Communication - N-node networks

- *Assume 1-port communication and virtual cut through.*
- *Message size m, startup time t_s (processor), hop time t_h (switch), and link latency t_w.*
- *Hypercube broadcast in $2(t_s \cdot logN + t_w \cdot m)$ steps.*
- *$\gamma(q)$ = largest integer j s.t. q is divisible by 2^j.*

Operation	Ring	2-D Mesh (wraparound, square)	Hypercube
One-to-all broadcast	$(t_s + t_w m)\log N$ $+t_h(N-1)$	$(t_s + t_w m)\log N$ $+2t_h(\sqrt{N}-1)$	$(t_s + t_w m)\log N$
All-to-all broadcast	$(t_s + t_w m)(N-1)$	$2t_s(\sqrt{N}-1) + t_w m(N-1)$	$t_s \log N + t_w m(N-1)$
One-to-all personalized	$(t_s + t_w m)(N-1)$	$2t_s(\sqrt{N}-1) + t_w m(N-1)$	$t_s \log N + t_w m(N-1)$
All-to-all personalized	$(t_s + t_w mp/2)(N-1)$	$(2t_s + t_w mp)(\sqrt{N}-1)$	$(t_s + t_w m)(N-1)$ $+(t_h/2)N\log N$
Circular q-shift	$(t_s + t_w m)\lfloor N/2 \rfloor$	$(t_s + t_w m)(2\lfloor \sqrt{N}/2 \rfloor + 1)$	$t_s + t_w m$ $+t_h(\log N - \gamma(q))$

Sorting Models

- *Comparison-based sorting*. Use repeated compare & swap operations. The lower bound for sorting N elements serially is $\Omega(NlogN)$.
- *Noncomparison-based sorting*. Use extra properties of the elements, related to their binary representation or distribution (e.g. rank sort). The lower bound for sorting N elements serially is $\Omega(N)$.

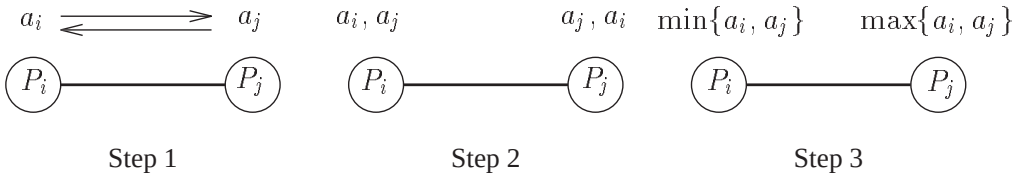
If you know all numbers are disjoint and distributed between 1 *and* N*, how would you sort them?*

We now consider parallel comparison-based sorting.

Notice that, $O(logN)$ *time complexity is difficult to achieve*. Linear speedup only from parallelizing $O(N^2)$ serial sorting algorithms, such as bubblesort!

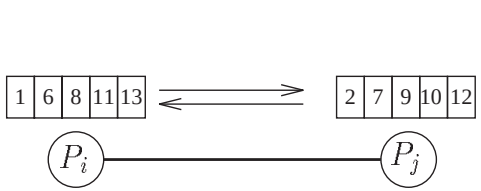
Compare & Swap Sorting Model

- *Sorting based on primitive swaps.* First, data elements swapped between processors P_i and P_j. Then, processor P_i retains the $\min\{a_i, a_j\}$, while processor P_j retains the $\max\{a_i, a_j\}$, where $P_i \leq P_j$.
- *Sorting is according to an implicit processor order*, such as decimal order, gray-code, and snake-like ordering in 2-dimensional mesh.

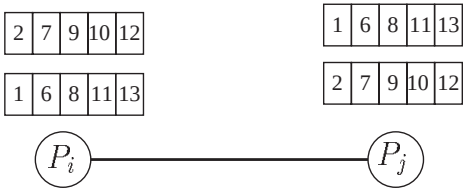


Sorting Model - Compare & Split

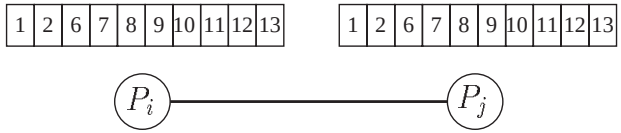
- *Simple extension of primitive swaps to blocks.* First, data blocks are swapped between processors P_i and P_j. Then, these blocks are merged and each processor retains its appropriate half of the merged block.



Step 1



Step 2



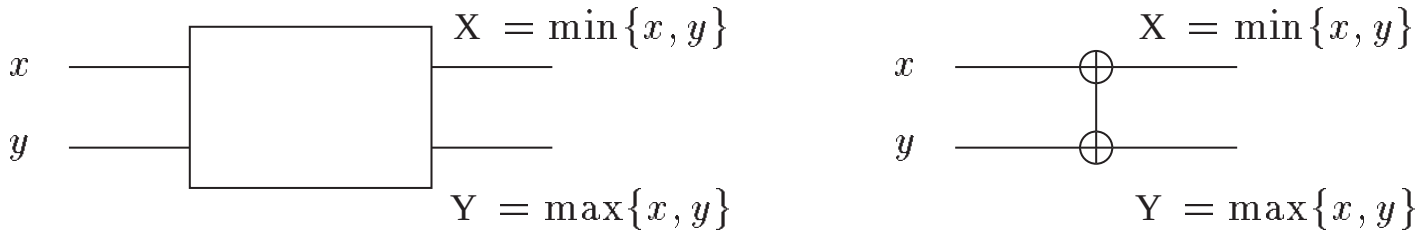
Step 3



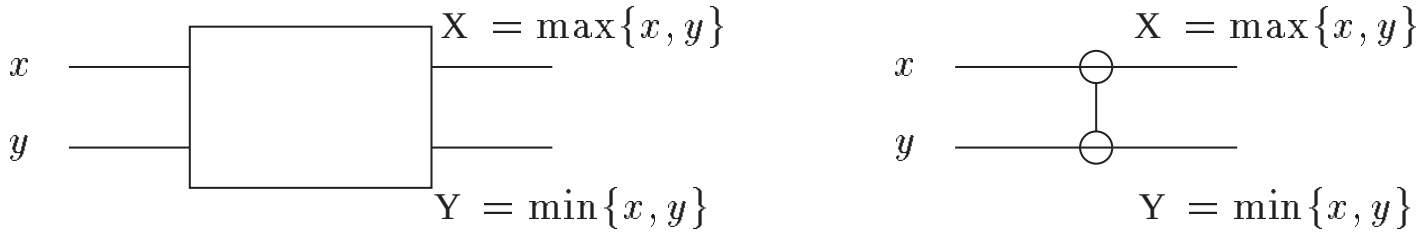
Step 4

Sorting Networks - Comparators

- *Increasing and decreasing comparators.*



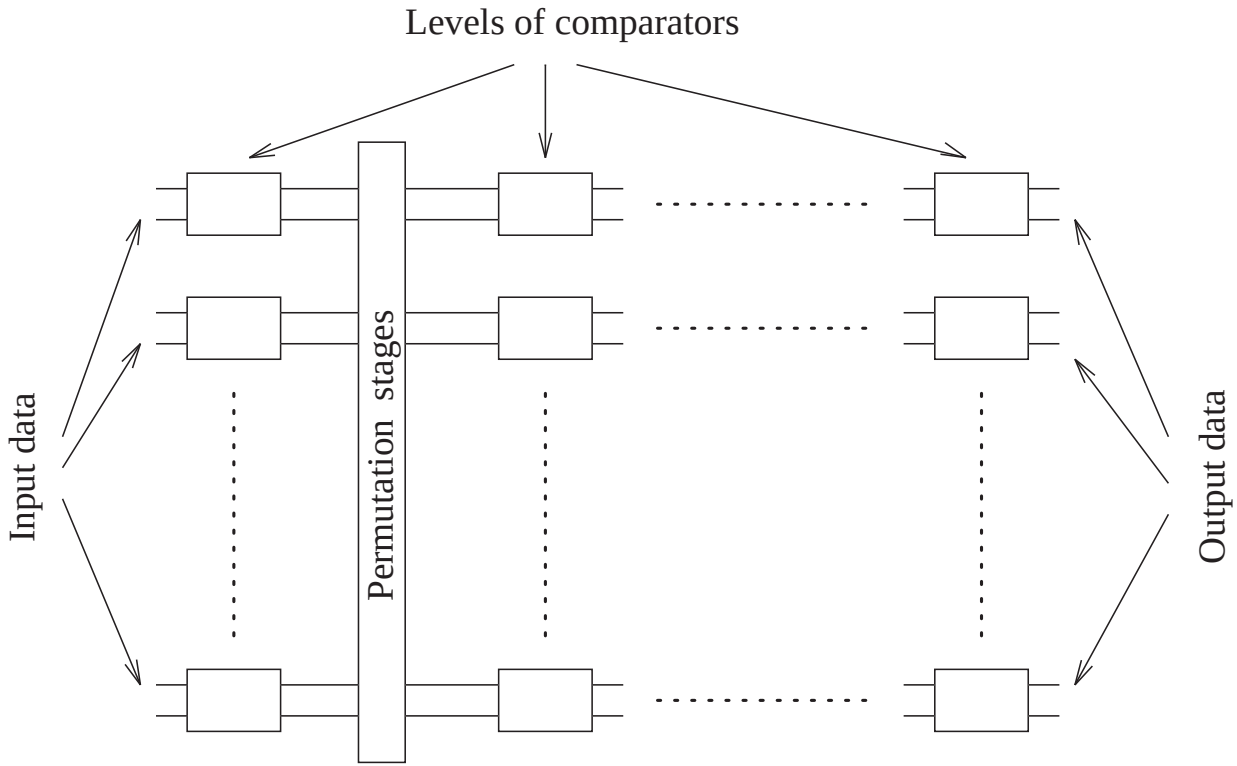
(a)



(b)

