Parallel Mesh Algorithms

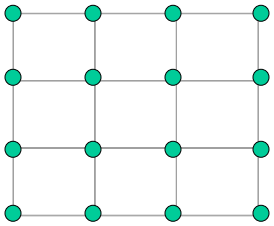
Reference : Horowitz, Sahni and Rajasekaran, *Computer Algorithms*

Computation Model

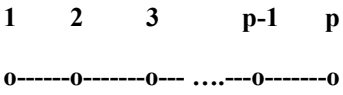
- A mesh is an $a \times b$ grid in which there is a processor at each grid point
- The edges are bi-directional communication links, i.e. two separated uni-directional links
- Each processor can be labeled with a tuple (i, j)
- Each processor has some local memory, and can perform basic operations

- There is a global clock which synchronize all processors
- We only consider square meshes here, i.e. $a = b$, and linear array

A 4 × 4 mesh (16 processors)



A Linear array of p processors



Packet Routing

- Primitive Interprocessor Communication Operation – packet routing
- A packet contains

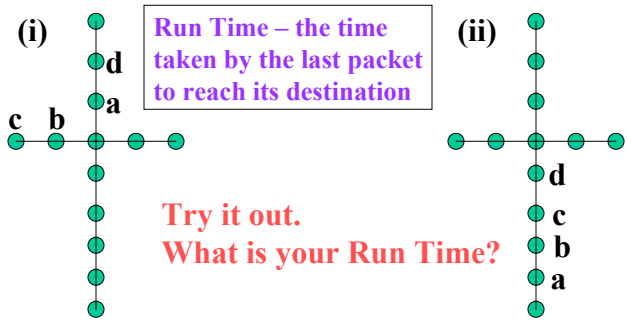
  data + source processor + destination processor
- A link can handle only one packet at one unit time
- A processor may receive multiple packets (from different links) and send multiple packets (to different links) at the same time

- A processor may queue some packets in its local storage
- Each processor uses the same packet routing algorithm
- Partial Permutation Routing (PPR) is a special case of general routing problem. In PPR, each processor is the origin (and destination) of at most one packet.
- The performance of an algorithm is measured by run time, i.e. time to complete all operations, and maximum queue length, i.e. the maximum number of packets in a processor queue.

- The time taken by any packet to reach its destination is dictated by the distance of the chosen path between the packet's origin and destination and the amount of time (referred to as delay) the packet spends waiting in queues.
- A Packet Routing Algorithm is specified by the path to be taken by each packet and a priority scheme.
- Run Time is the time taken by the last packet to reach its destination

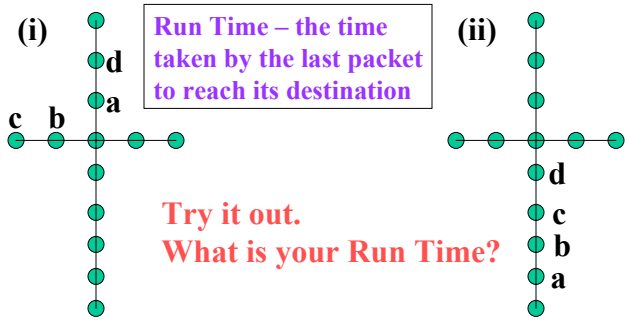
Example: Consider the packets a, b , c and d in (i).
Final destinations are in (ii).
Each packet takes the shortest path to its destination.

(a) Use FIFO Priority Scheme (If two packets reached a node at the same time, there is a tie. This can be broken arbitrary.)



Example: Consider the packets a, b , c and d in (i).
Final destinations are in (ii).
Each packet takes the shortest path to its destination.

(b) Use Farthest-destination-first Scheme (If two packets have the same destination, there is a tie. This can be broken arbitrary.)



• Farthest-destination-first strategy : At each time unit, each processor chooses at most two packets from its queue, one to farthest left (if any) and one to farthest right (if any).

- Packet routing on a linear array of p processors

  Problem 1 : At most one packet originated from each processor. (with arbitrary destinations)

  Problem 1 can be solved in <= p – 1 steps.

  Every processor starts to send packet using the shortest route. There is no contention for any link. The maximum distance is p – 1 links.

Packet routing on a linear array of p processors

Problem 2 : Each processor is the destination for exactly one packet. (can have multiple packets starting from a single origin)

Note : A processor may have multiple packets to be routed to multiple processors.

- Farthest-destination-first strategy : At each time unit, each processor chooses at most two packets from its queue, one to farthest left (if any) and one to farthest right (if any).

