PARALLEL PROCESSING SYSTEMS

Chapter 2: A Taste of Parallel Algorithms

Some simple computations

- five fundamental building-block computations
  - Semigroup (reduction, fan-in) computation
  - Parallel prefix computation
  - Packet routing
  - Broadcasting, and its more general version, multicasting
  - Sorting records in ascending/descending order of their keys

Some simple computations

- Semigroup (reduction, fan-in) computation
  - Let $\otimes$ be an associative binary operator
    - i.e., $(x \otimes y) \otimes z = x \otimes (y \otimes z)$ for all $x, y, z \in S$.
  - A semigroup is simply a pair $(S, \otimes)$
    - where S is a set of elements on which $\otimes$ is defined
  - Semigroup computation is defined as:
    - Given a list of n values x0, x1, . . . , xn–1, compute x0$\otimes$ x1 $\otimes$ . . . $\otimes$ xn–1 .
    - Common examples for the operator $\otimes$ include $+$, $\times$, $\wedge$, $\vee$, $\oplus$, $\cap$, $\cup$, max, min.
    - The operator $\otimes$ may or may not be commutative
      - i.e., it may or may not satisfy $x \otimes y = y \otimes x$

Some simple computations

- Semigroup (reduction, fan-in) computation

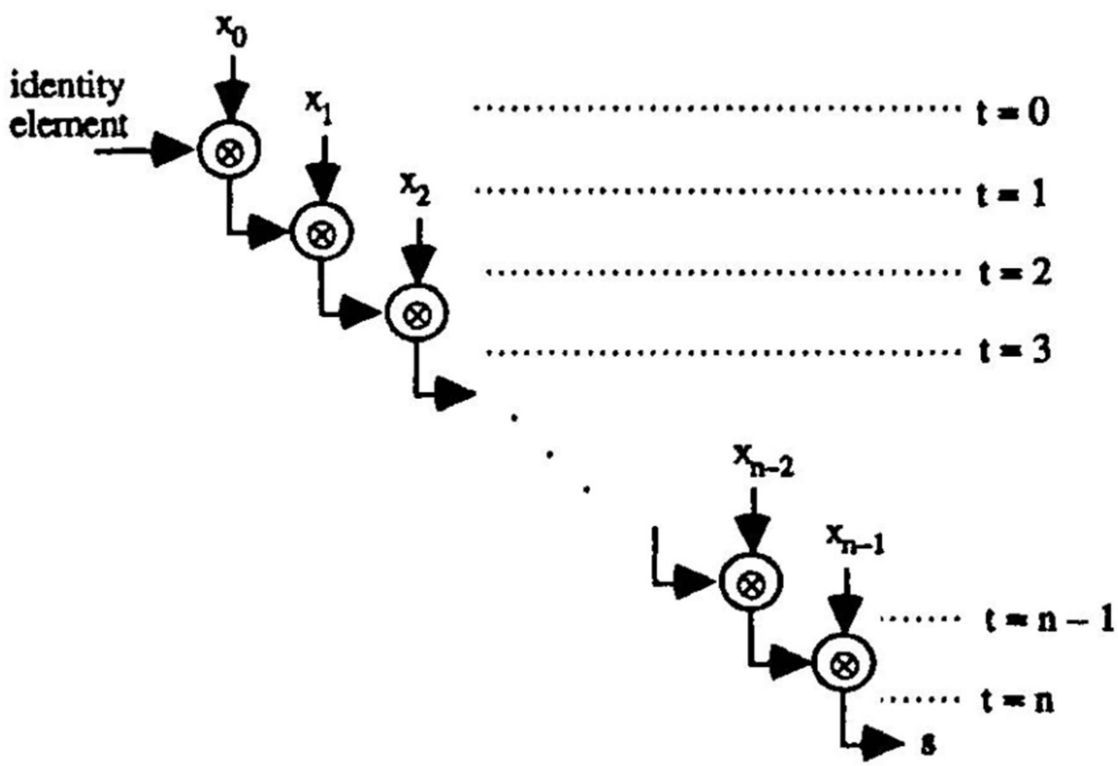


Figure 2.1. Semigroup computation on a uniprocessor.

Some simple computations

- Parallel prefix computation
  - With the same assumptions as in the semigroup
    - a parallel prefix computation is defined as
      - simultaneously evaluating all the prefixes of the expression $x0 \otimes x1 \ldots \otimes xn{-}1$;
        - i.e., $x0$, $x0 \otimes x1$, $x0 \otimes x1 \otimes x2$, . . . , $x\,0 \otimes x1 \otimes \ldots \otimes xn{-}1$.
    - The comment about commutativity of the binary operator $\otimes$ applies here as well.

Some simple computations

- Packet Routing
  - A packet of information resides at Processor i and must be sent to Processor j.
  - The problem is to route the packet through the fastest path
  - The problem becomes more challenging when
    - multiple packets reside at different processors
    - each with its own destination.
    - the packet routes may interfere with one another
      - as they go through common intermediate processors.
  - It is called one-to-one communication or 1–1 routing
    - When each processor has at most one packet to send and one packet to receive

Some simple computations

- Broadcasting
  - Disseminate a value *a* known at a certain processor I to all p processors as quickly as possible
  - sometimes referred to as one-to-all communication.
  - Multicasting is the more general case
    - one-to-many communication

Some simple computations

- Sorting
  - Given a list of n keys x0, x1, . . . , xn–1, and a total order ≤ on key values, rearrange the n keys as

$x_{i_0}, x_{i_1}, \ldots, x_{i_{n-1}}$, such that $x_{i_0} \le x_{i_1} \le \ldots \le x_{i_{n-1}}$

Some simple architectures

- four simple parallel architectures:
  - Linear array of processors
  - Binary tree of processors
  - Two-dimensional mesh of processors
  - Multiple processors with shared variables

Some simple architectures

- Linear array
  - The diameter is $D = p - 1$
    - defined as the longest of the shortest distances between pairs of processors
  - The (maximum) node degree is $d = 2$
    - defined as the largest number of links or communication channels associated with a processor
  - The ring variant has
    - the same node degree of 2
    - a smaller diameter of $D = \lfloor p/2 \rfloor$.