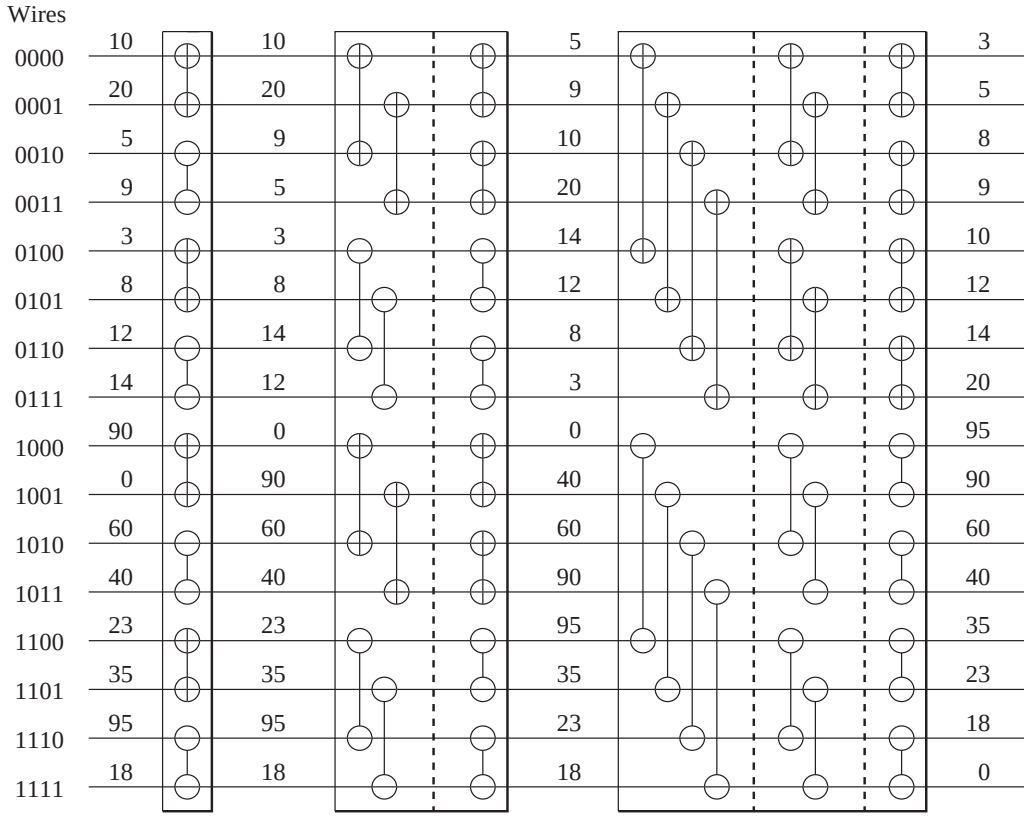
Sorting Networks - Comparators

- *Sorting (multistage) networks with comparators.*



From Random to Bitonic Sequences

- *A bitonic sequence has at most one local maximum and one local minimum.* The endpoints wrap-around.
- *Generating a bitonic sequence of* 16 *elements.*



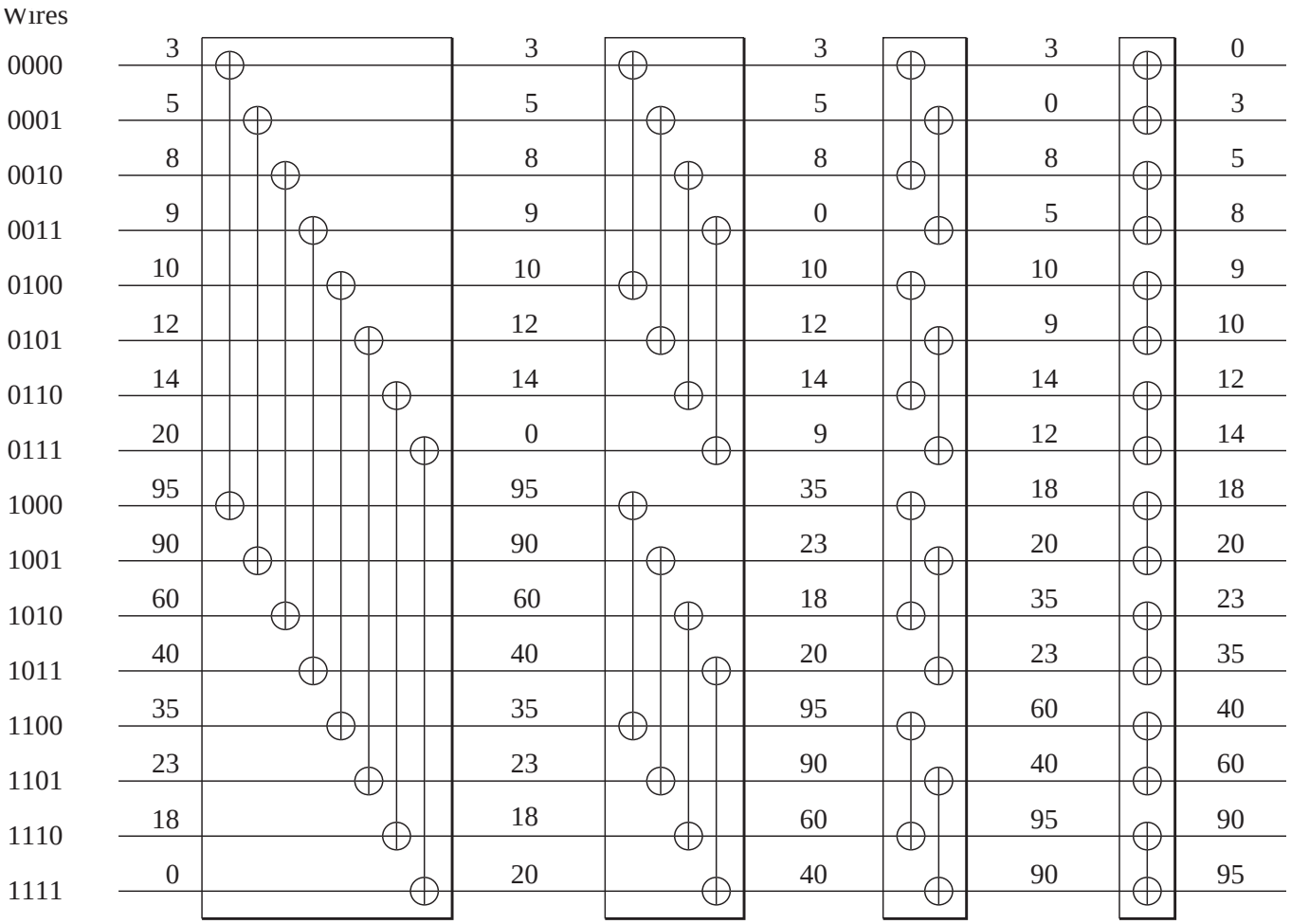
Sorting of Bitonic Sequences

- *Sorting algorithm for bitonic sequence of* 16 *elements*. The list is divided into half. Elements at similar positions in each sublist compare and swap values. We assume a bitonic sequence of length $N = 2^n$ (padding).

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

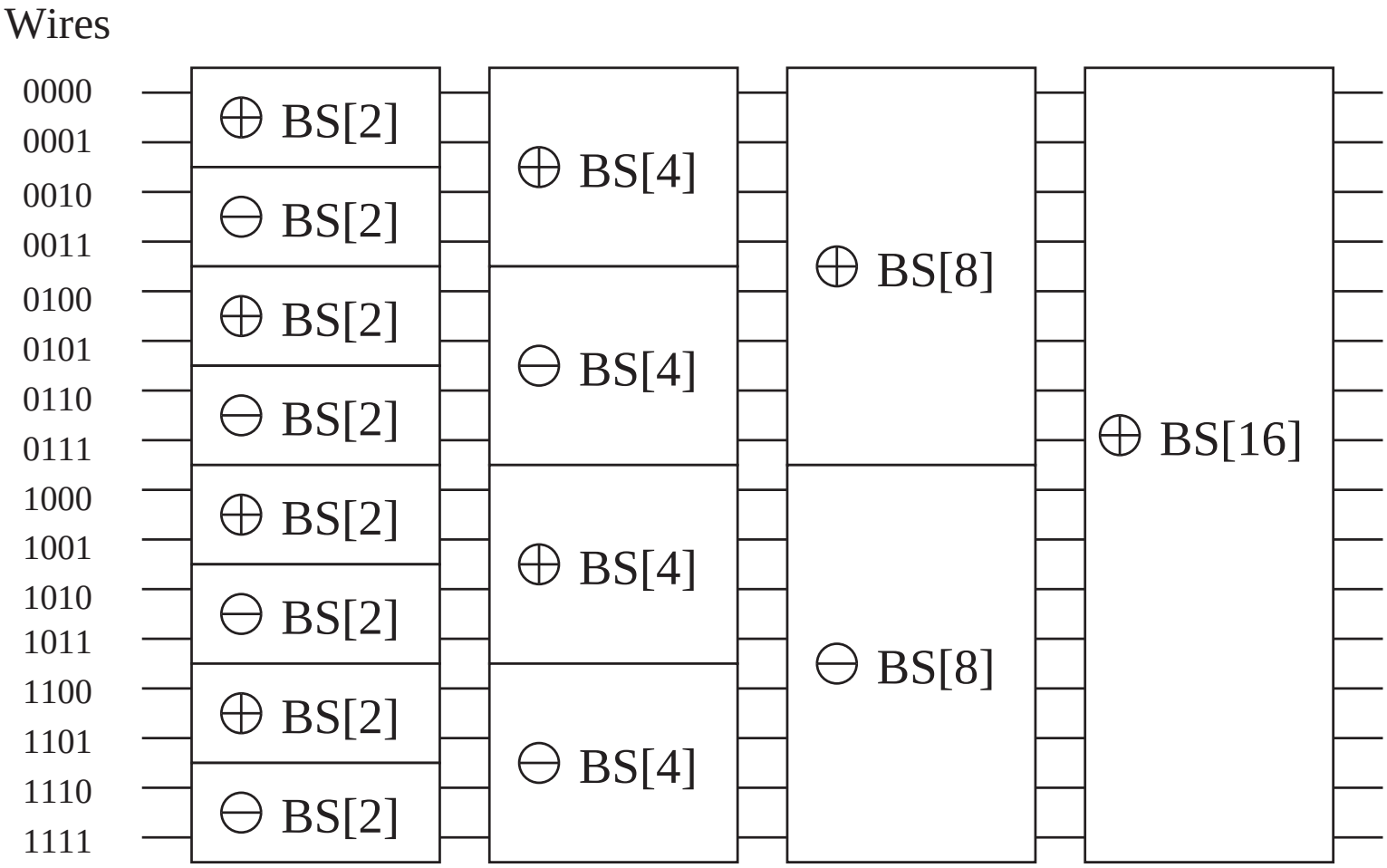
Sorting of Bitonic Sequences

- *Sorting network for bitonic sequence of* 16 *elements.*



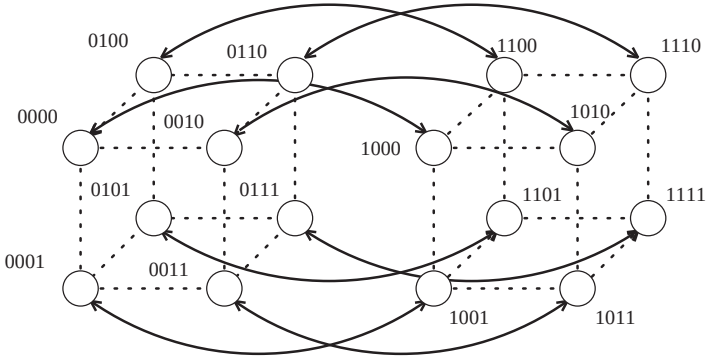
Bitonic Sorting Network

- *Sorting network for* $p = 16$. It is based on sorting bitonic sequences of two, four and eight elements.

