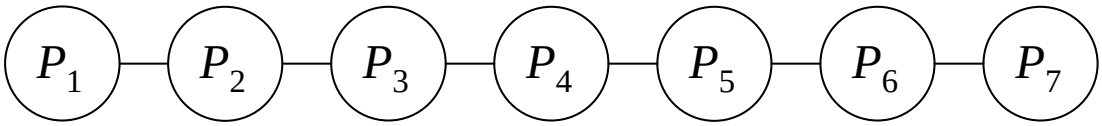
3
Linear Processor Arrays

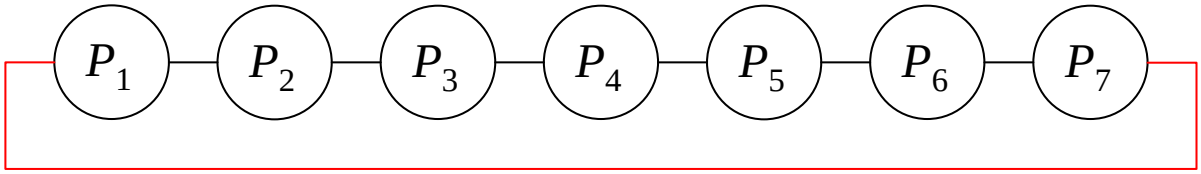
■ Linear Processor Array

n=7



(1) Diameter = $O(n)$ $n-1$
(2) Maximum degree = 2 interconnection network
(3) Link complexity = $O(n)$ $n-1$

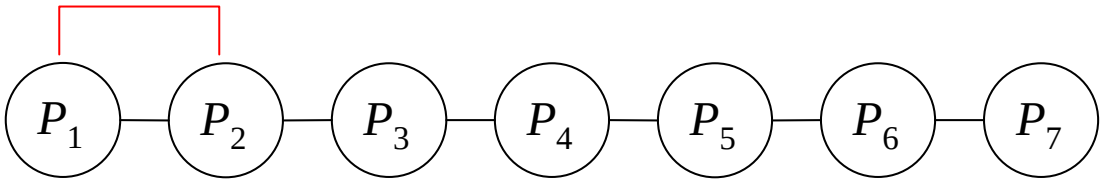
■ Ring (a linear processor array with end-around connection)



■ Problems

Problem LPA-1: Sorting (Odd-Even Transposition Sort)
Input: $A[1\ldots n]=\{7, 6, 2, 1, 3, 5, 4\}$
Output: $A[1\ldots n]=\{1, 2, 3, 4, 5, 6, 7\}$
Model: LPA of size n. Initially, a_i is stored in P_i.

Compare & Exchange

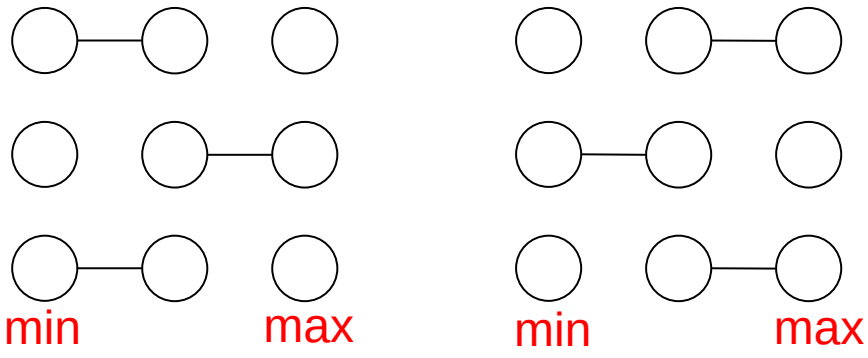


Step							
	7	6	2	1	3	5	4
1							
	6	7	1	2	3	5	4
2							
	6	1	7	2	3	4	5
3							
	1	6	2	7	3	4	5
4							
	1	2	6	3	7	4	5
5							
	1	2	3	6	4	7	5
6							
	1	2	3	4	6	5	7
7							
	1	2	3	4	5	6	7

Theorem: Odd-Even transposition sort produces a sorted sequence of *n* data after *n* steps.