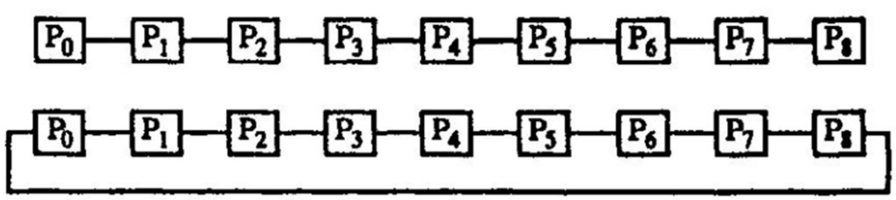


Figure 2.2. A linear array of nine processors and its ring variant.

Some simple architectures

- Binary Tree
  - The binary tree is balanced
    - if leaf levels differ by at most 1.
    - Diameter is $2\lfloor \log_2 p \rfloor$ or $2\lfloor \log_2 p \rfloor - 1$
  - The binary tree is complete
    - If all leaf levels are identical and every non-leaf processor has two children
    - diameter is $2 \log_2 (p + 1) - 2$
  - The (maximum) node degree in a binary tree is d = 3

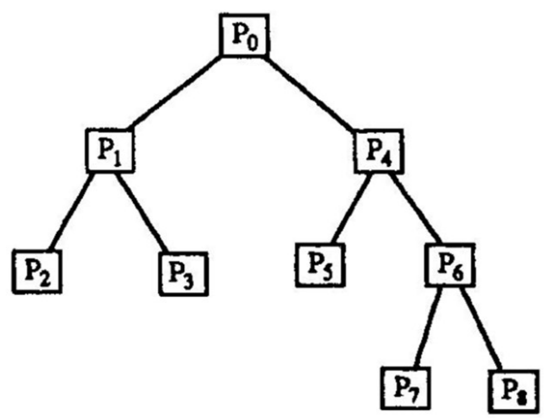


Figure 2.3. A balanced (but incomplete) binary tree of nine processors

Some simple architectures

- 2D Mesh
  - The diameter of a square mesh is $2\sqrt{p} - 2$
  - the mesh does not have to be square.
  - The diameter of a p-processor $r \times (p/r)$ mesh is $D = r + p/r - 2$.
  - multiple 2D meshes may exist for the same number p of processors,
    - e.g., 2×8 or 4×4.
  - Square meshes are usually preferred
    - because they minimize the diameter.
  - The torus variant has end-around or wraparound links for rows and columns.
  - The node degree for both meshes and torus is $d = 4$.
  - But a p-processor $r \times (p/r)$ torus has a smaller diameter of $D = \lfloor r/2 \rfloor + \lfloor p/(2r) \rfloor$.

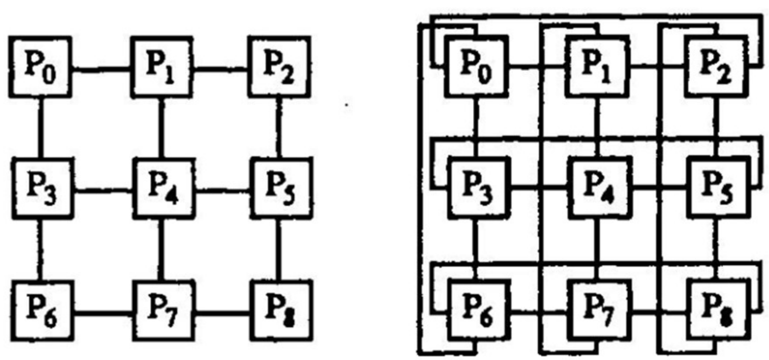
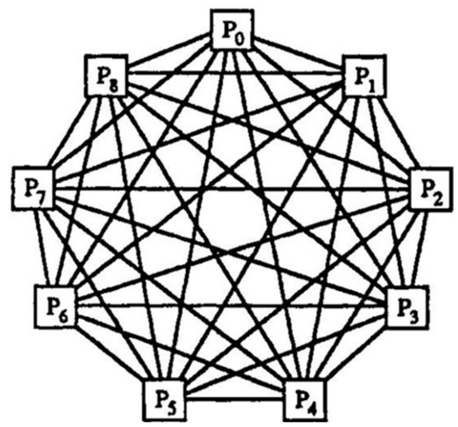


Figure 2.4. A 2D mesh of nine processors and its torus variant.

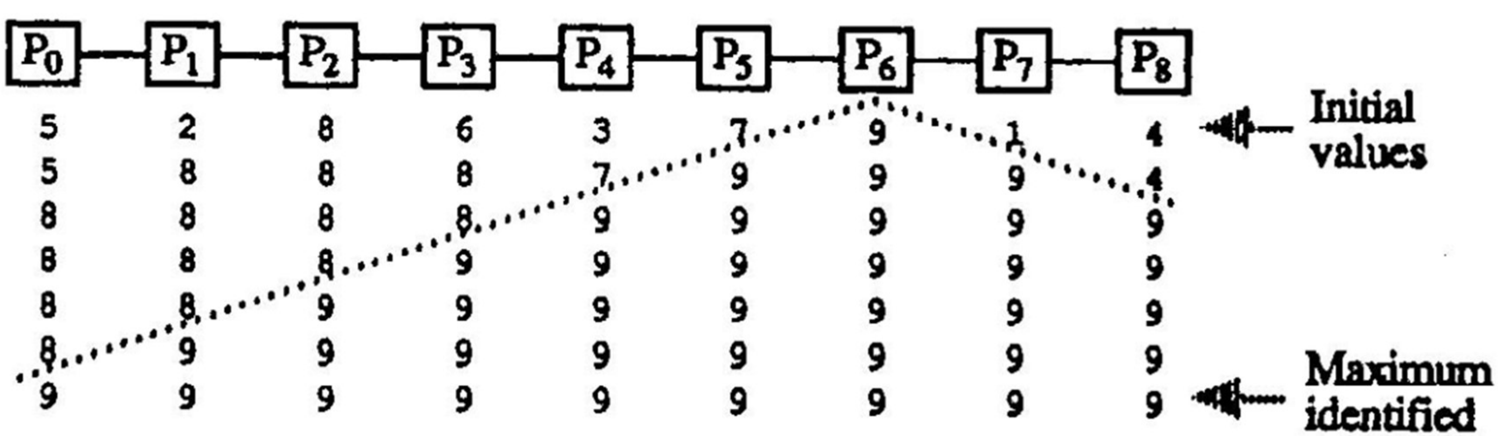
Some simple architectures

- Shared Memory
  - can be modeled as a complete graph
  - every piece of data is directly accessible to every processor
    - assuming each processor can simultaneously send/receive data over all of its p – 1 links
    - The diameter D = 1 is an indicator of this direct access.
  - The node degree d = p – 1
    - indicates that is quite costly to implement
      - if no restriction is placed on data accesses