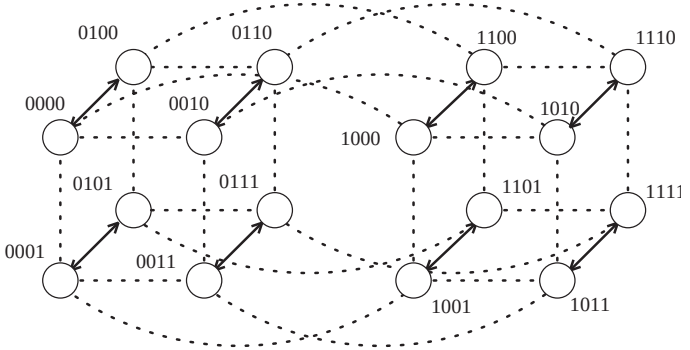


Bitonic Sequences on n-cube

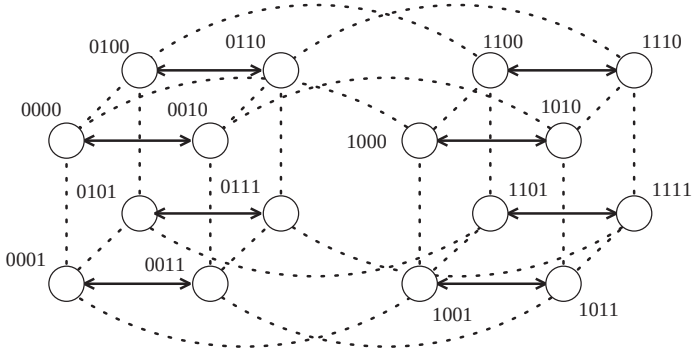
- *Sorting a* 16 *element bitonic sequence.* The mapping of wires from any stage of the sorting network to edges of the hypercube is one to one.



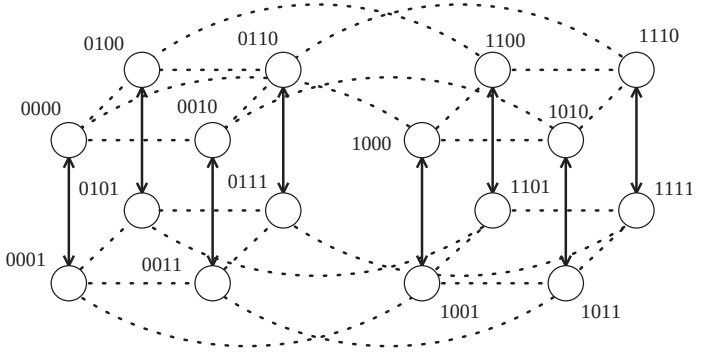
Step 1



Step 2



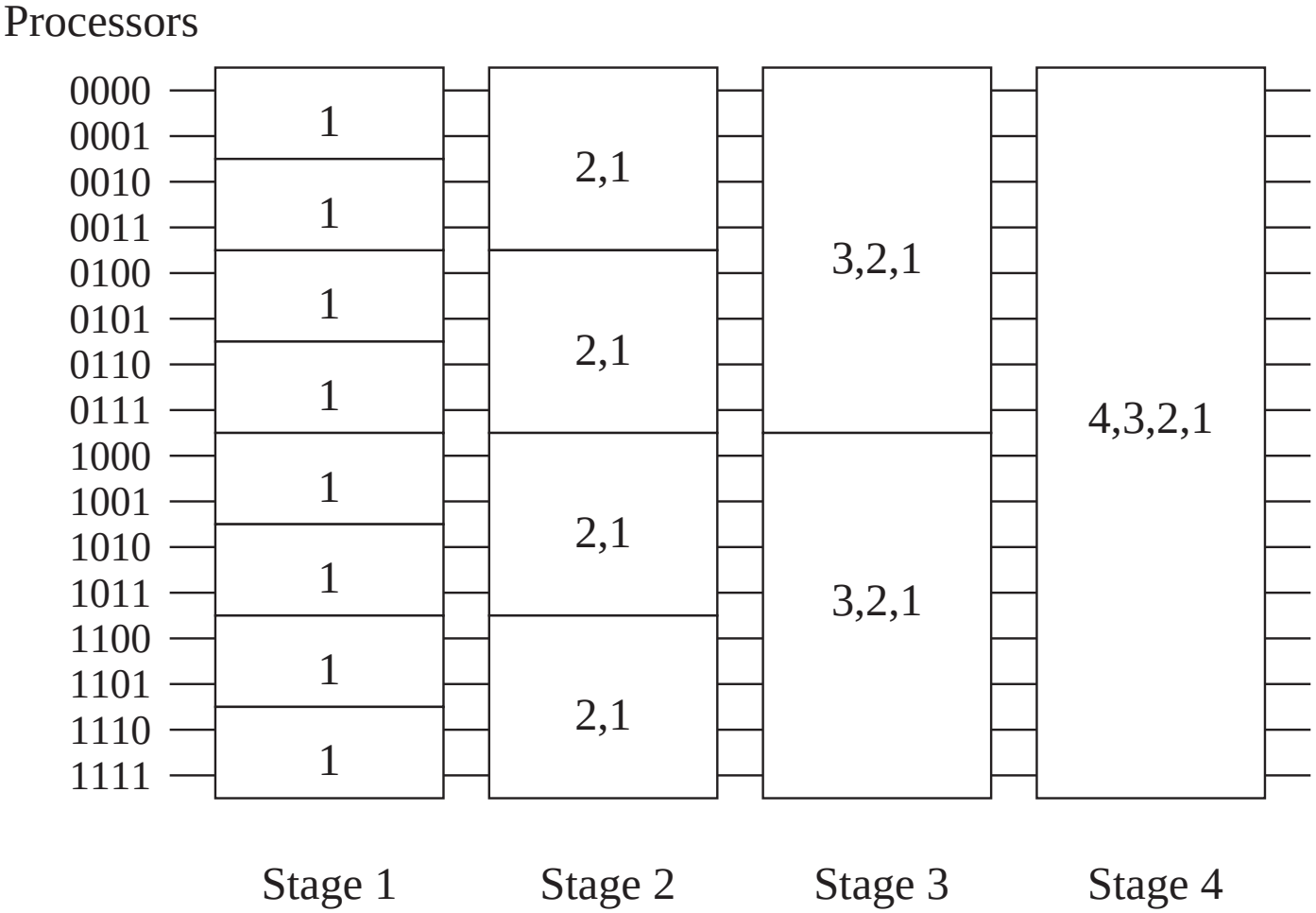
Step 3



Step 4

Bitonic Sorting on n-cube

- *Processors swap data along the dimensions shown.*



The recurrence relation describing the time complexity is: $T(N) = T(N/2) + O(logN) = O(log^2 N)$.

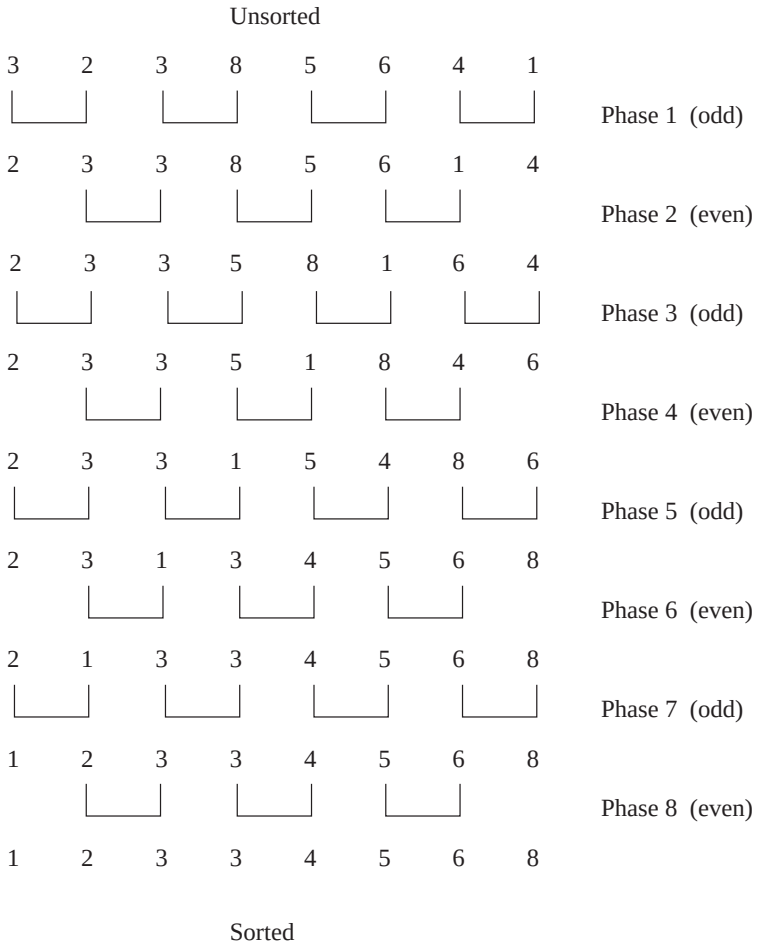
Comparisons of Bitonic Sorting

- *Bitonic sorting of p elements on N-node networks.*

Architecture	Maximum Number of Processors for $E = \Theta(1)$	Corresponding Parallel Run Time	Isoefficiency Function
Hypercube	$\sqrt{\log p}$	$\sqrt{\log p}$	$N^{\log N}$ ^{2} N
Mesh	^{2}		$\sqrt{N}$ $\sqrt{N}$
Ring			N N

Odd-Even Transposition Sorting

- *Odd-even transposition sorting for* $p = 16$. The p elements are sorted in p phases, each phase requiring $p/2$ compare & swaps.

