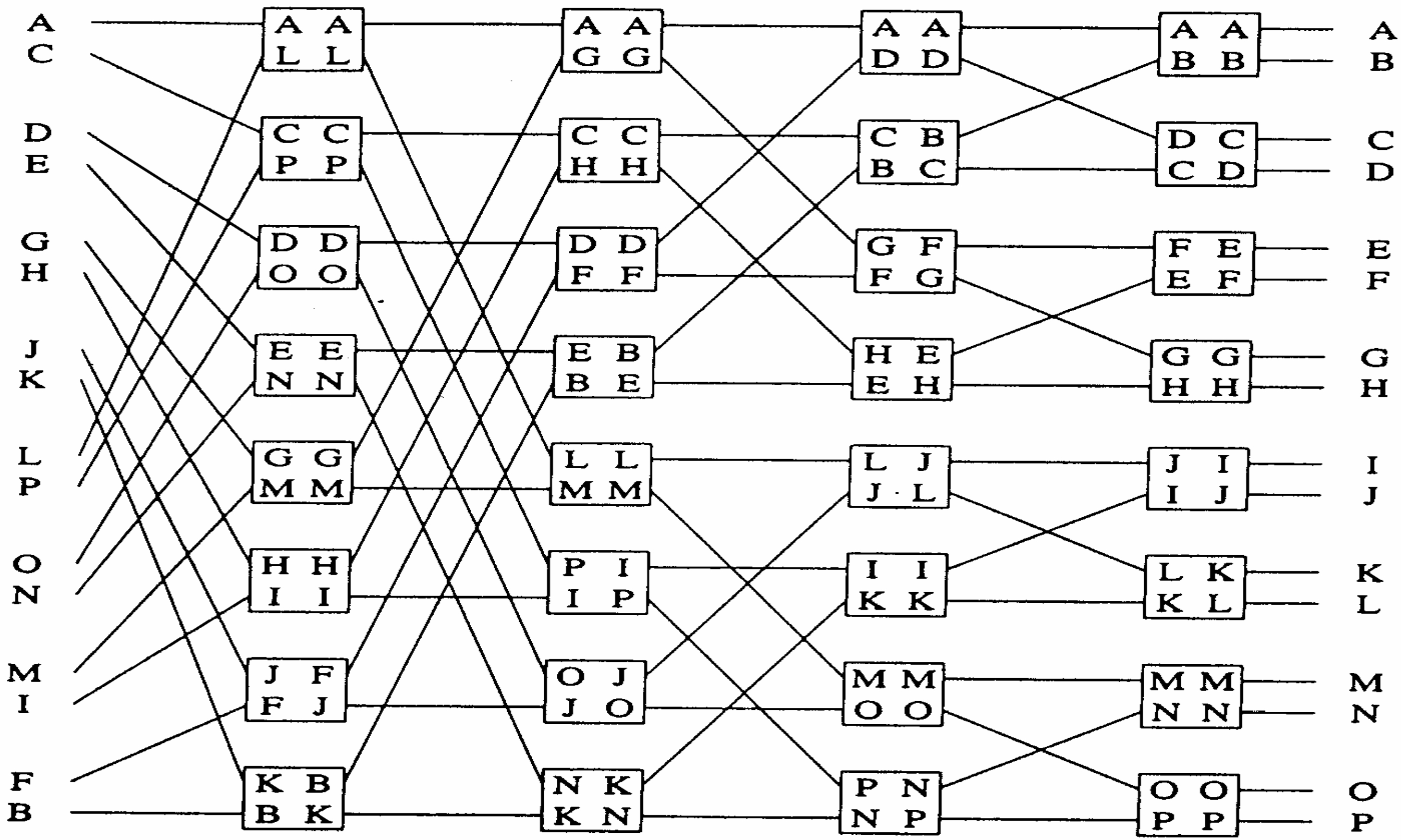

After fifth merge


After sixth merge


After seventh merge

Parallel Sorting Algorithms

– Bitonic Merge

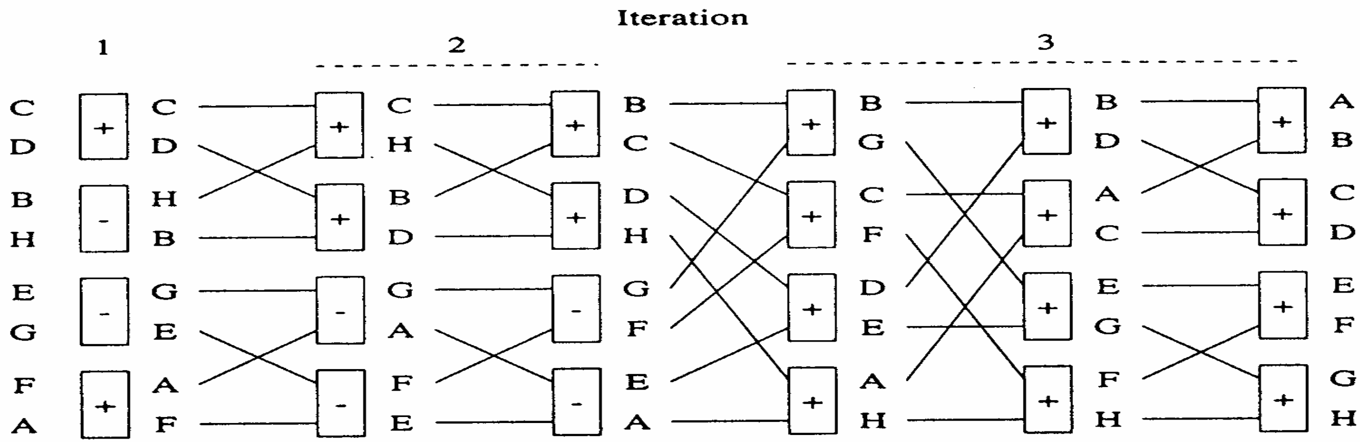


Sorting a bitonic sequence of length 16 by using bitonic merge.

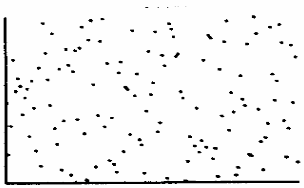
Parallel Sorting Algorithms

– Bitonic Merge–Sort

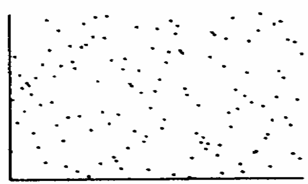
Theorem 10.6. A list of $n = 2^k$ unsorted elements can be sorted in time $\Theta(\log^2 n)$ with a network of $2^{k-1}[k(k-1)+1]$ comparators using the shuffle-exchange interconnection scheme exclusively. (See Stone 1971.)



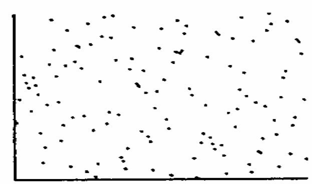
Bitonic merge-sort of an unsorted list of eight elements.