Proof: (by induction)

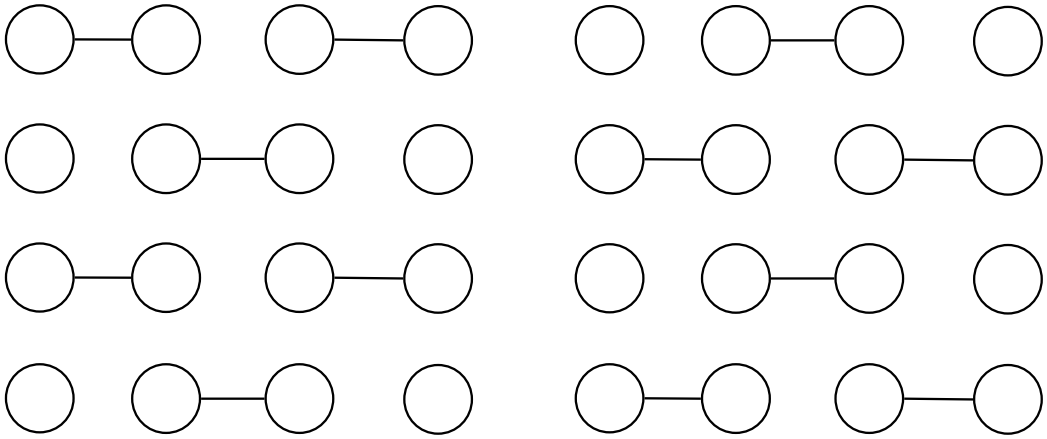
(1) *n*=3 case 1:odd-even case 2:even-odd



What is the sequential algorithm?
(Pipeline!)

(Assume $n = q \geq 3$ holds)