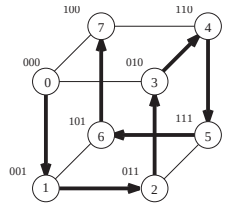


For $p = N$, $T(N) = O(N)$.
For $p > N$, $T(N) = (p/N) \cdot log(p/N) + 2N \cdot (p/N)$.

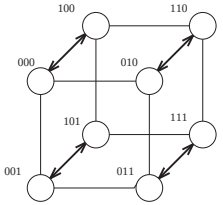
- *If* $N = O(logp)$ $T(N) = O(p) \Rightarrow$ *linear speedup*!
- *Iisoefficiency* $W = k(N \cdot T(N) - T(1)) = O(N \cdot 2^N)$.

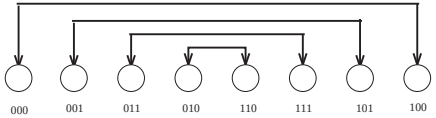
Shellsort on n-cube

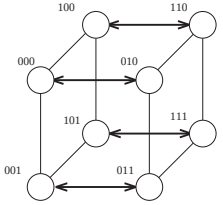
- *Example: first phase of shellsort on a 3-cube.*

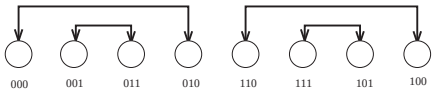


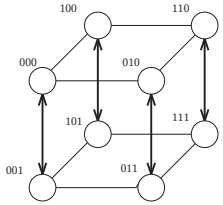
A ring embedded on a hypercube specifies the global ordering of the sorted elements.

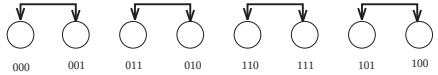






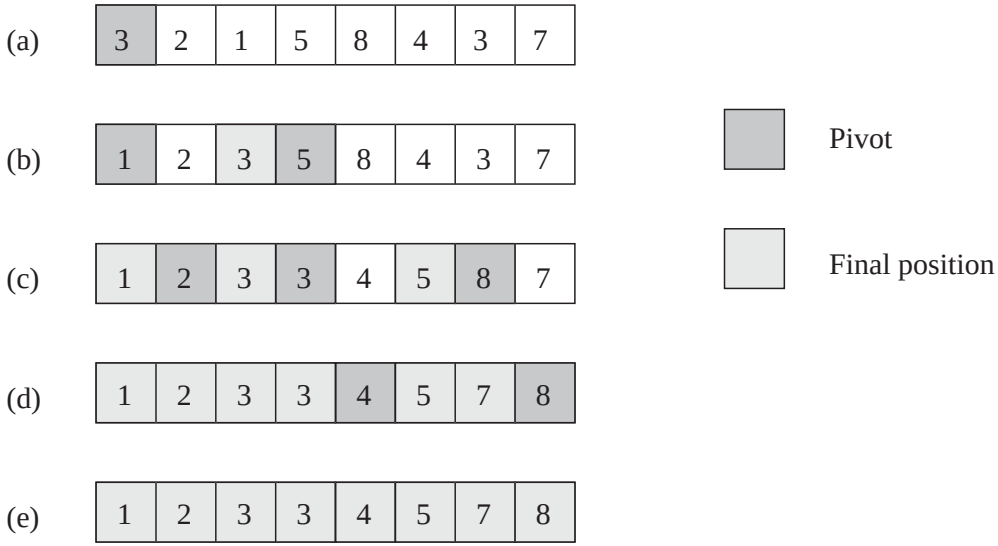




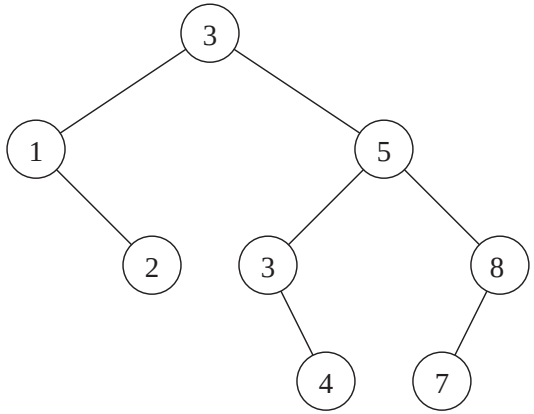


Quicksort

- *Quicksort example on an* 8 *element sequence.* Pivot selection, partitioning using pivot, and recursion.

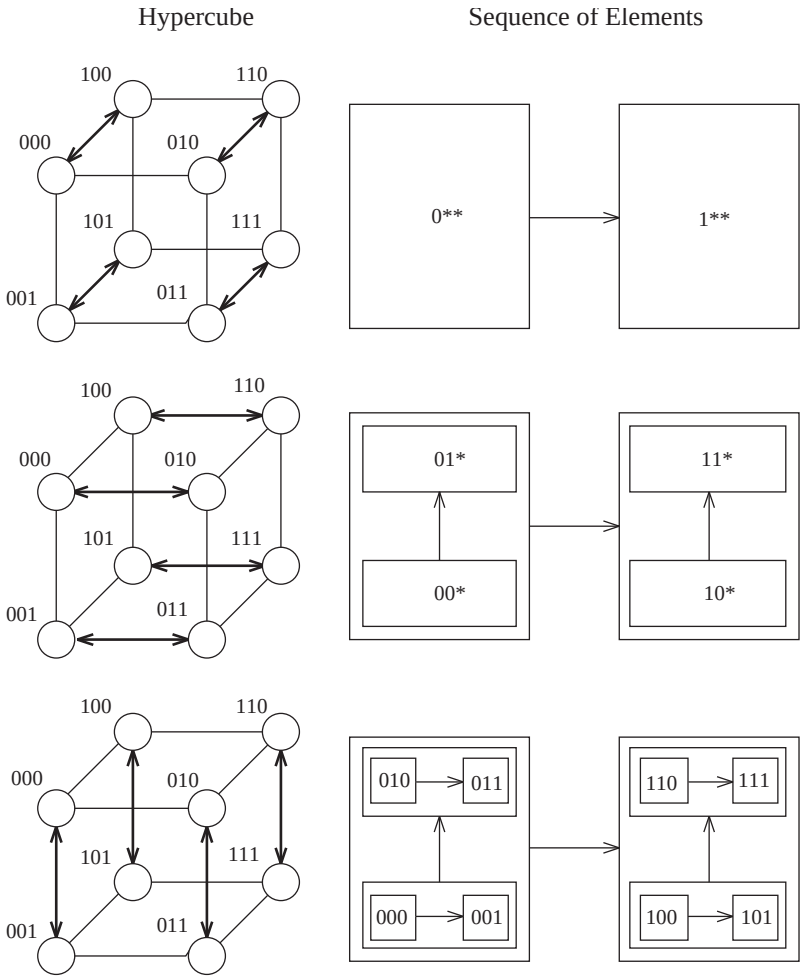


- *Pivot selection is important*!



Quicsort on n-cube

- *Parallel quicksort on* 3*-cube.* Local sort, broadcast and merge.



(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

Comparisons of Quicksort

- *Quicksort of p elements on N processor networks.*

Architecture	Maximum Number of Processors for $E = \Theta(1)$	Corresponding Parallel Run Time	Isoefficiency Function
CRCW PRAM	p	$\Theta(\log p)$	$\Theta(p \log p)$
Hypercube			
best case	$\Theta(p/\log p)$	$\Theta(\log^2 p)$	$\Theta(N \log^2 N)$
worst case	$\Theta(\log p)$	$\Theta(p)$	$\Theta(N 2^N)$
Mesh	$O(((\log p)/(\log \log p))^2)$		$\Theta(N\sqrt{N}\sqrt{N} \log N)$

Scalability of Parallel Systems

- *Speedup* $S = \frac{T_1}{T_N}$. Best serial vs. parallel algorithm.
- *Efficiency* $E = \frac{S}{N} \leq 1$.
- *Overhead function* $T_o = N \cdot T_N - T_1$. For a cost-optimal solution $T_o = O(T_1)$.
- *Isoefficiency (W)* is the rate at which p must increase as a function of N to keep the efficiency E constant.

Scalable system: when N increases, we can maintain the same efficiency by increasing the problem size p.

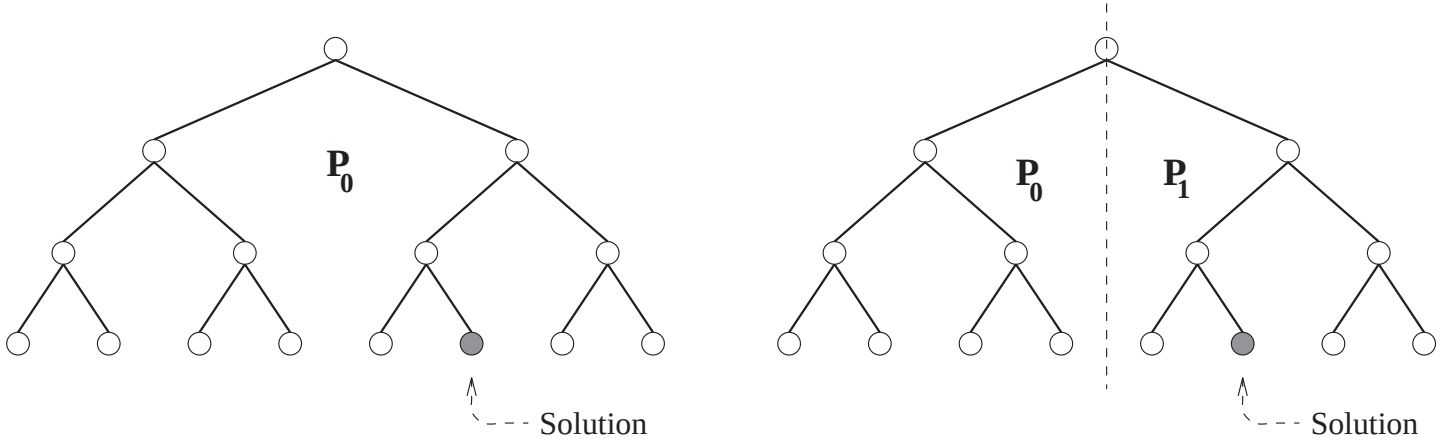
1. *For unscalable systems W does not exist!*
2. *System scalability is higher when W is smaller.*
3. *W used for performance prediction in larger systems.*

Superlinear Speedup - $E > 1$

Caused by sequential program nonoptimality.