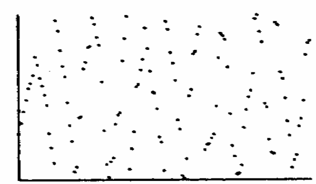
Initial values



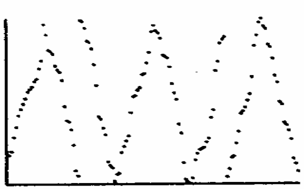
After first iteration



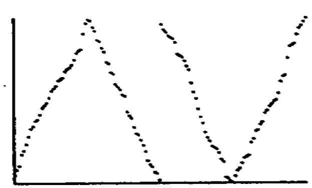
After second iteration



After third iteration

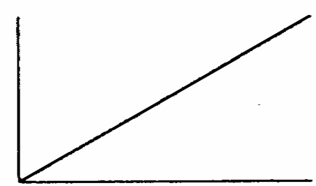


After fourth iteration



After fifth iteration

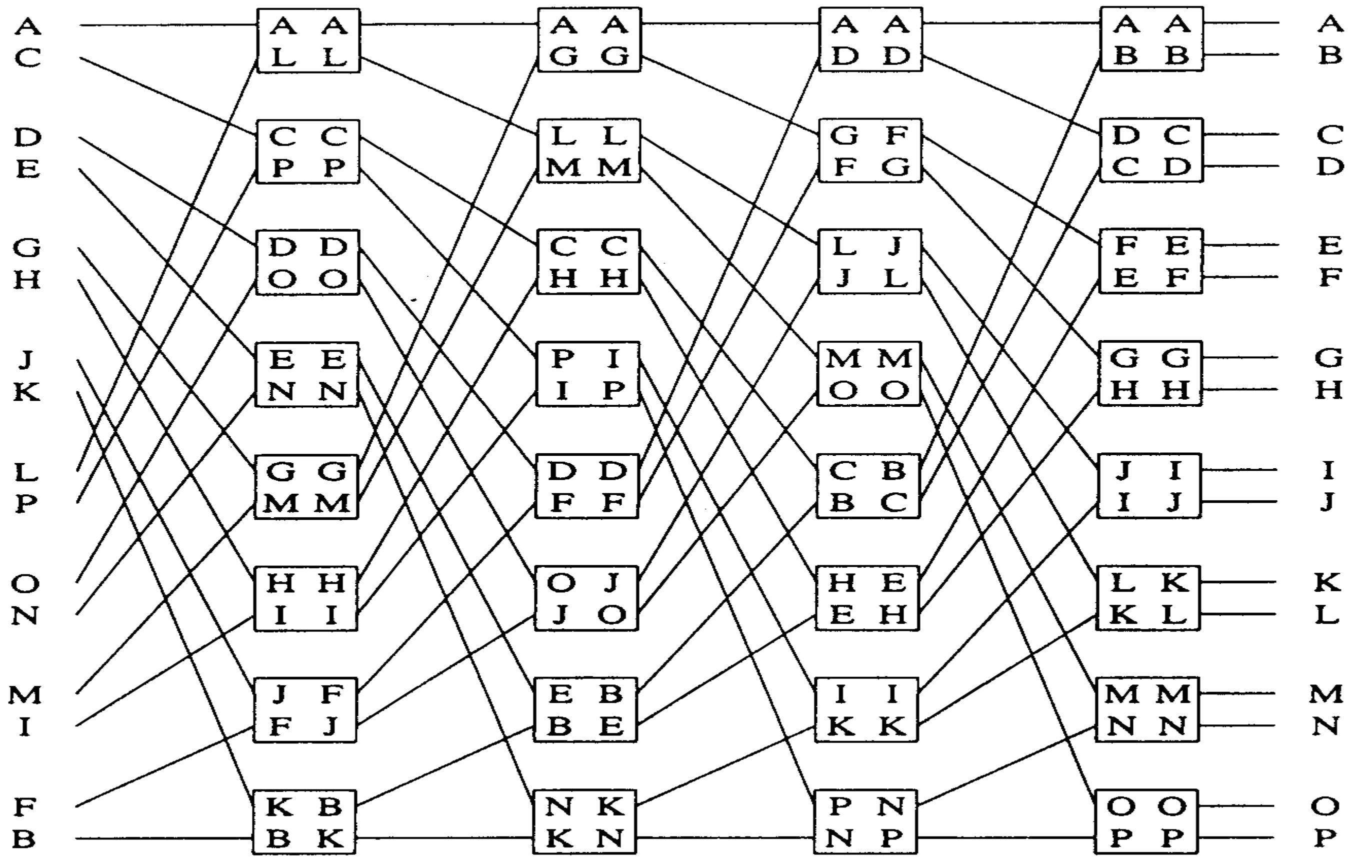
After sixth iteration



After seventh iteration

Parallel Sorting Algorithms

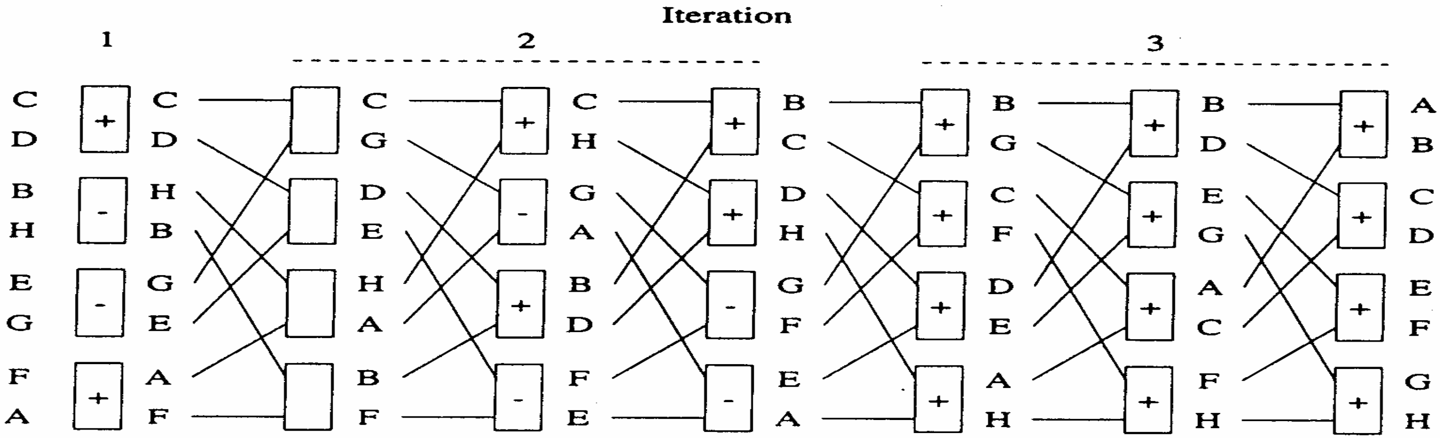
– Bitonic Merge–Sort



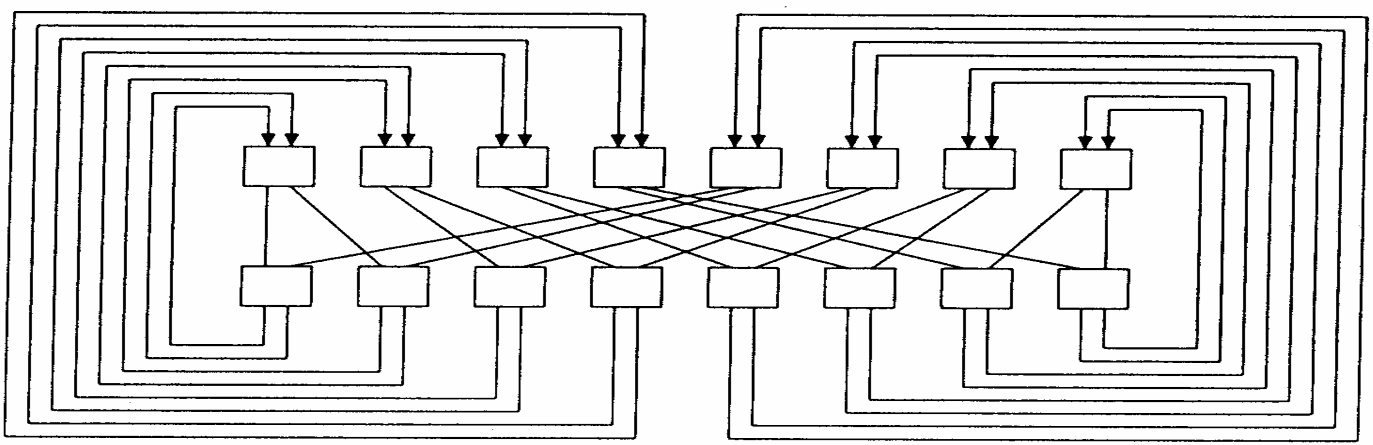
Sorting a bitonic sequence of length 16 by using Stone's perfect shuffle.