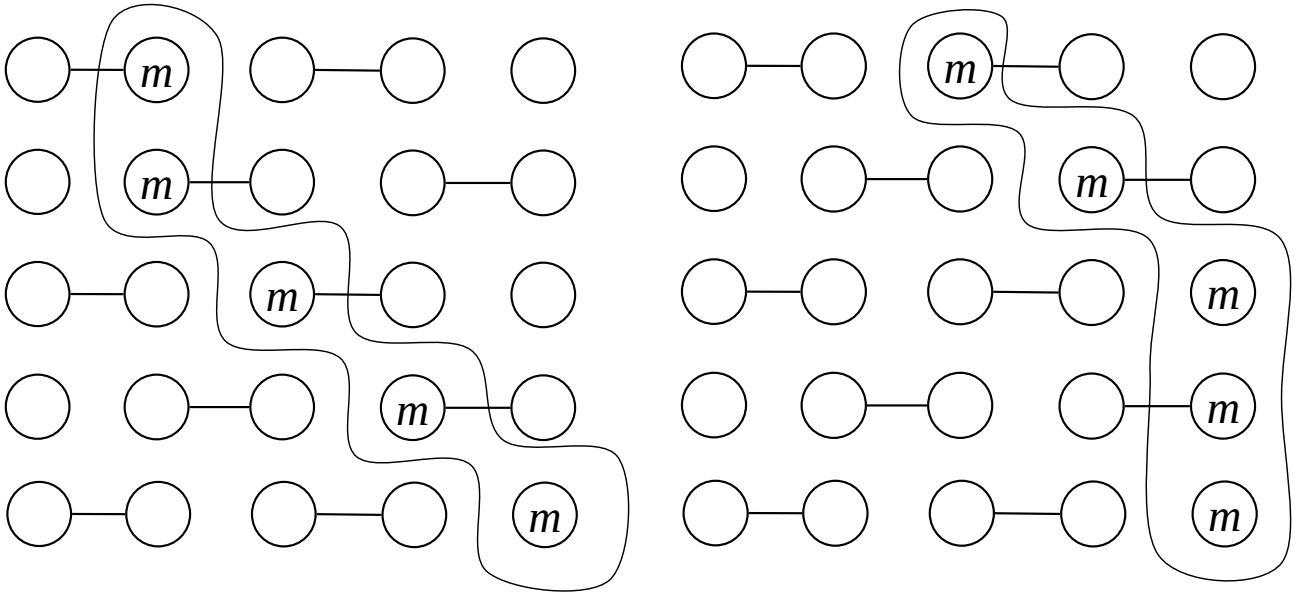
(2) n=4 case 1:odd-even case 2:even-odd



(Prove $n = q+1$ also holds)

(3) n=5 (Assume the maximum m is stored in P_k.)

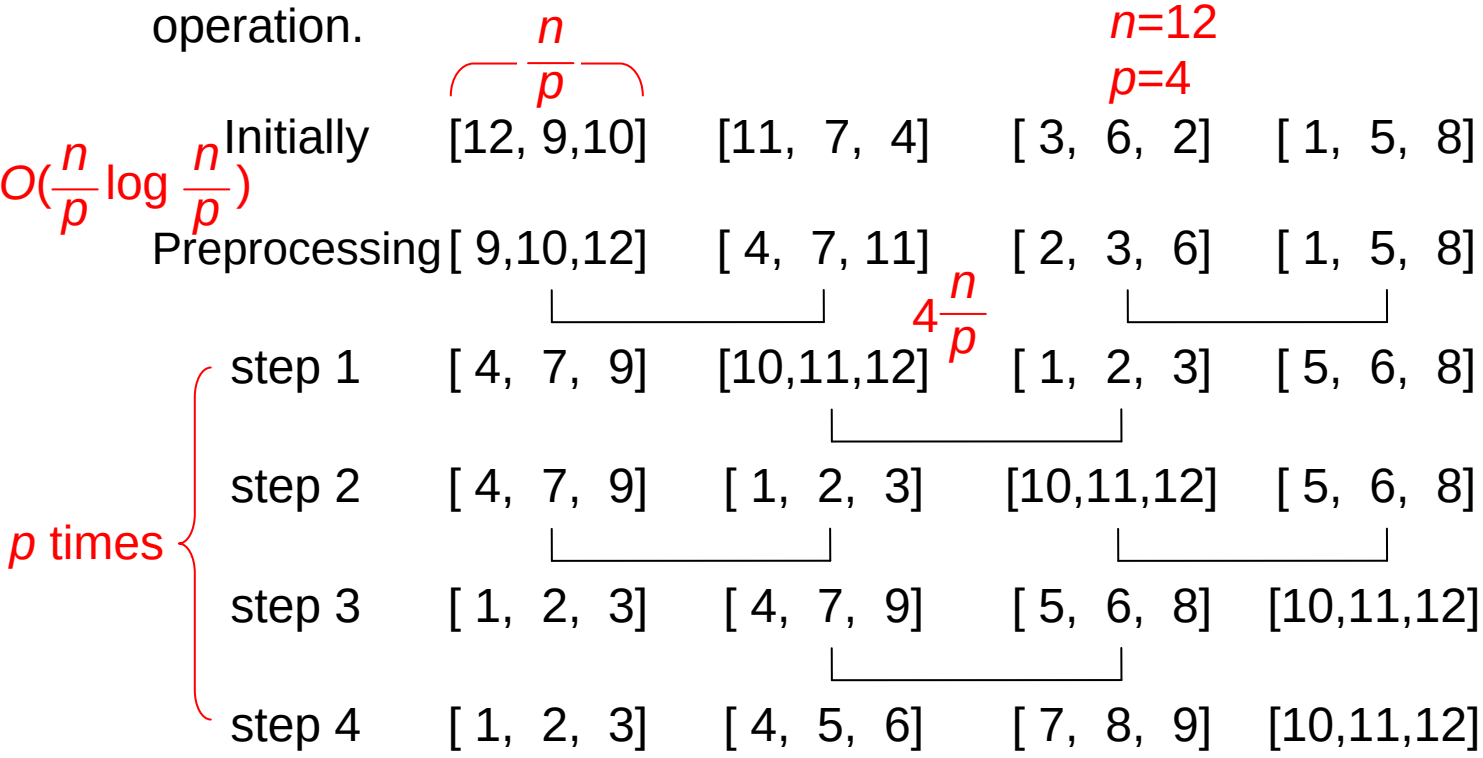
case 1.1:odd-even, k is even case 1.2:odd-even, k is odd