- By using the farthest-destination-first strategy, the time needed for a packet stating at processor i to reach its destination is no more than Max (p-i,i-1)

Informal Proof :

Just consider only packets that are moving from left to right. (same for those moving from right to left)

(1) A packet to processor p cannot be delayed according to the strategy, so, it will reach destination at p-i units.

(2) A packet to processor p-1 can only be delayed by packet to p, so it will reach destination at p-1-i units + 1 delayed unit <= p-i units

(3) and so on.

Problem 2 can be solved in <= p – 1 steps using the farthest-destination-first strategy

- Packet routing (PPR) on a Mesh

(Assume p × p processors)

Algorithm PPR:

Let q be an arbitrary packet with (i,j) as its origin and (u,v) as its destination.

Phase 1 : Travel along column j to row u

Phase 2 : Travel along row u to its destination (u,v)

Running time analysis for Algorithm PPR :

The lower bound is 2(p-1) steps, i.e. from (1,1) to (p,p) (opposite corners)

Phase 1 can be done in (p-1) steps (by problem 1 above)

Phase 2 can be done in (p-1) steps (by problem 2 above)

Total : 2(p-1) steps → optimal algorithm

Problem : in the worst case, the queue size is p/2.

i.e. In each time unit, 2 packets arrive and only one can be sent out

There are at most p packets in one column → may need to queue up to p/2 packets

Fundamental Algorithms

- Broadcasting problem : To broadcast a message to all processors

  Assume a processor can duplicate message.

  Phase 1 : Send the message along the row.

  Phase 2 : For each processor in the row, send message along its column.

  Total time <= 2(p-1) = O(p)

- Prefix computation problem for p × p mesh; Total time O(p)

  Assume there is a number in each processor, i.e. p^2 numbers $\{x_{1,1}, x_{1,2}, \ldots, x_{1,p}, x_{2,1}, x_{2,2}, \ldots, x_{2,p}, \ldots, x_{p,1}, x_{p,2}, \ldots, x_{p,p}\}$

  Phase 1 : Each row i, compute prefix

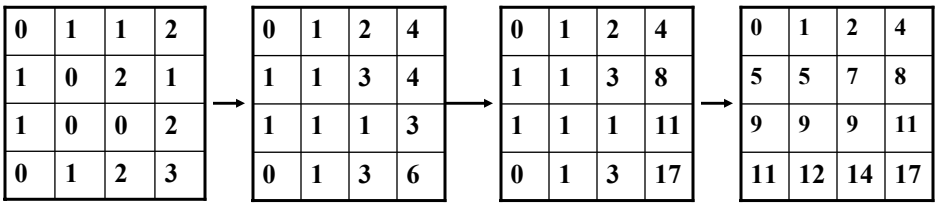
  $$x_{i,1}, x_{i,1} \oplus x_{i,2}, \ldots, x_{i,1} \oplus x_{i,2} \oplus \ldots \oplus x_{i,p}$$

  Phase 2 : Compute prefix for number in column p.

  Assume final numbers in column p are $\alpha_1, \alpha_2, \ldots, \alpha_p$

Phase 3 : For each processor (i,p) in column p, broadcast the number α_{i-1} to all other elements in same row i. The number α_{i-1} will be added to each number in row i.

Example : (assume each small square is a processor in 4 x 4 mesh)



Merging

- Odd-Even Merge on a Linear Array

Assume two sorted lists with 2m numbers :

$a_1\ a_2\ a_3\ a_4 \ldots a_{m-1}\ a_m\ b_1\ b_2\ b_3\ b_4 \ldots b_{m-1}\ b_m$

Assume there are 2m processors and each processor has a number.

1st m processors hold 1st list of m sorted numbers,

2nd m processors hold 2nd list of m sorted numbers.