Algorithms for a linear array

- Semigroup Computation
  - Finding maximum
    - Each of the p processors holds a value initially
    - our goal is for every processor to know the largest of these values.
    - In each step
      - a processor sends its max value to its two neighbors.
    - on receiving values from its left and right neighbors
      - Each processor sets its max value to the largest of the three values, i.e., max(left, own, right)
    - The dotted lines show how the maximum value propagates from P6 to all other processors
      - Had there been two maximum values, say in P2 and P6 , the propagation would have been faster.
      - Needed operations in the worst case
        - p – 1 communication steps
          - each involving sending a processor's value to both neighbors
          - This is the best one can hope for
            - given that the diameter of a p-processor linear array is D = p – 1
        - the same number of three-way comparison steps



Algorithms for a linear array

- Semigroup Computation
  - For a general semigroup computation
    - initially, all processors are dormant or inactive
    - the processor at the left end becomes active and sends its data value to the right
    - On receiving a value from its left neighbor, a processor
      - becomes active
      - applies the semigroup operation $\otimes$ to the value received from the left and its own data value
      - sends the result to the right
      - becomes inactive again
      - This wave of activity propagates to the rightmost processor
    - The computation result is then propagated leftward to all processors
    - In all, $2p - 2$ communication steps are needed

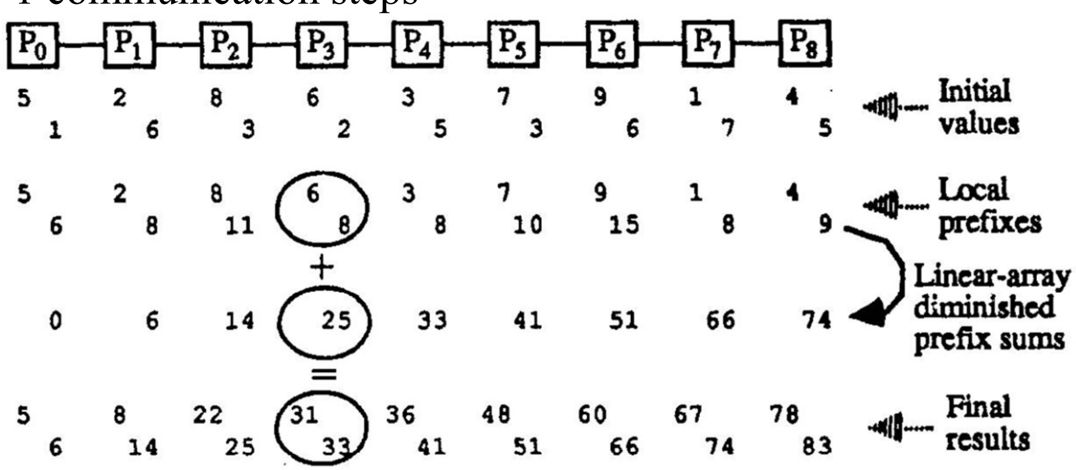
Algorithms for a linear array

- Parallel Prefix Computation
  - we want the ith prefix result at the ith processor, $0 \le i \le p - 1$
    - we already have an algorithm
      - The general semigroup algorithm without the last broadcast
      - takes p – 1 communication/combining steps.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	
5	2	8	6	3	7	9	1	4	Initial values
5	7	8	6	3	7	9	1	4	
5	7	15	6	3	7	9	1	4	
5	7	15	21	3	7	9	1	4	
5	7	15	21	24	7	9	1	4	
5	7	15	21	24	31	9	1	4	
5	7	15	21	24	31	40	1	4	
5	7	15	21	24	31	40	41	4	
5	7	15	21	24	31	40	41	45	Final results

Algorithms for a linear array

- Extension of the semigroup and parallel prefix algorithms
  - each processor initially holds several data
  - The algorithm consists of each processor doing
    - a prefix computation on its own data set of size n/p
      - takes n/p – 1 combining steps
    - a diminished parallel prefix computation
      - each processor holds onto the value received from the left
      - Takes p – 1 communication/combining steps
    - finally combining the result of two prefix computations
      - Takes n /p combining steps
  - Number of operations required in all
    - 2n/p + p– 2 combining steps
    - p – 1 communication steps



Algorithms for a linear array

- Packet Routing
  - To send a packet of information from Processor i to Processor j
    - Attach a routing tag with the value j – i to it
      - The sign determines the direction of move
        - (+ = right, – = left)
      - The magnitude indicates the action to be performed
        - (0 = remove the packet, nonzero = forward the packet).
    - With each forwarding
      - the magnitude of the routing tag is decremented by 1

Algorithms for a linear array

- Broadcasting
  - If Processor i wants to broadcast a value a to all processors
    - it sends
      - an rbcast(a) (read r-broadcast) message to its right neighbor
      - an lbcast(a) message to its left neighbor
    - receiving an rbcast(a) message, any processor
      - copies the value a and forwards the message to its right neighbor (if any).
    - receiving an lbcast(a) message, any processor
      - copies the value a and forwards the message to its left neighbor (if any)
    - The worst-case number of communication steps is $p - 1$