Problem LPA-2: Sorting (Merge-Splitting Sort)
Input: $A[1\ldots n]$={12, 9, 10, 11, 7, 4, 3, 6, 2, 1, 8, 5}
Output: $A[1\ldots n]$={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
Model: LPA of size p. Initially, $A[(i-1)*(n/p)+1 \ldots i*(n/p)]$ are stored in P_i.

(G. Baudet and D. Stevenson, "Optimal sorting algorithms for parallel computers," *IEEE Trans. Comput.* C-27 (1), 84-87.)

Merge-splitting sort is a generalization of odd-even transposition sort in which each processor holds a subsequence rather than a single data item, and the comparison-exchange operation is replaced with a merge-split operation.



* $T(n) = O((n/p)\log (n/p)) + p*O(n/p) = O((n \log n)/p + n)$

* For $p \le \log n$, the proposed algorithm is cost-optimal.

$$\text{cost} = (n\log n/p + n)\times p$$
$$= n\log n + pn$$

0/1
integer

Problem LPA-3: Integer Knapsack Problem

Input: C, $P[1\ldots n]$, and $W[1\ldots n]$ (capacity, profit, and weight)
(C, and w_i are integers)

Output: MAX $\sum_{i=1}^{n} x_i p_i$

such that (1) $\sum_{i=1}^{n} x_i w_i \le C$

(2) $x_i \ge 0$ are integers

Model: LPA of size n. Initially, p_i and w_i are stored in P_i, and C is stored in P_1.

(1) Sequential Approach (dynamic programming)

Let $f(g, k)$ be the maximal value of the objective function using only the first k items with capacity limitation g. Clearly, $f(C, n)$ is the answer, and

not to take k take k

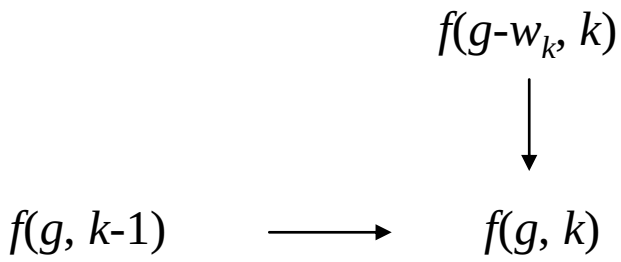
$f(g, k) = \text{Max}\{ f(g, k-1), p_k + f(g-w_k, k) \}$. (if $g \ge w_k$)

Iteration k=1 k=2 k=3 k=4 …

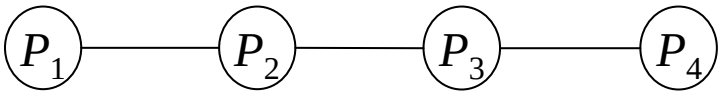
$f(0,1)$	$f(0,2)$	$f(0,3)$	$f(0,4)$	…
$f(1,1)$	$f(1,2)$	$f(1,3)$	$f(1,4)$	…
$f(2,1)$	$f(2,2)$	$f(2,3)$	$f(2,4)$	…
$f(3,1)$	$f(3,2)$	$f(3,3)$	$f(3,4)$	…
$f(4,1)$	$f(4,2)$	$f(4,3)$	$f(4,4)$	…
.	.	.	.	…
.	.	.	.	…