Examples:

- *Single processor cache/memory insufficient ⇒ extra page-faults!*
- *Superlinear speedup in depth first search!*



(a) DFS with one processor (b) DFS with two processors

Parallel Algorithms and Embeddings

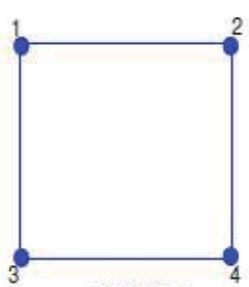
- *Design New Algorithms.*
- *Port Existing Algorithms.* This method is known as graph embedding, simulation, or mapping.

Embedding function $f: G = (V, E) \Rightarrow G' = (V', E')$.

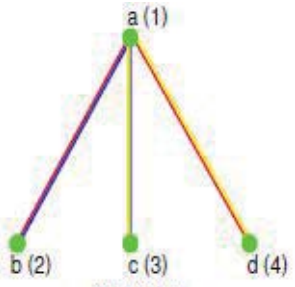
Measures for quality of graph embedding.

1. *Dilation* $d_f = max_{v_1 \neq v_2} \frac{d(f(v_1), f(v_2))}{d(v_1, v_2)}$ *is the length of the longest path that any edge of* G *is mapped.* It is related to the communication overhead in simulating graph G by graph G'.
2. *Node expansion* $c_f = \frac{|V'|}{|V|} \geq 1$.
3. *Edge congestion is the maximum number of edges of* G *mapped on a single edge of* G'.

Embeddings onto n-cube



GRAPH G



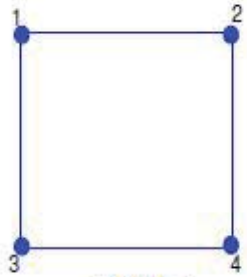
GRAPH H

EMBEDDING FUNCTION 1-> a, 2-> b, 3-> c, 4-> d

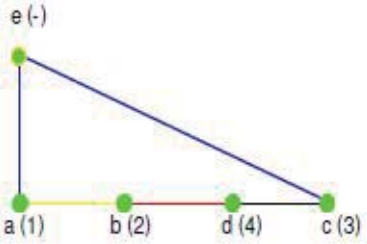
DILATION = 2. Edges 3-4 and 2-4 of G mapped to two steps in H.

EXPANSION = 4/4 = 1.

CONGESTION = 2. Two paths cross over links a-b, a-c, a-d.



GRAPH G



GRAPH H

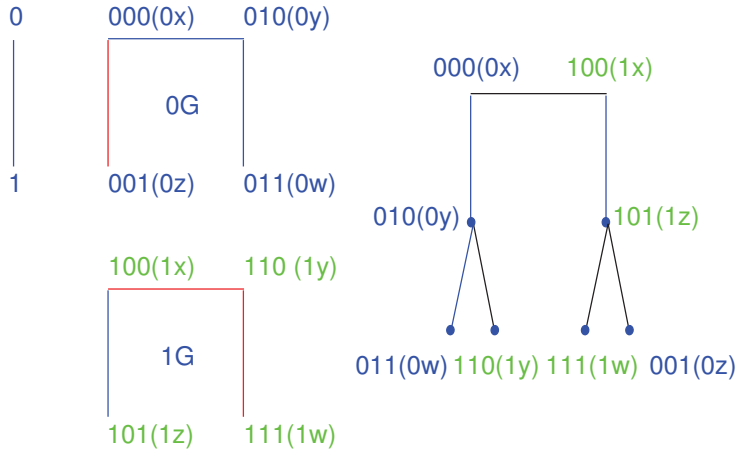
EMBEDDING FUNCTION 1-> a, 2-> b, 3-> c, 4-> d

DILATION = 2. Edge 1-3 of G mapped to two steps in H.

EXPANSION = 5/4 = 1.25.

CONGESTION = 1. Two paths never cross over any link.

Tree Embeddings onto n-cube



1. *Unit dilation and expansion embedding of complete binary tree with $2^n - 1$ nodes onto n-cube* **not possible** *(parity). Possible for base-4 cube with 2^n nodes.*
2. *Unit dilation and expansion two embedding of the above complete binary tree onto the $(n+1)$-cube.*
3. *Unit dilation and expansion embedding of double rooted binary tree with 2^n nodes onto the n-cube.*

Asymptotically good embeddings for arbitrary binary trees, mesh of trees, trees of meshes, pyramids, etc.

Embeddings onto n-cube

- *Many embedding problems are NP-complete.*

1. *Optimal embedding of a graph onto another.*
2. *Embedding of a graph onto the n-cube.*
3. *General tree embeddings onto the n-cube.*

- *Unit dilation and expansion embeddings for symmetric graphs.*

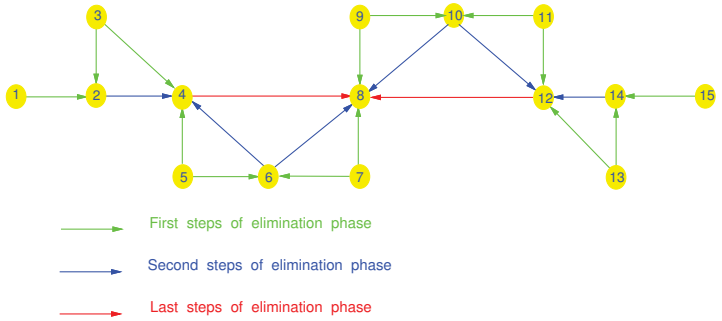
1. *Linear arrays of size* 2^k, *when* $k \leq n$. The smallest n-cube onto which a linear array of arbitrary size s can be mapped is $n = \lceil \log s \rceil$.

2. *Rings of size* 2^k, *when* $k \leq n$.

3. *Rings of even size* s, *with* $4 \leq s \leq 2^n$. Let $m = s/2$, $n = \lceil logs \rceil$ and $G_{n-1}[m]$ be the first m elements of G_{n-1}. Then form: $\{0G_{n-1}[m],\ 1G^R_{n-1}[m]\}$.

4. *k-dimensional grids of size* $2^{s_1} \times 2^{s_2} \times \ldots \times 2^{s_k}$, *if* $s_1 + s_2 \ldots + s_k \leq n$.

Embeddings onto n-cube

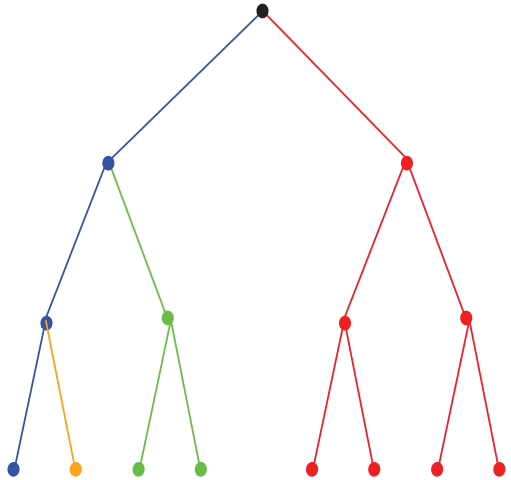


Binary hypercube does not admit triangles!

In any mapping of cyclic reduction on binary n-cube, at least $n/2$ steps involve communication between processors at distance two or more.

Binary-reflected Gray code on n-cube is optimal.

Binary-Reflected Gray Codes



Property 1. $G_1 = \{0, 1\}$.

Property 2. $G_i = \{0G_{i-1}, 1G_{i-1}^R\}, \quad 1 \leq i \leq n$.

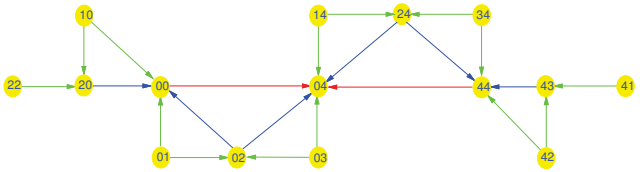
Examples:

- $G_2 = \{0G_1, 1G_1^R\} = \{00, 01, 11, 10\}$.
- $G_3 = \{0G_2, 1G_2^R\} = \{000, 001, 011, 010, 110, 111, 101, 100\}$.

Embeddings onto base-b n-cube

Theorem 6.1 *There does not exist unit dilation embedding on the base-b cube with* 2^n *nodes, unless* $b \geq 2^{n-2} + 1$.

In our example, $M = 15 = 2^4 - 1,\ n = 4$.



Base-5 cube with 25 *nodes and* 100 *edges admits unit dilation. But, a fully-connected graph with* 15 *nodes contains only* 105 *edges!*

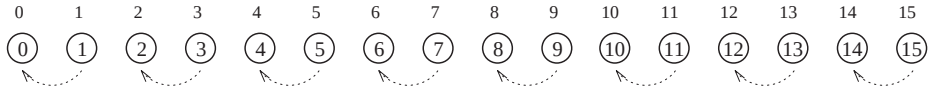
Topological Properties

Static interconnection networks with N nodes.