Algorithms for a linear array

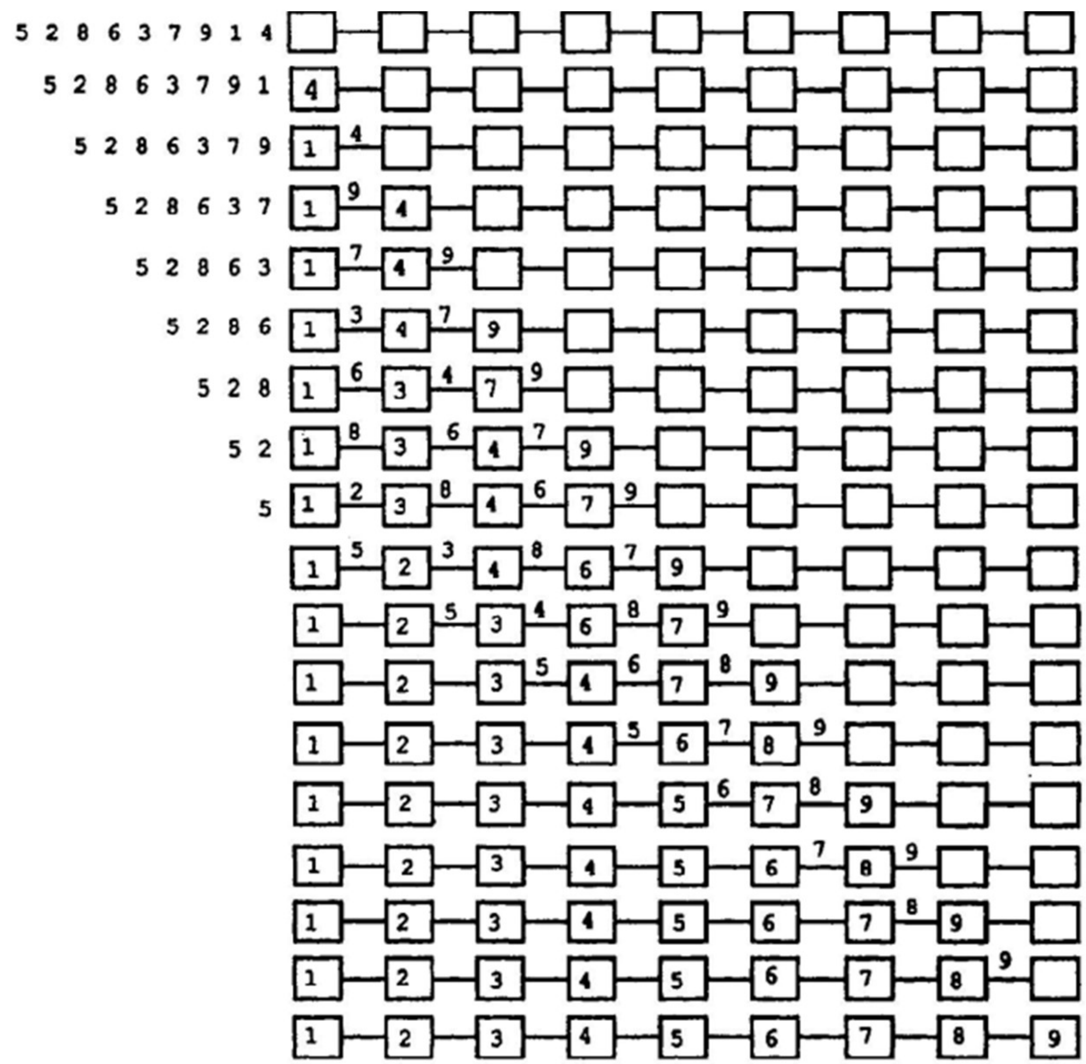
- Sorting
  - two versions of sorting on a linear array
    - with I/O
    - without I/O

Algorithms for a linear array

- Sorting
  - with I/O
    - p keys are input, one at a time, from the left end
    - Each processor, on receiving a key value from the left
      - compares the received value with the value stored in its local register
        - initially, all local registers hold the value $+\infty$
      - The smaller of the two values is kept in the local register
      - larger value is passed on to the right
    - Once all p inputs have been received
      - we must allow p – 1 additional communication cycles for the key values that are in transit to settle into their respective positions
    - If the sorted list is to be output from the left
      - the output phase can start immediately after the last key value has been received
      - an array half the size of the input list would be adequate
      - we effectively have zero-time sorting
        - the total sorting time is equal to the I/O time