The algorithm

for k=1 **to** n
  for g=0 **to** C determine $f(g, k)$



(2) Parallel Approach (pipelining)



time	C, p_1w_1	p_2w_2	p_3w_3	p_4w_4	
0	C →				
1	$f(0, 1)$→	C →			
2	$f(1, 1)$→	$f(0, 2)$→	C →		
3	$f(2, 1)$→	$f(1, 2)$→	$f(0, 3)$→	C →	
4	$f(3, 1)$→	$f(2, 2)$→	$f(1, 3)$→	$f(0, 4)$→	…
5	$f(4, 1)$→	$f(3, 2)$→	$f(2, 3)$→	$f(1, 4)$→	…
6	.	.	.	.	…
7	.	.	.	.	…

* $T(n)=O(C+n-1)$

*The 0/1 knapsack problem: x_i=0 or 1.
For this problem, we have

$$f(g, k)=\mathbf{MAX}\{f(g, k-1), p_k+f(g-w_k, k-1)\}.$$

Problem LPA-4: Matrix Vector Product
Input: a matrix $A[1\ldots n, 1\ldots n]$ and a vector $X[1\ldots n]$.
Output: $Y[1\ldots n]=AX$
Model: LPA of size p. Let A and X be partitioned into p blocks as follows:

$A=(A_1, A_2, \ldots, A_p)$ and $X=(X_1, X_2, \ldots, X_p)$,

where each A_i is of size $n*r$ and each X_i is of size r. Initially, A_i and X_i are stored in P_i.

$$[A_1,A_2]\begin{bmatrix}X_1\\X_2\end{bmatrix} = A_1X_1 + A_2X_2$$

Example: (n=6, p=2) $\quad \frac{6}{2}=3$

$$Y = AX = \left[\begin{array}{ccc:ccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} \end{array}\right] \begin{array}{c} A_1 \\ \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] \\ \hdashline \\ A_2 \\ \left[\begin{array}{c} x_4 \\ x_5 \\ x_6 \end{array}\right] \end{array} \quad \begin{array}{c} n\times n \quad n\times 1 \\ X_1 \\ \\ X_2 \end{array}$$

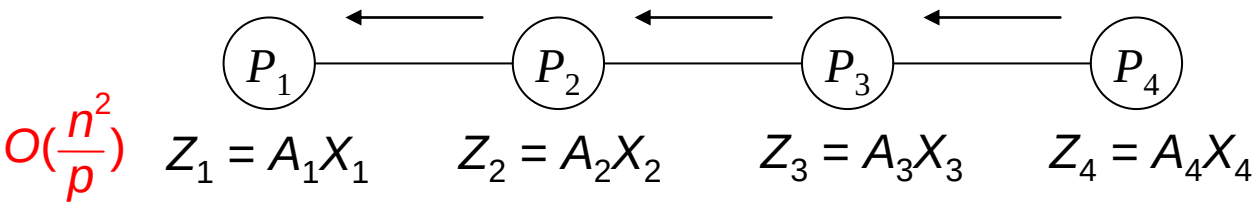
$$= \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \\ a_{5,1} & a_{5,2} & a_{5,3} \\ a_{6,1} & a_{6,2} & a_{6,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} a_{1,4} & a_{1,5} & a_{1,6} \\ a_{2,4} & a_{2,5} & a_{2,6} \\ a_{3,4} & a_{3,5} & a_{3,6} \\ a_{4,4} & a_{4,5} & a_{4,6} \\ a_{5,4} & a_{5,5} & a_{5,6} \\ a_{6,4} & a_{6,5} & a_{6,6} \end{bmatrix} \begin{bmatrix} x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

$$n\times\frac{n}{p} \qquad\qquad n\times\frac{n}{p}$$

$$= \quad A_1X_1 \quad + \quad A_2X_2$$

$$Z_1 \qquad\qquad Z_2$$

(p=4)



stage 1: Each processor P_i computes $Z_i=A_iX_i$. (Note that Z_i is a vector of order n.)