Step 1: Group odd part and even part of a_i (also, same for b_i)

i.e. $a_1\ a_3\ \ldots a_{m-1}$ | $a_2\ a_4\ \ldots\ a_m$ | $b_1\ b_3\ \ldots b_{m-1}$ | $b_2\ b_4\ \ldots b_m$

Run time : O(m/2)

Step 2 : Group odd parts (and even parts) of both lists

i.e. $a_1\ a_3\ \ldots a_{m-1}$ | $b_1\ b_3\ \ldots b_{m-1}$ | $a_2\ a_4\ \ldots\ a_m$ | $b_2\ b_4\ \ldots b_m$

Odd Parts **Even Parts**

Run time : O(m/2)

Step 3 : Odd parts (and even parts) are merged recursively to get two sorted lists

i.e. $o_1\ o_2\ \ldots o_{m-1}\ o_m$ | $e_1\ e_2\ \ldots e_{m-1}\ e_m$

Run time : T(m/2)

Step 4 : Shuffled odd and even numbers

i.e. $o_1\ e_1\ o_2\ e_2\ o_3\ \ldots\ e_{m-2}\ o_{m-1}\ e_{m-1}\ o_m\ e_m$

Run time : O(m)

Step 5 : Compare adjacent elements and swap numbers if out of order

i.e. o_1 | e_1 o_2 | e_2 o_3 .. | … | e_{m-2} o_{m-1} | e_{m-1} o_m | e_m

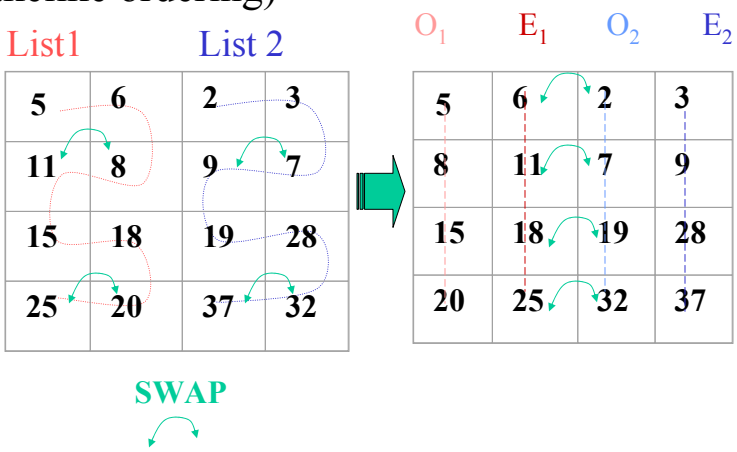
Run time : O(1)