Algorithms for a linear array

- Sorting
  - with I/O

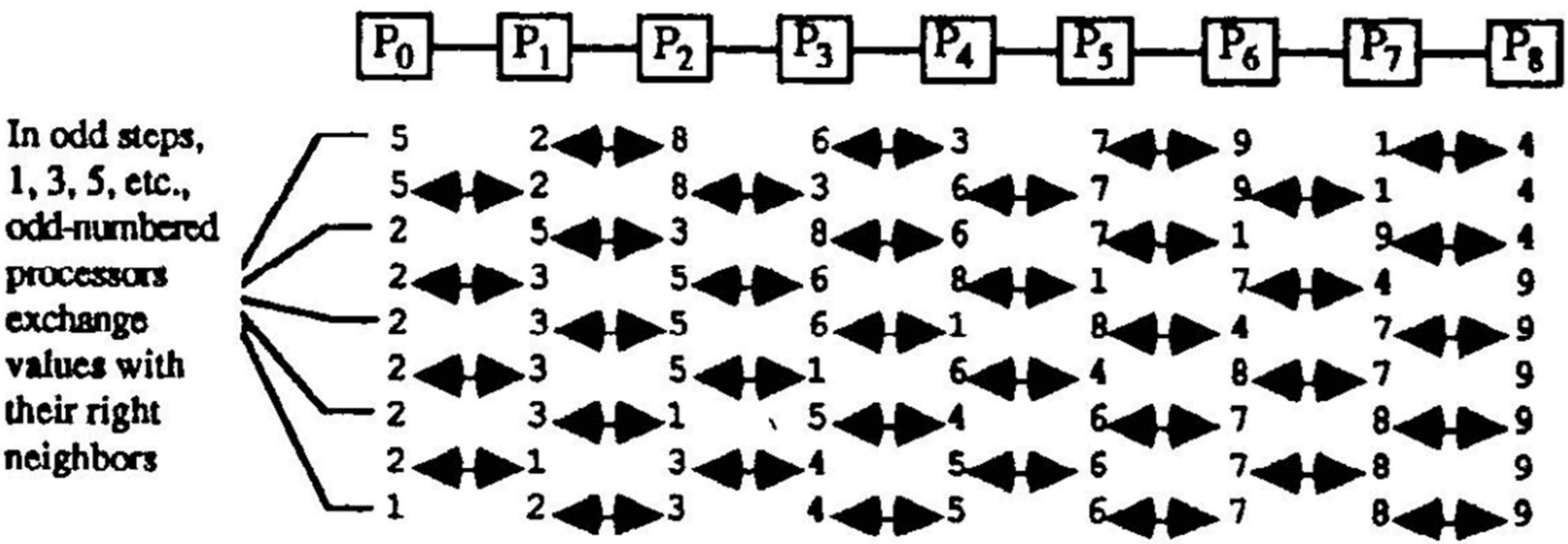


Algorithms for a linear array

- Sorting
  - without I/O
    - the key values are already in place, one per processor
    - an algorithm known as odd–even transposition can be used
      - A total of p steps are required.
      - In an odd-numbered step
        - odd-numbered processors compare values with their even-numbered right neighbors
        - The two processors exchange their values if they are out of order.
      - in an even-numbered step
        - even-numbered processors compare–exchange values with their right neighbors
      - In the worst case
        - the largest key value
          - resides in Processor 0
          - must move all the way to the other end of the array
        - This needs p – 1 right moves

Algorithms for a linear array

- Sorting
  - without I/O



Algorithms for a linear array

- Sorting with the number n of keys is greater than the number p of processors
  - the odd–even algorithm with n/p keys per processor
    - each processor sorts its list using any efficient sequential sorting algorithm.
      - Let us say this takes (n/p)log2(n/p) compare–exchange steps.
    - the odd–even transposition sort is performed as before
      - except that each compare–exchange step is replaced by a merge–split step
        - the two processors merge their sublists of size n/p into a single sorted list of size 2n/p
        - then split the list down the middle
        - one processor keeping the smaller half
        - the other keeps the larger half

Algorithms for a linear array

- Sorting with the number n of keys is greater than the number p of processors
  - the odd–even algorithm with n/p keys per processor
    - E.g., if P0 is holding (1, 3, 7, 8) and P1 has (2, 4, 5, 9)
      - a merge–split step will turn the lists into (1, 2, 3, 4) and (5, 7, 8, 9), respectively
    - Because the sublists are sorted
      - the merge–split step requires n/p compare–exchange steps.
      - the total time of the algorithm is $(n/p)\log_2 (n/p) + n$
      - the first term (local sorting) will be dominant if $p < \log_2 n$
      - the second term (array merging) is dominant for $p > \log_2 n$.
      - For $p \geq \log_2 n$
        - the time complexity of the algorithm is linear in n
        - the algorithm is more efficient than the one-key-per-processor version

Algorithms for a linear array

- One final observation about sorting
  - it is important
  - occasionally it also helps us in data routing
  - permutation routing problem
    - data values being held by the p processors are to be routed to other processors
    - such that the destination of each value is different from all others.
    - the p distinct destinations must be 0, 1, 2, . . . , p – 1
    - The correct destination for each record can be find by
      - forming records with the destination address as the key
      - sorting these records
    - p compare–exchange steps.
    - Effectively
      - p packets are routed in the same amount of time that is required for routing a single packet in the worst case

Algorithms for a binary tree

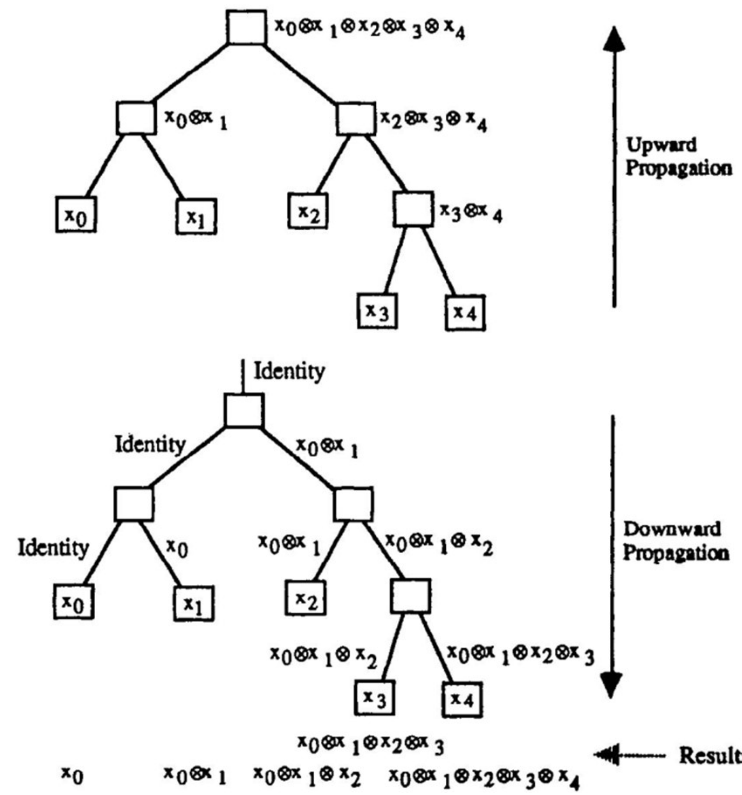
- we assume that the data elements are initially held by the leaf processors only
- The nonleaf (inner) processors participate in the computation
  - but do not hold data elements of their own
- This simplifying assumption
  - can be easily relaxed
  - leads to simpler algorithms
  - Does not pose great inefficiency
    - Because roughly half of the tree nodes are leaf nodes