Parallel Sorting Algorithms

– Bitonic Merge–Sort



Bitonic merge-sort of an unsorted list of eight elements, by using Stone's perfect shuffle interconnection.



Sorting machine based upon perfect shuffle connection (Sedgewick 1983).

Parallel Sorting Algorithms

– Bitonic Merge–Sort

```
BITONIC MERGE SORT (SHUFFLE-EXCHANGE PROCESSOR ARRAY):

    Parameter     n        {Size of array}
    Global        j, k
    Local         a        {Element to be sorted}
                  m        {Mask bit that indicates kind of comparison to perform}
                  r        {Bit used to compute mask bit}
  begin
    {Compute initial value of the mask M}
    for all P_i where 0 ≤ i ≤ n − 1 do
      r ← i modulo 2
      m ← r
    endfor
    for k ← 1 to log n do
      for all P_i where 0 ≤ i ≤ n − 1 do
        m ← m ⊕ r   {Exclusive OR}
        shuffle(m) ⇐ m
      endfor
    endfor

    {Now do the sort}
    COMPARE-EXCHANGE (a, m)
    for k ← 1 to log n − 1 do
      for all P_i where 0 ≤ i ≤ n − 1 do
        shuffle(r) ⇐ r
        m ← m ⊕ r   {Exclusive OR}
        for j ← 1 to log n − k − 1 do
          shuffle(a) ⇐ a
          shuffle(m) ⇐ m
        endfor
      endfor
      for j ← log n − k to log n do
        for all P_i where 0 ≤ i ≤ n − 1 do
          shuffle(a) ⇐ a
          shuffle(m) ⇐ m
        endfor
        COMPARE-EXCHANGE (a, m)
      endfor
    endfor
  end
```

FIGURE 10-16 Implementation of bitonic merge-sort algorithm on the shuffle-exchange SIMD model.

Parallel Sorting Algorithms

– Bitonic Merge–Sort

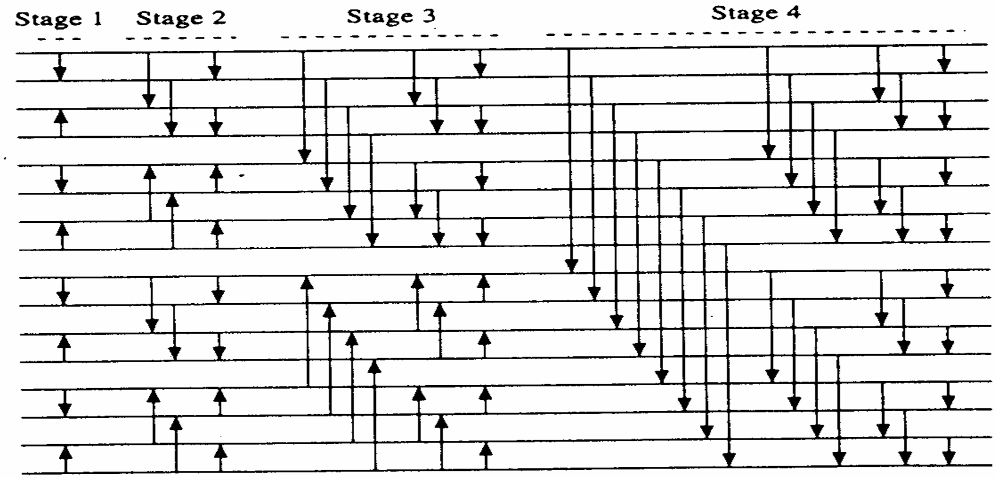
```
COMPARE-EXCHANGE (a, m):

   Reference    a      {Element of list to be sorted}
                m      {Mask bit indicating sort order}
                t      {Value retrieved from successor processor element}
 begin
   for all P_i where 0 ≤ i ≤ n − 1 do
     if even(i) then
       t ⇐ exchange(a)
       if m = 0 then                  {Sort low to high}
         exchange(a) ⇐ max(a,t)
         a ← min(a, t)
       else                           {Sort high to low}
         exchange(a) ⇐ min(a,t)
         a ← max(a, t)
       endif
     endif
   endfor
 end
```