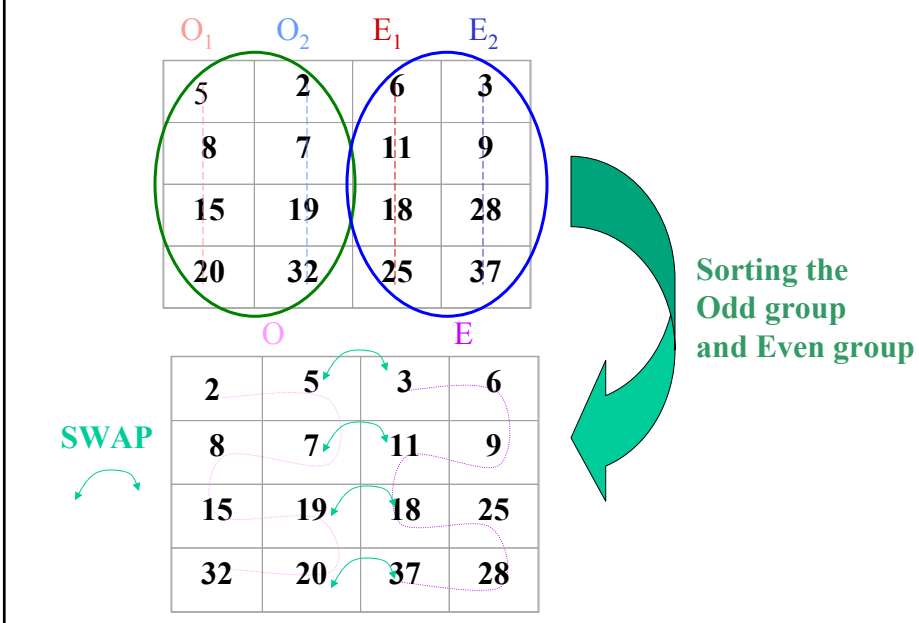
…

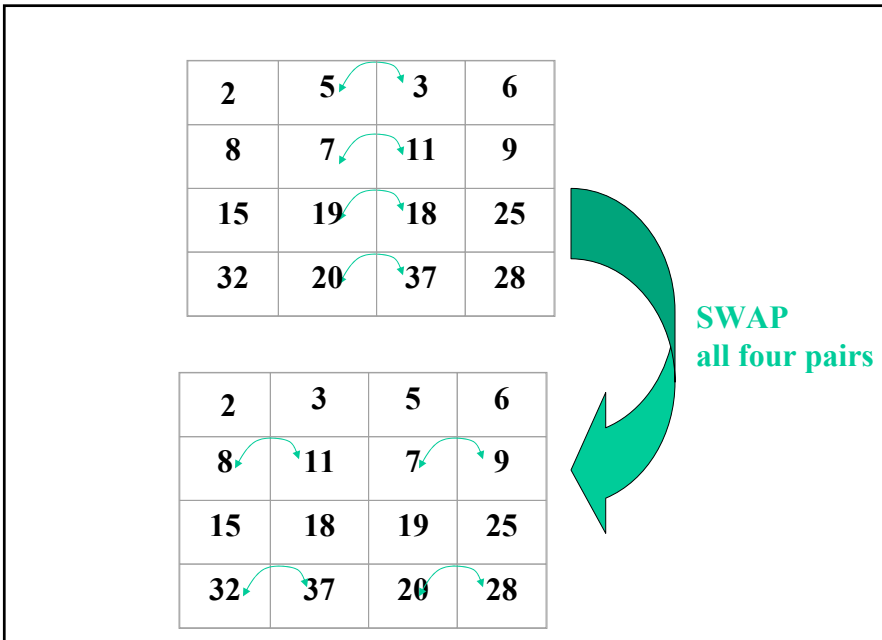
Total Run time T(m) = T(m/2)+2m+1= O(m)

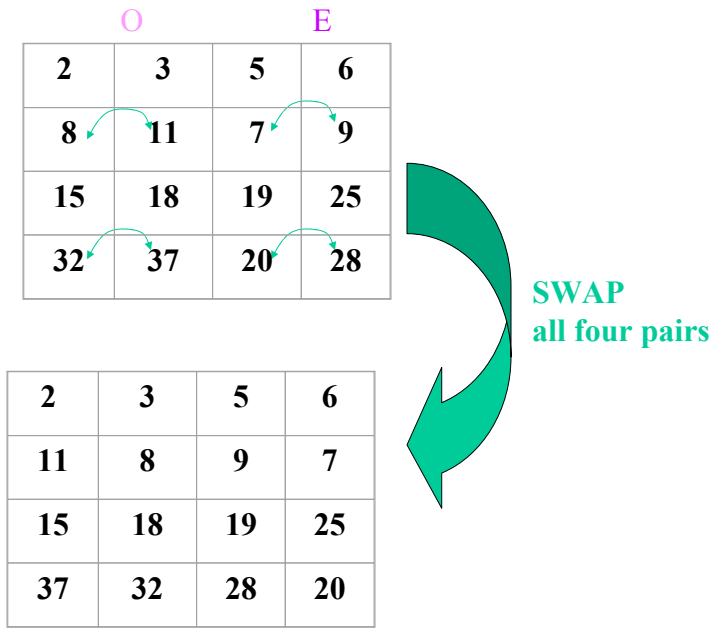
Note : T(m) means running time to merge 2 sorted lists, each with m elements

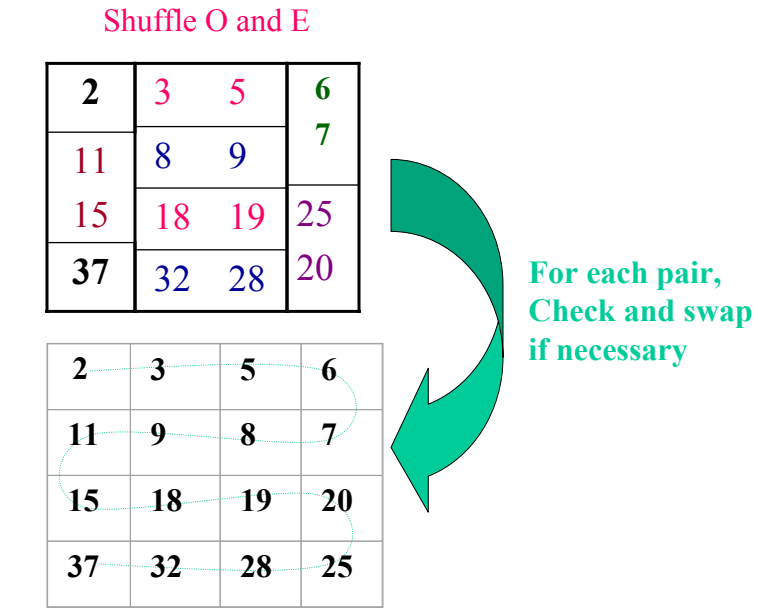
- Odd-Even Merge on a Mesh (O(p), use snakelike ordering)