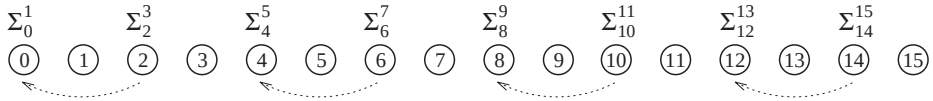
Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$\mathbf{N}^2/4$	$\mathbf{N}-1$	$\mathbf{N}(\mathbf{N}-1)/2$
Star	2	1	1	$\mathbf{N}-1$
Complete binary tree	$2\log((\mathbf{N}+1)/2)$	1	1	$\mathbf{N}-1$
Linear array	$\mathbf{N}-1$	1	1	$\mathbf{N}-1$
Ring	$\lfloor \mathbf{N}/2 \rfloor$	2	2	$\mathbf{N}$
2-D mesh without wraparound	$2(\sqrt{\mathbf{N}}-1)$	$\sqrt{\mathbf{N}}$	2	$2(\mathbf{N}-\sqrt{\mathbf{N}})$
2-D wraparound mesh	$2\lfloor \sqrt{\mathbf{N}}/2 \rfloor$	$2\sqrt{\mathbf{N}}$	4	$2\mathbf{N}$
Hypercube	$\log \mathbf{N}$	$\mathbf{N}/2$	$\log \mathbf{N}$	$(\mathbf{N}\log \mathbf{N})/2$
Wraparound k-ary d-cube	$d\lfloor k/2 \rfloor$	$2k^{d-1}$	$2d$	$d\mathbf{N}$

Parallel Sum ($N = p$)

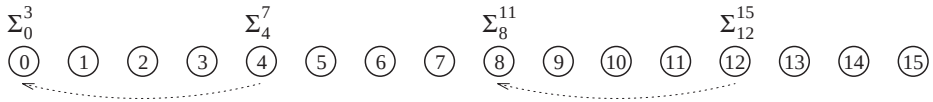
- *Example:* $p = 16$ *numbers on* $N = 16$ *processors.*



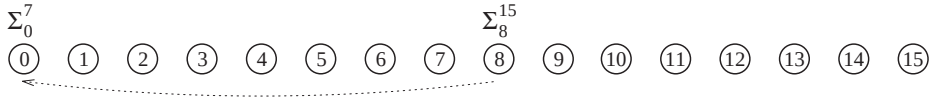
(a) Initial data distribution and the first communication step



(b) Second communication step



(c) Third communication step



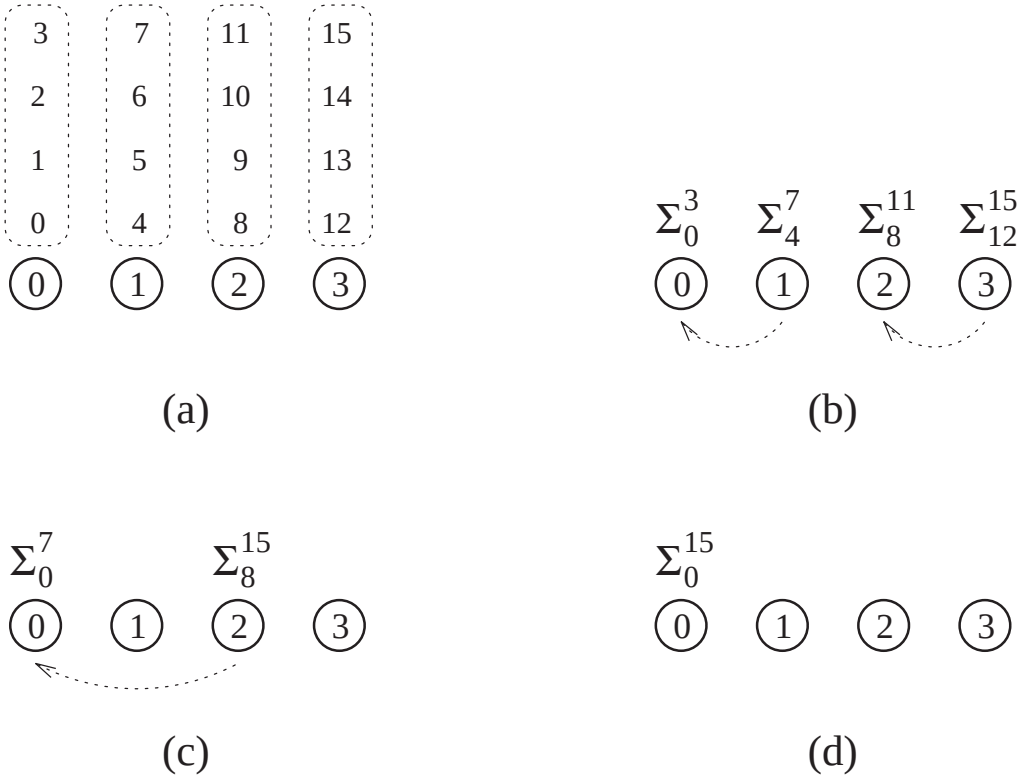
(d) Fourth communication step



(e) Accumulation of the sum at processor 0 after the final communication

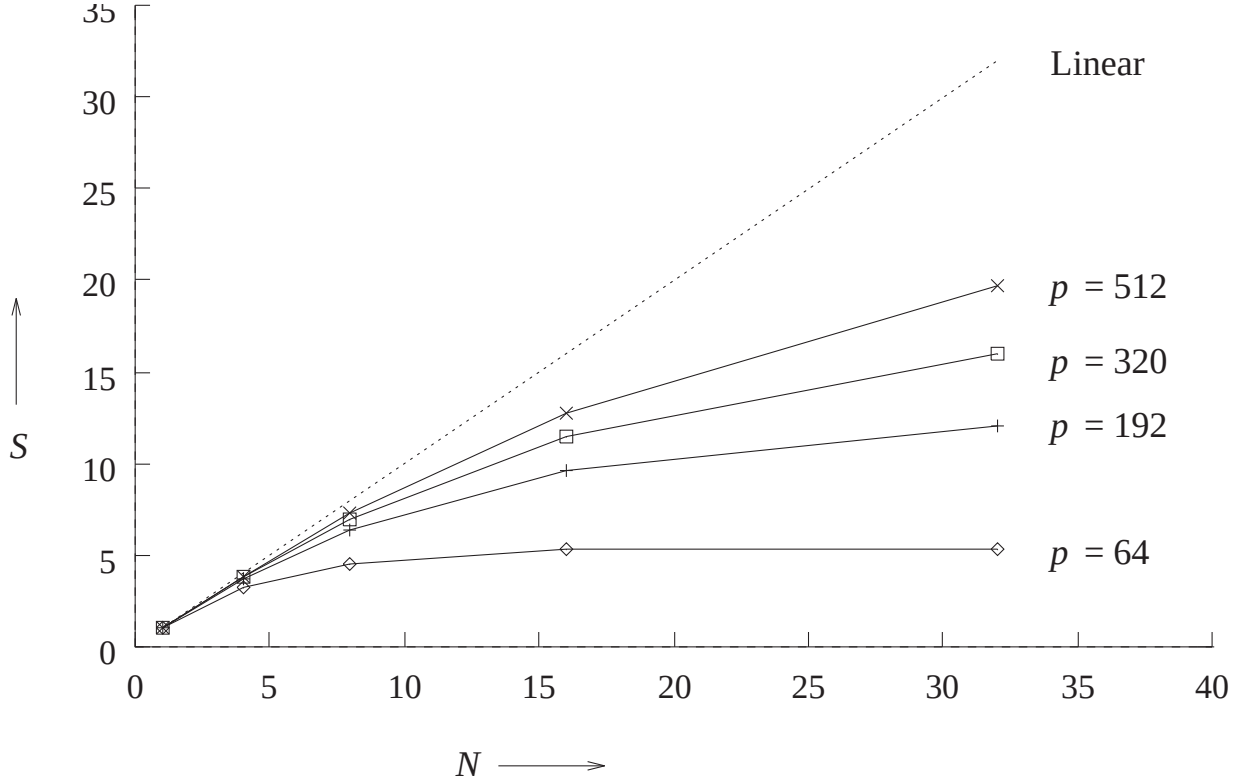
Parallel Sum ($N \leq p$)

- *Example:* $p = 16$ *numbers on* $N = 4$ *processors.*



Sum p Numbers on n-cube

- *Parallel summation done as before.*
- *Time complexity:* $p/N - 1 + 2logN$.



Sum p numbers on n-cube

- *Efficiency as a function of p and $N = 2^n$.*
- *Increasing the no. of processors reduces efficiency.*
- *Increasing the problem size increases efficiency.*
- *Keep efficiency constant by increasing problem size and no. of processors (system scalability).*

p	***N* = 1**	***N* = 4**	***N* = 8**	***N* = 16**	***N* = 32**
64	1.0	*.80*	.57	.33	.17
192	1.0	.92	*.80*	.60	.38
320	1.0	.95	.87	.71	.50
512	1.0	.97	.91	*.80*	.62

Sum p numbers on n-cube - Scalability

- *Speedup* $S = \frac{T_1}{T_N} = \frac{p-1}{\frac{p}{N}+2logN-1}$
- *Efficiency* $E = \frac{S}{N}$
- *Overhead function* $T_o = N \cdot T_N - T_1 \approx 2N \cdot logN - p$. Cost-optimal iff $T_o = O(T_1) \Rightarrow p = O(NlogN)$.
- *Isoefficiency* $W = O(NlogN)$.

If N increased to N', the problem size must be increased by $\frac{W'}{W} = \frac{N' \cdot logN'}{N \cdot logN}$ to keep the efficiency constant.

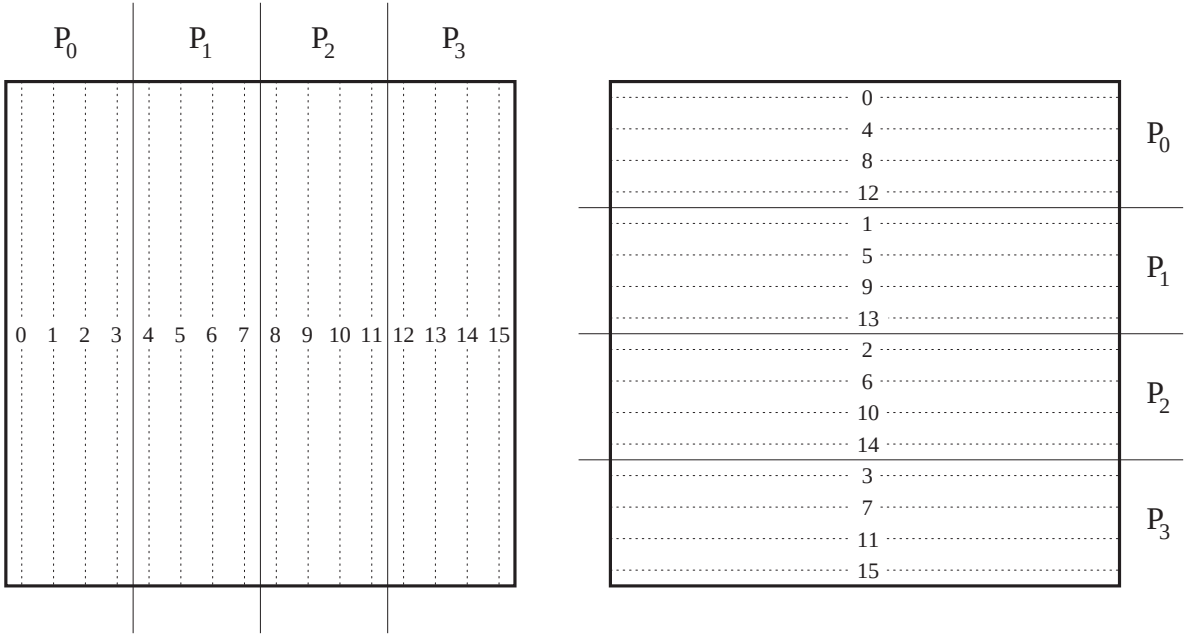
- *What is the minimum time (unlimited parallelism)?*

Minimize $T(N,p) = \frac{p}{N} + 2logN - 1$, *over all* N.