stage 2: **for** $i=p$ **to** 2

2.1 processor P_i sends Z_i to processor P_{i-1}, and then

2.2 processor P_{i-1} sets $Z_{i-1}=Z_{i-1}+Z_i$.

stage 3: Processor P_1 sets $Y=Z_1$.

partition

computation

* $T(n)=O(n^2/p) + (p-1)O(n) = O(n^2/p+np)$

communication

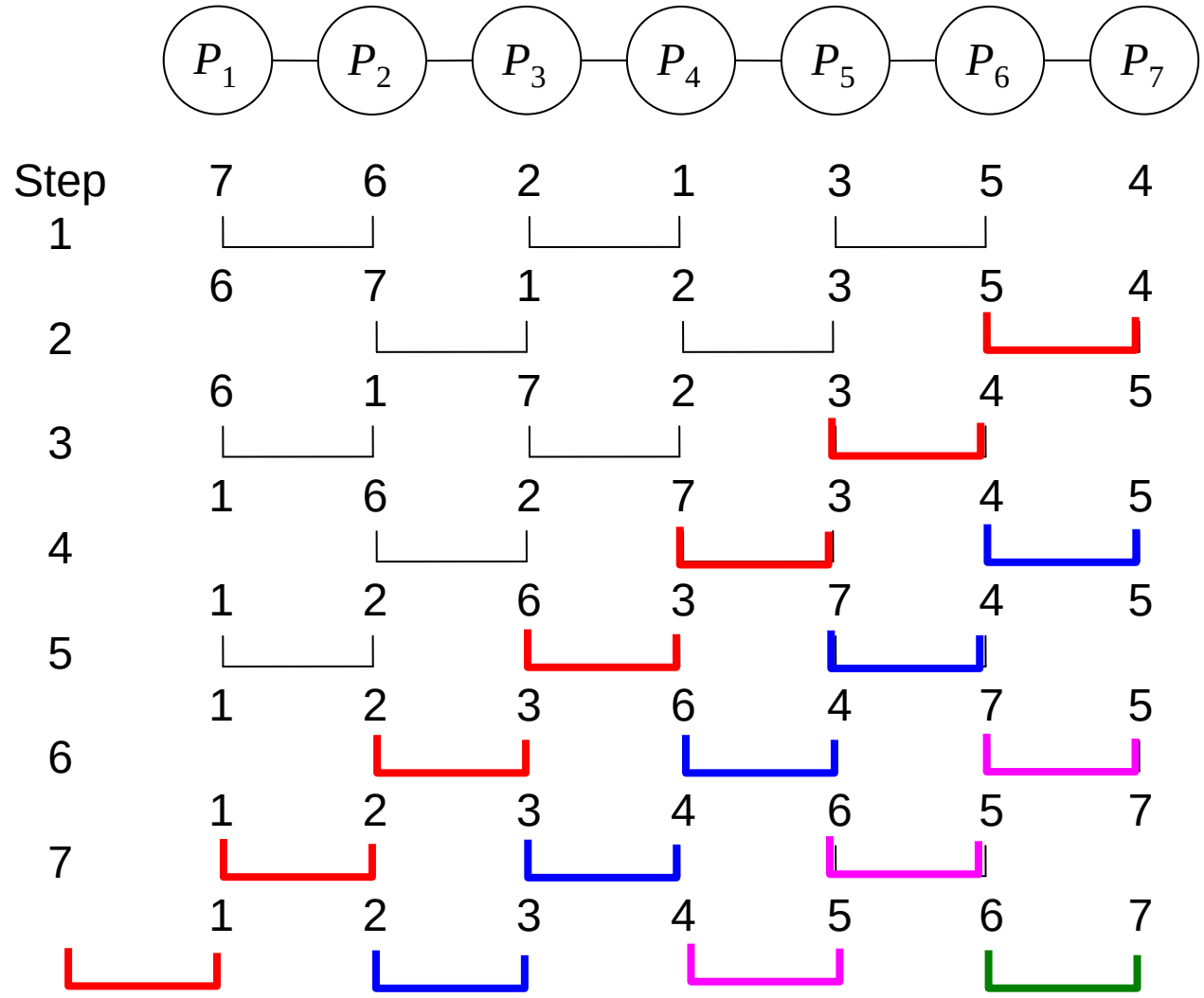
* For $p \le n^{1/2}$, the proposed algorithm is cost-optimal.