O1
O2
E1
E2
5
2
6
3
8
7
11
9
15
19
18
28
20
32
25
37
O
E
SWAP
2
5
3
6
8
7
11
9
15
19
18
25
32
20
37
28
Sorting the
Odd group
and Even group


2
5
3
6
8
7
11
9
15
19
18
25
32
20
37
28
2
3
5
6
8
11
7
9
15
18
19
25
32
37
20
28
SWAP
all four pairs


O
E
2 3 5 6
8 11 7 9
15 18 19 25
32 37 20 28
SWAP
all four pairs
2 3 5 6
11 8 9 7
15 18 19 25
37 32 28 20


Shuffle O and E
2 3 5 6
11 8 9 7
15 18 19 25
37 32 28 20
For each pair,
Check and swap
if necessary
2 3 5 6
11 9 8 7
15 18 19 20
37 32 28 25

Sorting

- Odd-even transposition sort on linear array
  Assume each processor i has a number x_i

Algorithm Odd-even-transposition-sort
For i = 1 to p do
  If i is odd : compare keys at processors
    2j-1and 2j for all j
  If i is even : compare keys at processors
    2j and 2j+1 for all j

This can be done in O(p), (Skip the proof)

Example :

i = 1 : [5 4] [8 1] [2 6] [3 7]
i = 2 : 4 [5 1] [8 2] [6 3] 7
i = 3 : [4 1] [5 2] [8 3] [6 7]
i = 4 : 1 [4 2] [5 3] [8 6] 7
i = 5 : [1 2] [4 3] [5 6] [8 7]
i = 6 : 1 2 3 4 5 6 7 8

- Shearsort on a Mesh(use snakelike order)

  Assume each processor i has a number x_i

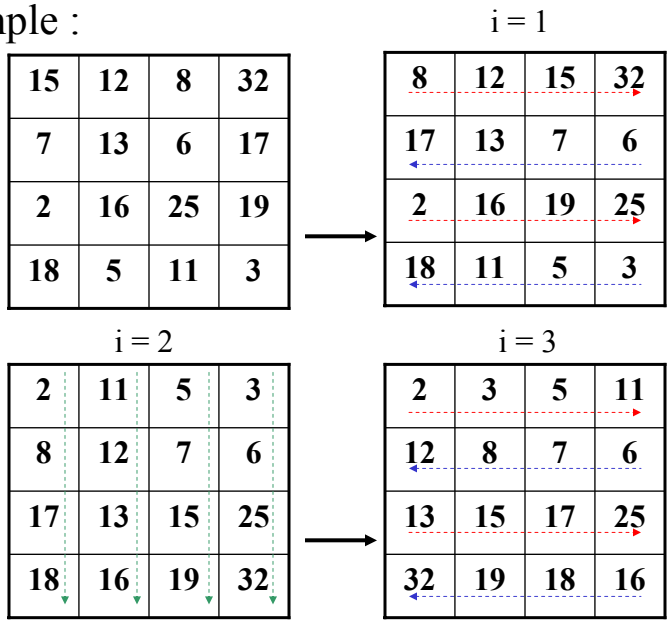
  Algorithm ShearSort

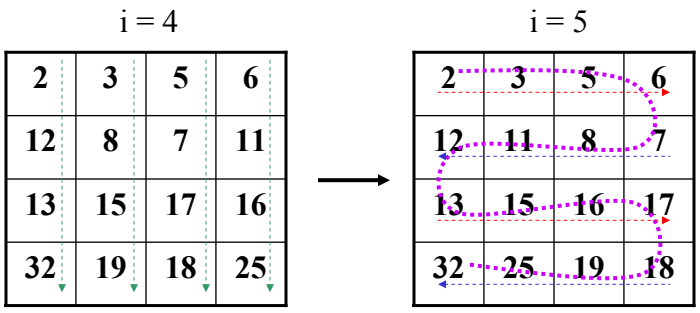
  For i = 1 to $\log(p^2) + 1$ do
  - If i is even : sort each columns in increasing order from top to bottom
  - If i is odd : sort each rows; alternate rows are sorted in reverse order.

  Use previous algorithm to sort p elements in O(p)

  This can be done in O(p log p), skip the proof!

Example :




i = 4
2 3 5 6
12 8 7 11
13 15 17 16
32 19 18 25
i = 5
2 3 5 6
12 11 8 7
13 15 16 17
32 25 19 18