Algorithms for a binary tree

- Semigroup Computation
  - binary-tree architecture is ideally suited for this
    - semigroup computation is sometimes referred to as tree computation
  - Each inner node
    - receives two values from its children
      - if each of them has already computed a value or is a leaf node
    - applies the operator to them
    - passes the result upward to its parent
  - After $\lfloor \log 2\ p \rfloor$ steps, the root processor will have the computation result
  - All processors can then be notified of the result through a broadcasting operation from the root.
  - Total time: $2\lfloor \log 2p \rfloor$ steps

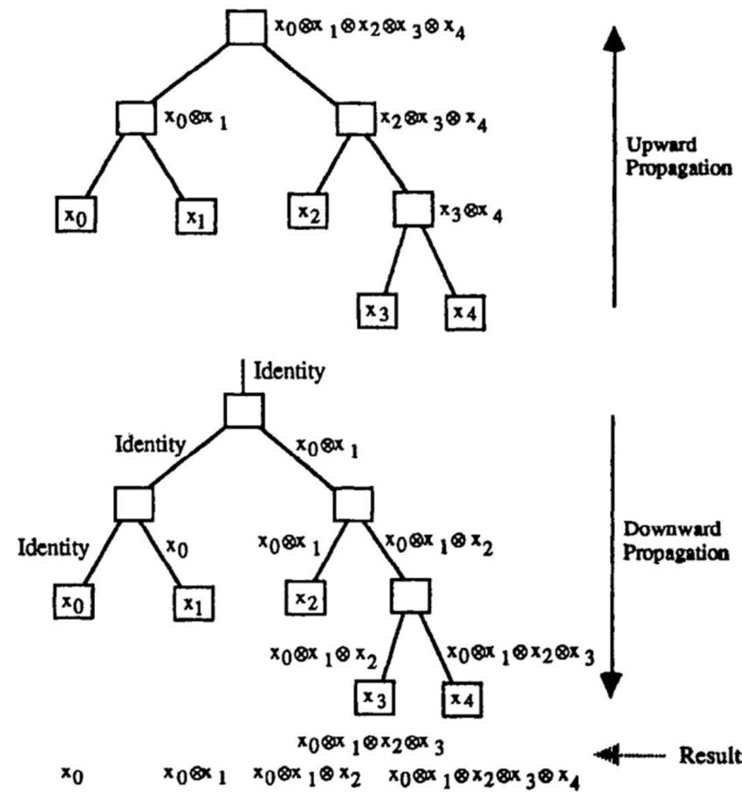
Algorithms for a binary tree

- Parallel Prefix Computation
  - Again
    - this is quite simple
    - can be done optimally in $2\lfloor \log_2 p \rfloor$ steps



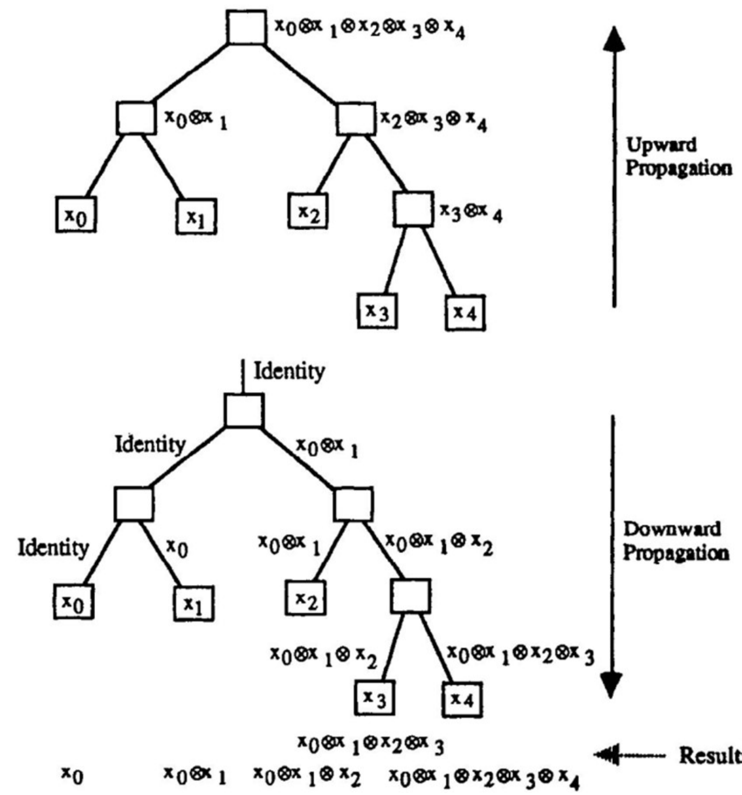
Algorithms for a binary tree

- Parallel Prefix Computation
  - upward propagation phase
    - identical to the upward movement of data in semigroup computation
    - At the end, each node will have the semigroup computation result for its subtree



Algorithms for a binary tree

- Parallel Prefix Computation
  - downward phase
    - Each processor remembers the value it received from its left child.
    - On receiving a value from the parent, a node passes
      - the value received from above to its left child and
      - combination of this value and the one that came from the left child to its right child.
    - The root initiates the downward phase by sending
      - the identity element to the left
      - the value received from its left child to the right.
    - At the end
      - the leaf processors compute their respective results



Algorithms for a binary tree

- Some applications of Parallel Prefix Computation
  - Given a list of 0s and 1s
    - the rank of each 1 in the list (its relative position among the 1s) can be determined by a prefix sum computation

Data:	0	0	1	0	1	0	0	1	1	1	0
Prefix sums:	0	0	1	1	2	2	2	3	4	5	5
Ranks of 1s:			1		2			3	4	5	

Algorithms for a binary tree

- Some applications of Parallel Prefix Computation
  - priority circuit
    - has a list of 0s and 1s as its inputs
    - picks the first (highest-priority) 1 in the list.
  - The function of a priority circuit can be defined as:

Data:	0	0	1	0	1	0	0	1	1	1	0
Diminished prefix logical ORs:	0	0	0	1	1	1	1	1	1	1	1
Complement:	1	1	1	0	0	0	0	0	0	0	0
AND with data:	0	0	1	0	0	0	0	0	0	0	0

Algorithms for a binary tree

- Packet Routing
  - The algorithm depends on the processor numbering scheme
  - "preorder" indexing
    - Nodes in a subtree are numbered by
      - first numbering the root node
      - then its left subtree
      - and finally the right subtree
    - the index of each node is less than of all its descendants.
    - We assume that each node
      - Is aware of its own index (self) in the tree
      - knows the largest node index in its left (maxl) and right (maxr) subtrees.