We find: $N_0 = \frac{p}{2}$. *Hence,* $T_{min}(N,p) = 2logN + 1$.

Matrix Allocation

- *Stripwise:* 16×16 *matrix on* 4 *processors.*



(a) Columwise block striping (b) Rowwise cyclic striping

Matrix Allocation

- *Checkerboard:* 8×8 *matrix on* 16 *processors.*

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
	P_0		P_1		P_2		P_3
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
	P_4		P_5		P_6		P_7
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
	P_8		P_9		P_{10}		P_{11}
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
	P_{12}		P_{13}		P_{14}		P_{15}
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

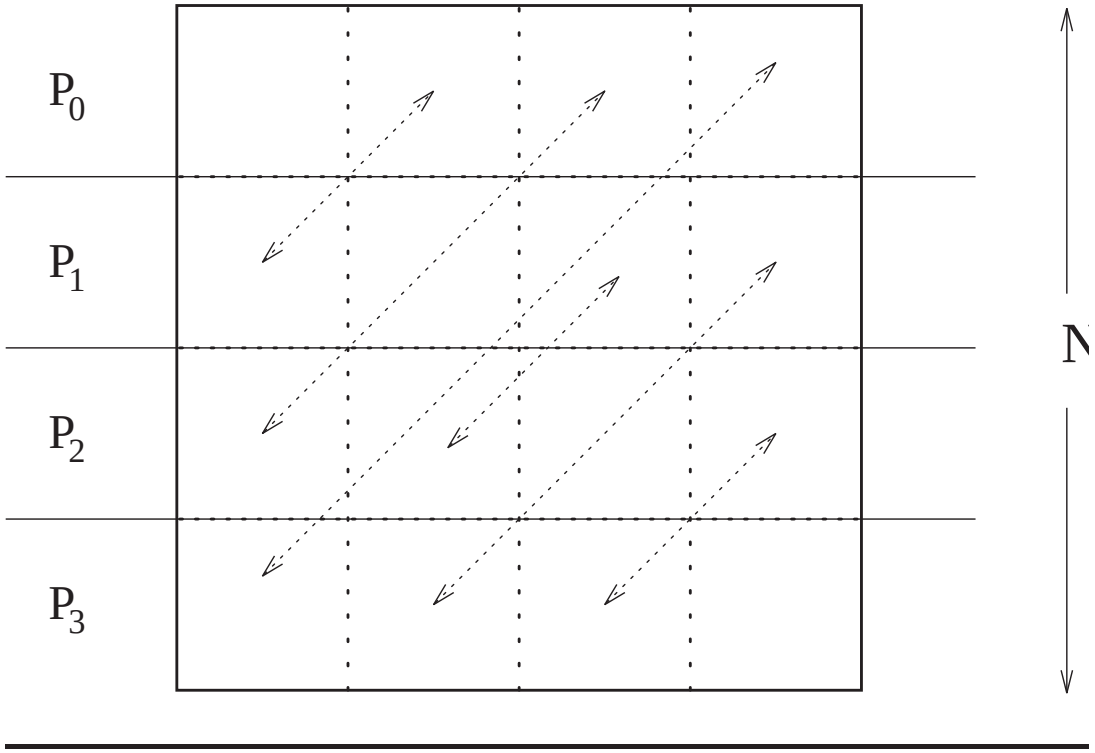
(a) Block-checkerboard partitioning

(0,0)	(0,4)	(0,1)	(0,5)	(0,2)	(0,6)	(0,3)	(0,7)
	P_0		P_1		P_2		P_3
(4,0)	(4,4)	(4,1)	(4,5)	(4,2)	(4,6)	(4,3)	(4,7)
(1,0)	(1,4)	(1,1)	(1,5)	(1,2)	(1,6)	(1,3)	(1,7)
	P_4		P_5		P_6		P_7
(5,0)	(5,4)	(5,1)	(5,5)	(5,2)	(5,6)	(5,3)	(5,7)
(2,0)	(2,4)	(2,1)	(2,5)	(2,2)	(2,6)	(2,3)	(2,7)
	P_8		P_9		P_{10}		P_{11}
(6,0)	(6,4)	(6,1)	(6,5)	(6,2)	(6,6)	(6,3)	(6,7)
(3,0)	(3,4)	(3,1)	(3,5)	(3,2)	(3,6)	(3,3)	(3,7)
	P_{12}		P_{13}		P_{14}		P_{15}
(7,0)	(7,4)	(7,1)	(7,5)	(7,2)	(7,6)	(7,3)	(7,7)

(b) Cyclic-checkerboard partitioning

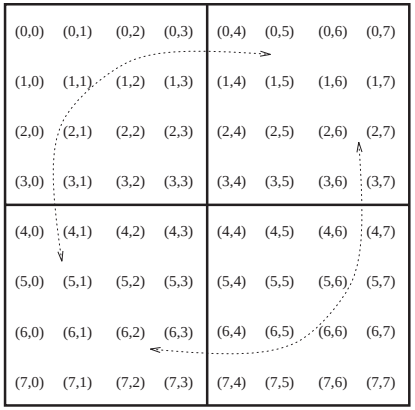
Matrix Transposition

- *Example: transpose* 4×4 *matrix on 4 processors.*
- *Total exchange communication pattern.*

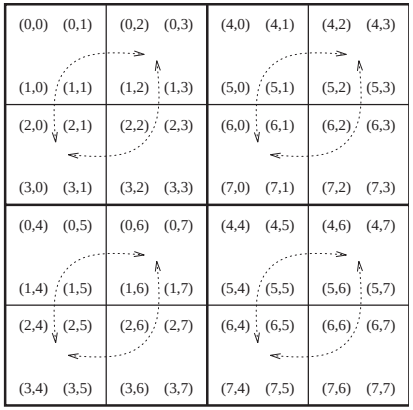


Recursive Matrix Transposition

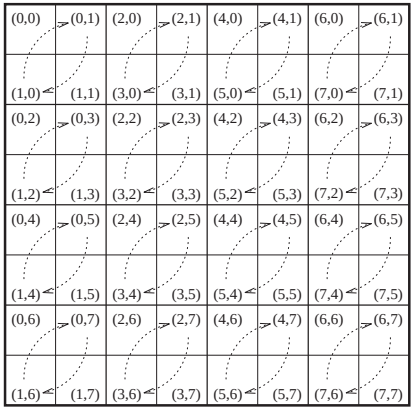
- *Example: Transpose an* 8×8 *matrix.*



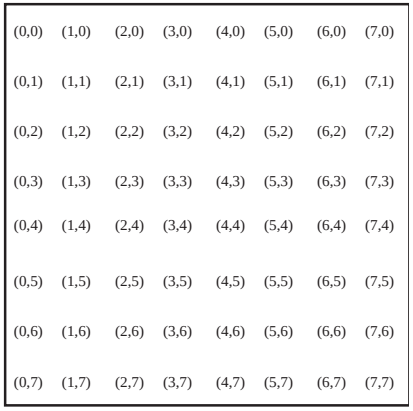
(a) Division of the matrix into four blocks and exchange of top-right and bottom-left blocks



(b) Division of each block into four subblocks and exchange of top-right and bottom-left subblocks



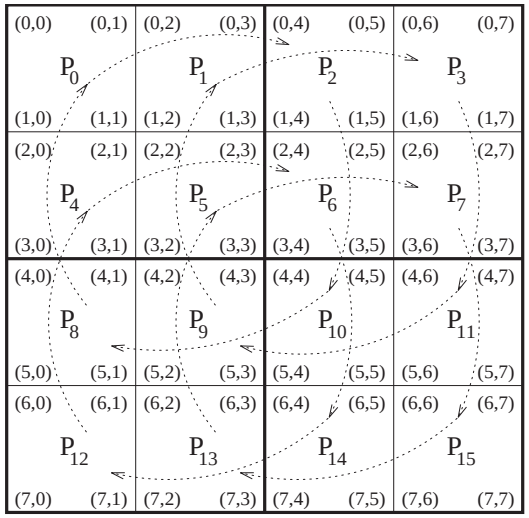
(c) Last subdivision and transposition



(d) Final configuration

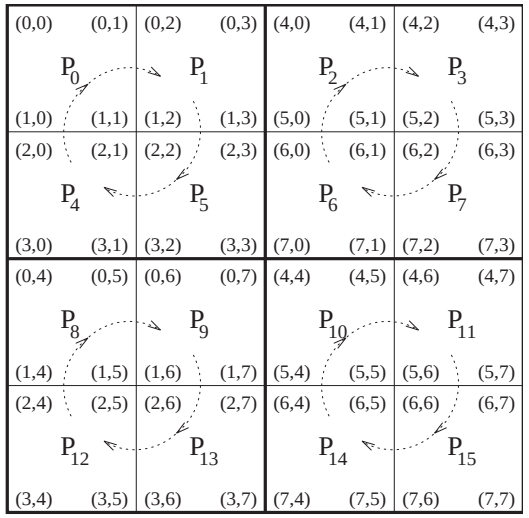
Matrix Transposition onto n-cube

- *Recursive transposition of* $p \times p$ *matrix on* 4*-cube.*
- *There are* $n/2$ *recursive phases.*
- *Store & forward time:* $p^2/(2N) + (t_s + p^2 \cdot t_w/N) \cdot n$.
- *Cut through time:* $0.5(t_s + 2t_h + p^2 \cdot t_w/N) \cdot n$.