FIGURE 10-17 Compare-exchange routine called by bitonic merge sort algorithm for shuffle-exchange processor array. The even-numbered processing elements assume the role of comparators.

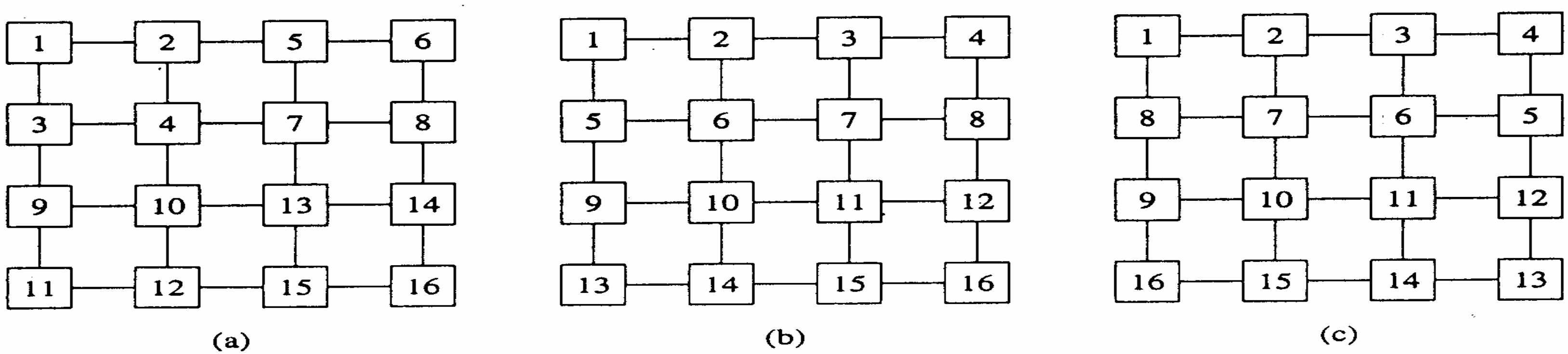
Parallel Sorting Algorithms

– Bitonic Merge–Sort on 2D Mesh



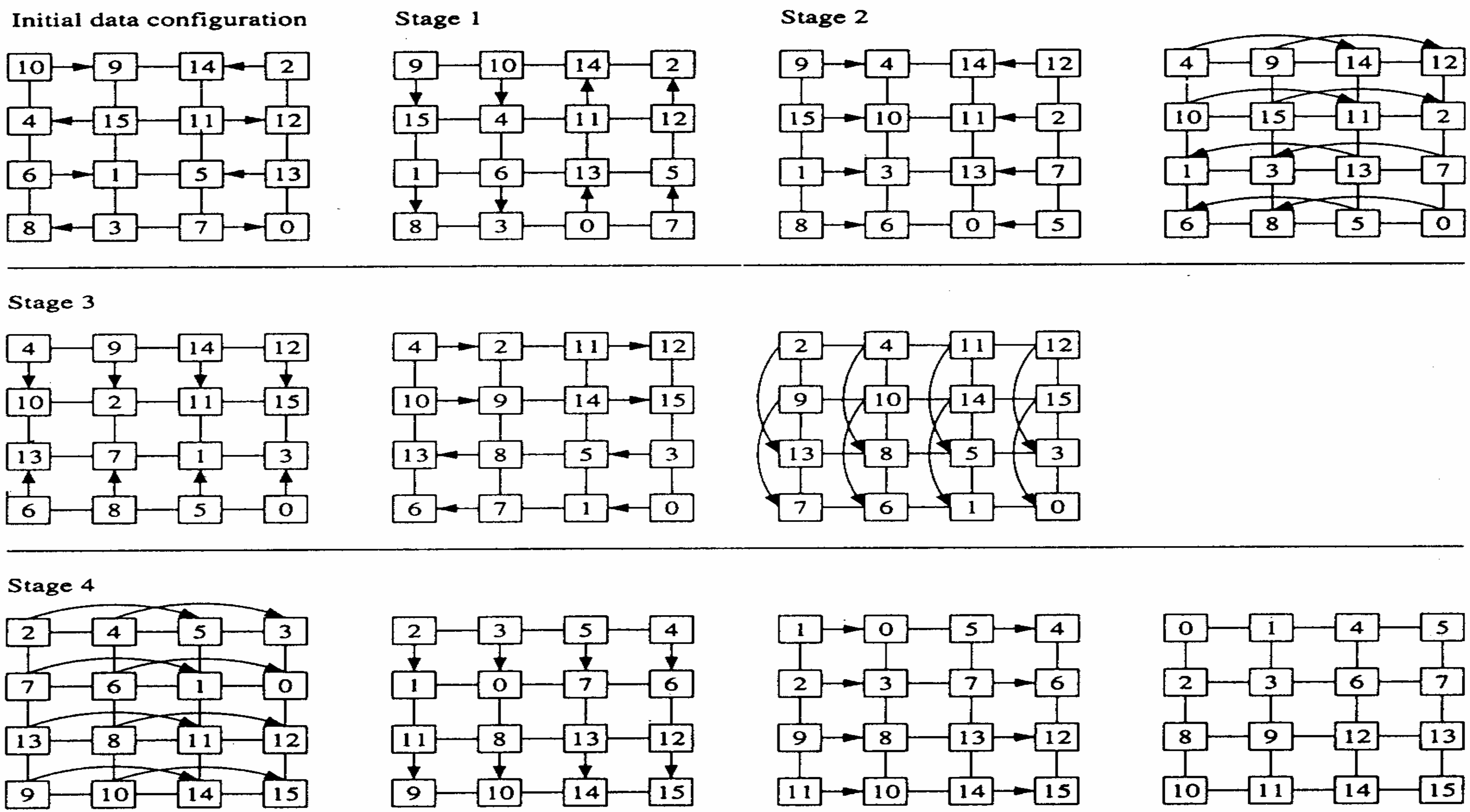
A sorting network based on bitonic merge. (Knuth 1973.)

Three index functions mapping list elements into a two-dimensional mesh. (a) Shuffled row-major order. (b) Row-major order. (c) Snakelike row-major order.



Parallel Sorting Algorithms

– Bitonic Merge–Sort on 2D Mesh



Sorting values into shuffled row-major order on the two-dimensional mesh processor array model. (Thompson and Kung (1977). Copyright ©1986 Association for Computing Machinery. Reprinted by permission.)

Parallel Sorting Algorithms

– Bitonic Merge–Sort on Hypercube

BITONIC MERGE SORT (HYPERCUBE PROCESSOR ARRAY):

```
  Global    d      {Distance between elements being compared}
  Local     a      {One of the elements to be sorted}
            t      {Element retrieved from adjacent processor}
begin
  for i ← 0 to m − 1 do
    for j ← i downto 0 do
      d ← 2^j
      for all P_k where 0 ≤ k ≤ 2^m − 1 do
        if k mod 2d < d then
          t ⇐ [k + d]a {Get value from adjacent processor}
          if k mod 2^(i+2) < 2^(i+1) then
            [k + d]a ⇐ max (t, a)   {Sort low to high...}
            a ← min (t, a)
          else
            [k + d]a ⇐ min (t, a)   {...or sort high to low}
            a ← max (t, a)
          endif
        endif
      endfor
    endfor
  endfor
end
```

Implementation of the bitonic merge-sort algorithm on the hypercube processor array model.