Remark: Given a parallel algorithm with computation time $C(n)$ and communication time $E(n)$. (Note that total running time $T(n)=C(n)+E(n)$.) If $C(n)<E(n)$, it is possible to reduce the cost and even the total running time, by using fewer processors. In this case, using fewer processors may increase $C(n)$, and possibly decrease $E(n)$; thus $C(n)$ and $E(n)$ are balanced.

* p	1	$n^{1/2}$	n
$C(n) = n*(n/p)$	n^2	$n^{3/2}$	n
$E(n) = p*n$	1	$n^{3/2}$	n^2
$T(n)$	n^2	$n^{3/2}$	n^2
cost	n^2	n^2	n^3

partition + pipeline ???

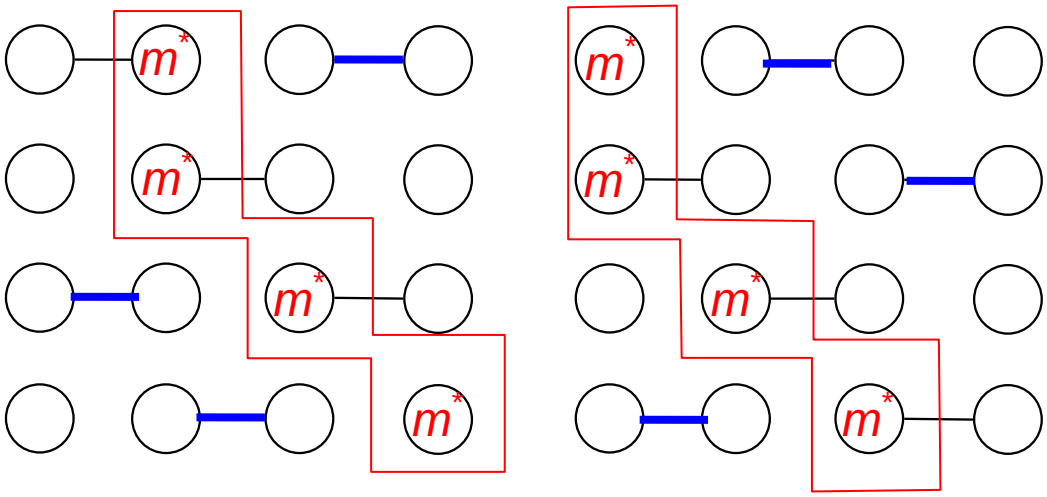
Appendix 1.


P_1 P_2 P_3 P_4 P_5 P_6 P_7
Step
7 6 2 1 3 5 4
1
6 7 1 2 3 5 4
2
6 1 7 2 3 4 5
3
1 6 2 7 3 4 5
4
1 2 6 3 7 4 5
5
1 2 3 6 4 7 5
6
1 2 3 4 6 5 7
7
1 2 3 4 5 6 7

Appendix 2.

(Assume $n = q \geq 3$ holds)

(2) n=4 case 1:odd-even case 2:even-odd



(Prove $n = q+1$ also holds)

(3) n=5 (Assume the maximum m is stored in P_k.)

case 1.1:odd-even, k is even case 1.2:odd-even, k is odd