Algorithms for a binary tree

- Packet Routing
  - "preorder" indexing
    - Routing algorithm for a packet
      - on its way from node i to node dest
      - currently residing in node self
    - This algorithm does not make any assumption about the tree except that it is a binary tree.
    - the tree need not be complete or even balanced

if $dest = self$
then remove the packet {done}
else if $dest < self$ or $dest > maxr$
    then route upward
    else if $dest \le maxl$
        then route leftward
        else route rightward
        endif
    endif
endif

Algorithms for a binary tree

- Broadcasting
  - Processor i sends the desired data upwards to the root processor
  - The root then broadcasts the data downwards to all processors

Algorithms for a binary tree

- Sorting
  - algorithm is similar to bubblesort
    - first, smaller elements in the leaves “bubble up” to the root processor
    - root “sees” all the data elements in nondescending order.
    - root then sends the elements to leaf nodes in the proper order.

Algorithms for a binary tree

- Sorting
  - upward movement
    - Initially
      - each leaf has a single data item
      - all other nodes are empty.
    - Each inner node has storage space for two values
      - migrating upward from its left and right subtrees.

```
if you have 2 items
then do nothing
else if you have 1 item that came from the left (right)
    then get the smaller item from the right (left) child
    else get the smaller item from each child
    endif
endif
```

Algorithms for a binary tree

- Sorting
  - upward movement up to the point when
    - the smallest element is in the root node
    - ready to begin its downward movement

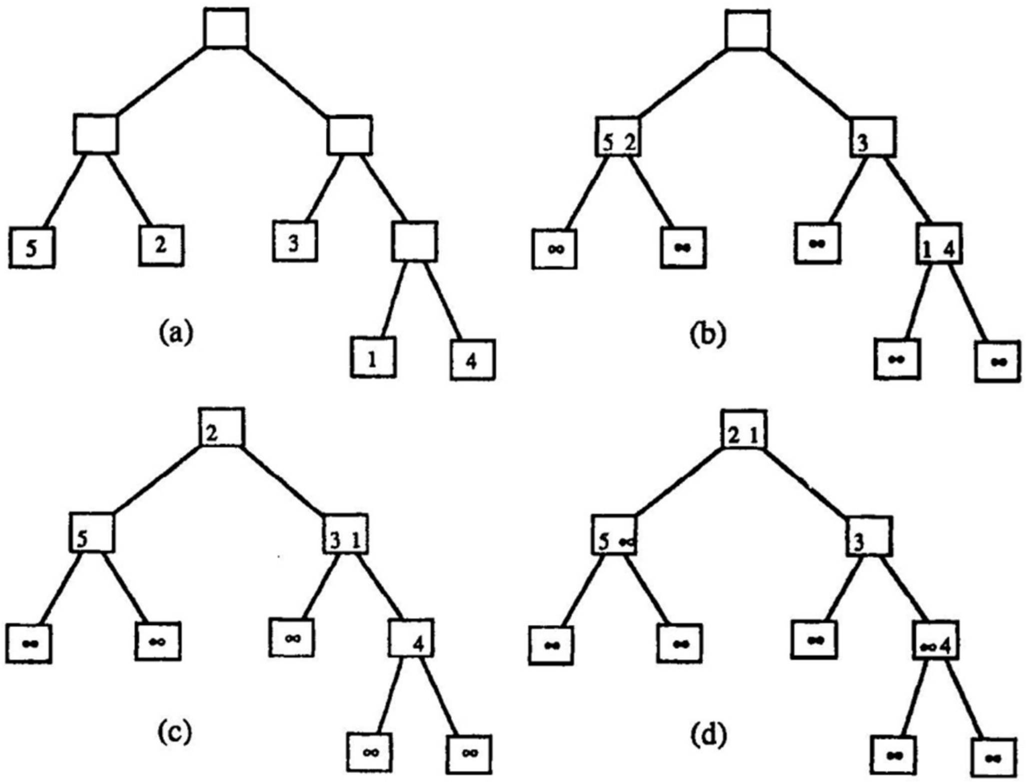


Figure 2.12. The first few steps of the sorting algorithm on a binary tree.

Algorithms for a binary tree

- Sorting
  - downward movement
    - coordinated if each node knows the number of leaf nodes in its left subtree.
    - For an element received from above
      - Keep the rank order in a local counter
      - If the rank order <= the number of leaf nodes to the left,
        - then the data item is sent to the left.
      - Otherwise,
        - it is sent to the right.
    - implicitly assumes that data are to be sorted from left to right in the leaves.

Algorithms for a binary tree

- Sorting
  - takes linear time in the number of elements to be sorted
    - reasoning based on a bisection-based lower bound:
      - partition a tree architecture into two equal or almost equal halves
      - in the worst case
        - all values in the left (right) half of the tree must move to the right (left) half
        - Hence, all data elements must pass through the single link
        - it takes linear time for all the data elements to pass through this bottleneck

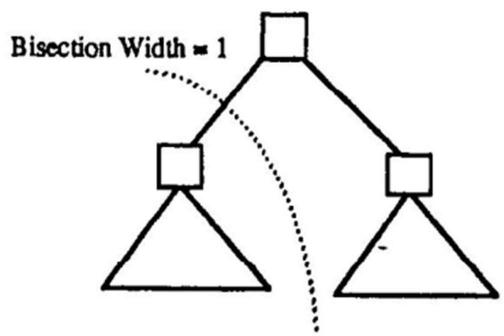


Figure 2.13. The bisection width of a binary tree architecture.

Algorithms for a 2d mesh

- Semigroup Computation
  - do the semigroup computation
    - in each row
    - then in each column.
  - E.g., finding the maximum of a set of p values, stored one per processor
    - the row maximums
      - are computed
      - made available to every processor in the row.
    - Then column maximums are identified.