(a)

Division of the matrix into four blocks and exchange of top-right and bottom-left blocks



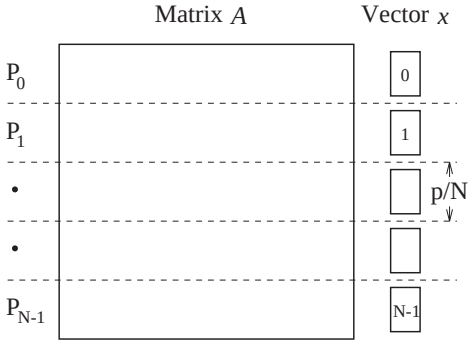
(b)

Division of each block into four subblocks and exchange of top-right and bottom-left subblocks

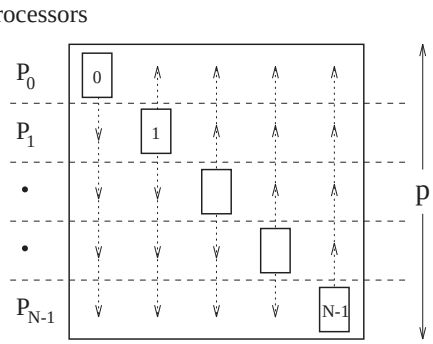
Matrix-Vector Multiplication

Multiply $p \times p$ matrix with $p \times 1$ vector on N processors using horizontal stripe allocation; p is multiple of N.

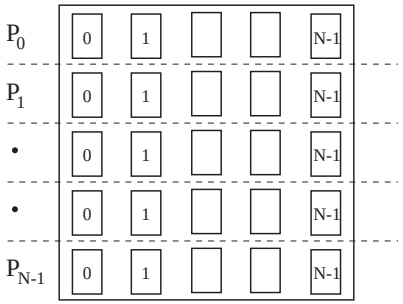
- *Time on n-cube:* $p^2/N + t_s \cdot n + t_w \cdot (p/N) \cdot (N-1)$.



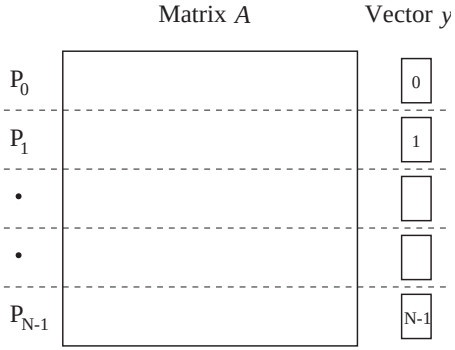
(a) Initial partitioning of the matrix and the starting vector x



(b) Distribution of the full vector among all the processors by all-to-all broadcast



(c) Entire vector distributed to each processor after the broadcast

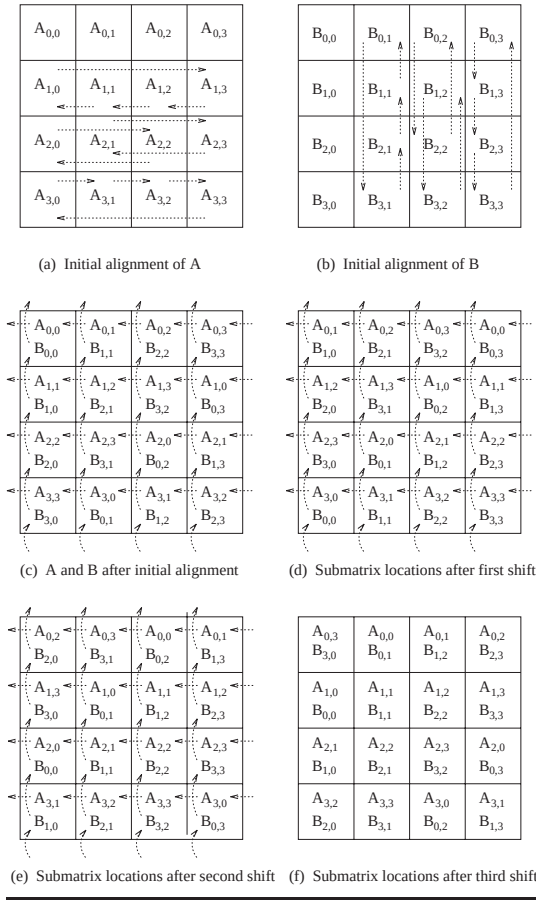


(d) Final distribution of the matrix and the result vector y

Matrix Multiplication - Cannon

Multiply $p \times p$ matrices partitioned into N blocks $A_{i,j}$ and $B_{i,j}$ of size $(p/\sqrt{N}) \times (p/\sqrt{N})$ and distributed to N processors. $O(p^2)$ memory.

- *Time on n-cube: $p^3/N + 2\sqrt{N} \cdot t_s + 2t_w \cdot p^2/\sqrt{N}$.*



(a) Initial alignment of A

(b) Initial alignment of B

(c) A and B after initial alignment

(d) Submatrix locations after first shift

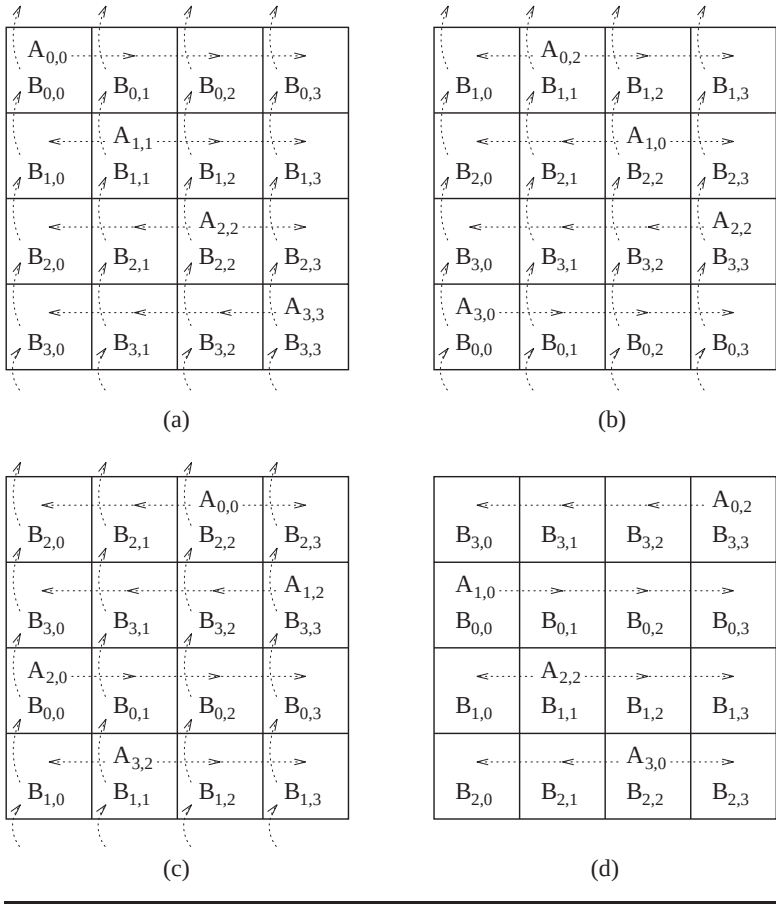
(e) Submatrix locations after second shift

(f) Submatrix locations after third shift

Matrix Multiplication - Fox

Multiply $p \times p$ matrices partitioned into N blocks $A_{i,j}$ and $B_{i,j}$ of size $(p/\sqrt{N}) \times (p/\sqrt{N})$ and distributed to N processors.

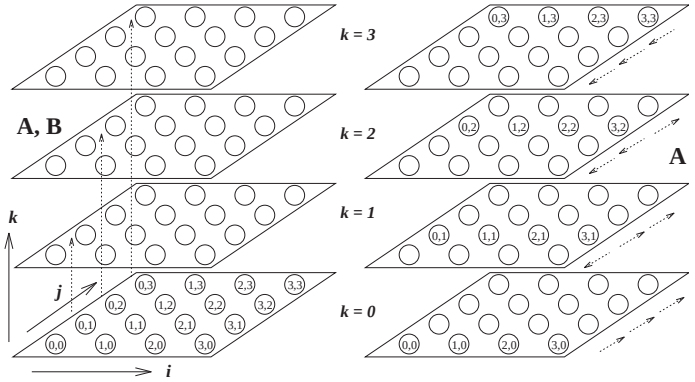
- *Time on n-cube: $p^3/N + 0.5n\sqrt{N}t_s + t_w p^2 n/\sqrt{N}$.*



Matrix Multiplication - DNS

Multiply $2^n \times 2^n$ matrices A, B on $N = 2^{3n}$ processors.

- *Overlap B (i axis) and A (k and j axis) broadcasts $\Rightarrow 3n$ time on $3n$-cube; $C_{i,j}$ at processor $C_{i,j,0}$.*
- *Normal algorithm with $4n$ time complexity.*



(a) Initial distribution of A and B

(b) $A[i,j]$ moved from $P_{i,j,0}$ to $P_{i,j,j}$

(c) All $A[i,j]$ broadcast along j axis

(d) Corresponding distribution of B

Tridiagonal - Cyclic Reduction

Solve $A \cdot x = b$, *with* $A_{m \times m}$ *tridiagonal,* $m = 2^n - 1$.

$$A = \begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 & \dots & 0 & 0 & 0 \\ . & . & . & . & . & . & \dots & . & . & . \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & a_{m-1} & b_{m-1} & c_{m-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & b_m & c_m \end{bmatrix}.$$

Balanced cyclic reduction applied during elimination. The algorithm considers triads of equations, eliminating x_{i-1} *and* x_{i+1}, *and relating* x_{i-2}, x_i, x_{i+2}.

DO $i = 2, 4, \dots, 2^n - 2$

$$\begin{aligned} a_{i-1}x_{i-2} + b_{i-1}x_{i-1} + c_{i-1}x_i &= d_{i-1} \\ a_i x_{i-1} + b_i x_i + c_i x_{i+1} &= d_i \\ a_{i+1}x_i + b_{i+1}x_{i+1} + c_{i+1}x_{i+2} &= d_{i+1} \end{aligned}$$