Algorithms for a 2d mesh

- Parallel Prefix Computation.
  - can be done in three phases
    - assuming that the processors (and their stored values) are indexed in row-major order
    1. do a parallel prefix computation on each row
    2. do a diminished parallel prefix computation in the rightmost column
    3. broadcast the results in the rightmost column to all the elements in the respective rows
       - combine with the initially computed row prefix value.
  - E.g., in doing prefix sums
    - first-row prefix sums are computed from left to right
      - At this point, the processors in the rightmost column hold the row sums
    - A diminished prefix computation in this last column yields the sum of all the preceding rows in each processor
    - Combining the sum of all the preceding rows with the row prefix sums yields the overall prefix sums.

Algorithms for a 2d mesh

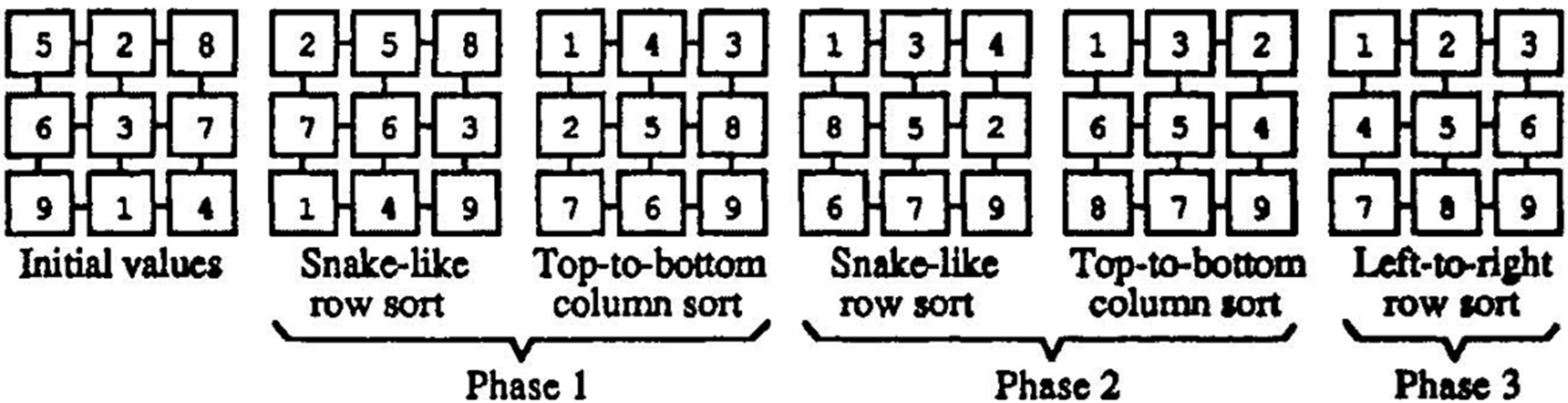
- Packet Routing
  - To route a data packet from the processor in Row r, Column c, to the processor in Row r', Column c’,
    - we first route it within Row r to Column c’.
    - Then, we route it in Column c' from Row r to Row r’.
    - This algorithm is known as row-first routing.
  - Clearly, we could do
    - column-first routing
    - or use a combination of horizontal and vertical steps to get to the destination node along a shortest path
  - When multiple packets must be routed between different source and destination nodes
    - the above algorithm can be applied to each packet independently of others
    - However, multiple packets might then compete for the same link
    - The processors must have sufficient buffer space to store the waiting packets

Algorithms for a 2d mesh

- Broadcasting
  - is done in two phases:
    1. broadcast the packet to every processor in the source node's row
    2. broadcast in all columns.
  - This takes at most $2\sqrt{p} - 2$ steps.

Algorithms for a 2d mesh

- Sorting
  - the simple version of a sorting algorithm known as shearsort
    - consists of $\lceil \log_2 r \rceil + 1$ phases in a 2D mesh with r rows.
    - In each phase, except for the last one
      - all rows are independently sorted in a snakelike order:
        - even-numbered rows 0, 2, … from left to right
        - odd-numbered rows 1, 3, . . . from right to left
      - Then, all columns are independently sorted from top to bottom.
    - E.g., in a 3 × 3 mesh, two such phases are needed
    - In the final phase, rows are independently sorted from left to right



Algorithms for a 2d mesh

- Sorting
  - the simple version of a sorting algorithm known as shearsort
    - we already know that row-sort and column-sort on a p-processor square mesh take $\sqrt{p}$ compare-exchange steps
    - the shearsort algorithm needs $(2\lceil \log_2 p \rceil + 1)\sqrt{p}$ exchange steps for sorting in row-major order.

Algorithms with shared variables

- Semigroup Computation
  - Each processor
    - obtains the data items from all other processors
    - performs the semigroup computation independently
  - all processors will end up with the same result
  - This approach is quite wasteful of the complex architecture
    - because its linear time complexity is
      - comparable to that of the semigroup computation algorithm for the much simpler linear-array architecture
      - worse than the algorithm for the 2D mesh.

Algorithms with shared variables

- Parallel Prefix Computation
  - Like the semigroup computation
  - except that each processor only obtains data items from processors with smaller indices
- Packet Routing
  - Trivial in view of the direct communication path between any pair of processors.
- Broadcasting
  - Trivial, as each processor can send a data item to all processors directly

Algorithms with shared variables

- Sorting
  - The algorithm has two phases
    - ranking
      - determining the relative order of each key in the final sorted list
      - Processor i is responsible for ranking its own key xi.
        - done by
          - comparing xi to all other keys
          - counting the number of keys that are smaller than xi
    - data permutation
      - If each processor holds one key
        - the jth-ranked key can be sent to Processor j
        - requiring a single parallel communication step

Algorithms with shared variables

- Sorting
  - Despite the greater complexity over the linear-array or binary-tree
    - the required linear time is wasteful
      - It is comparable to the algorithms for these simpler architectures.
  - We will see (in Chapter 6) that
    - logarithmic-time sorting algorithms can be developed for the shared variable architecture
      - leading to linear speed-up over sequential algorithms
        - that need n log n steps to sort n items.