Parallel Algorithms

Thoai Nam

Outline

- Introduction to parallel algorithms development
- Reduction algorithms
- Broadcast algorithms
- Prefix sums algorithms

Introduction to Parallel Algorithm Development

- Parallel algorithms mostly depend on destination parallel platforms and architectures
- MIMD algorithm classification
  - Pre-scheduled data-parallel algorithms
  - Self-scheduled data-parallel algorithms
  - Control-parallel algorithms
- According to M.J.Quinn (1994), there are 7 design strategies for parallel algorithms

Basic Parallel Algorithms

- 3 elementary problems to be considered
  - Reduction
  - Broadcast
  - Prefix sums
- Target Architectures
  - Hypercube SIMD model
  - 2D-mesh SIMD model
  - UMA multiprocessor model
  - Hypercube Multicomputer

Reduction Problem

- Description: Given *n* values $a_0, a_1, a_2 \ldots a_{n-1}$, an associative operation $\oplus$, let's use p processors to compute the *sum:*

  $$S = a_0 \oplus a_1 \oplus a_2 \oplus \ldots \oplus a_{n-1}$$

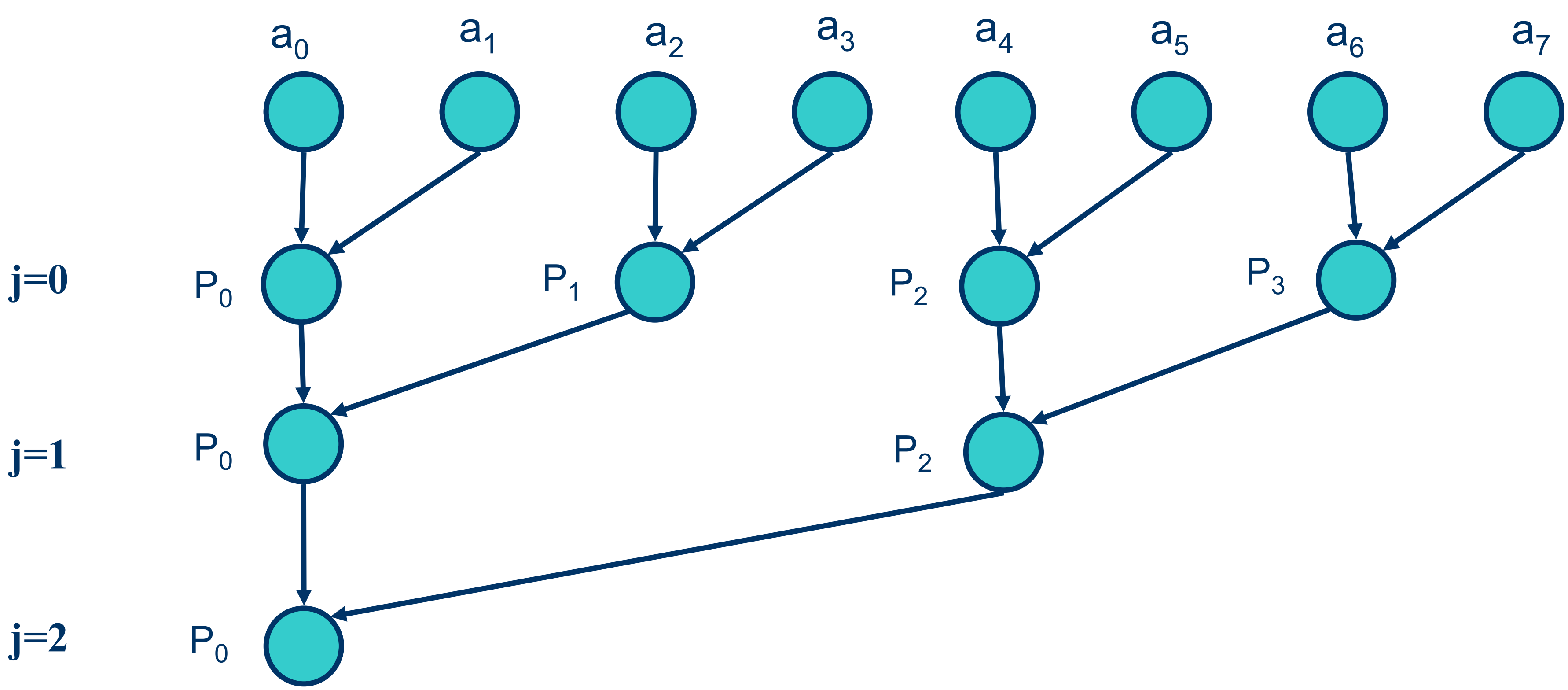
- **Design strategy 1**
  - "If a cost optimal CREW PRAM algorithms exists and the way the PRAM processors interact through shared variables maps onto the target architecture, a PRAM algorithm is a reasonable starting point"

Cost Optimal PRAM Algorithm for the Reduction Problem

- Cost optimal PRAM algorithm complexity:

  $O(\log n)$ (using n div 2 processors)

- Example for n=8 and p=4 processors



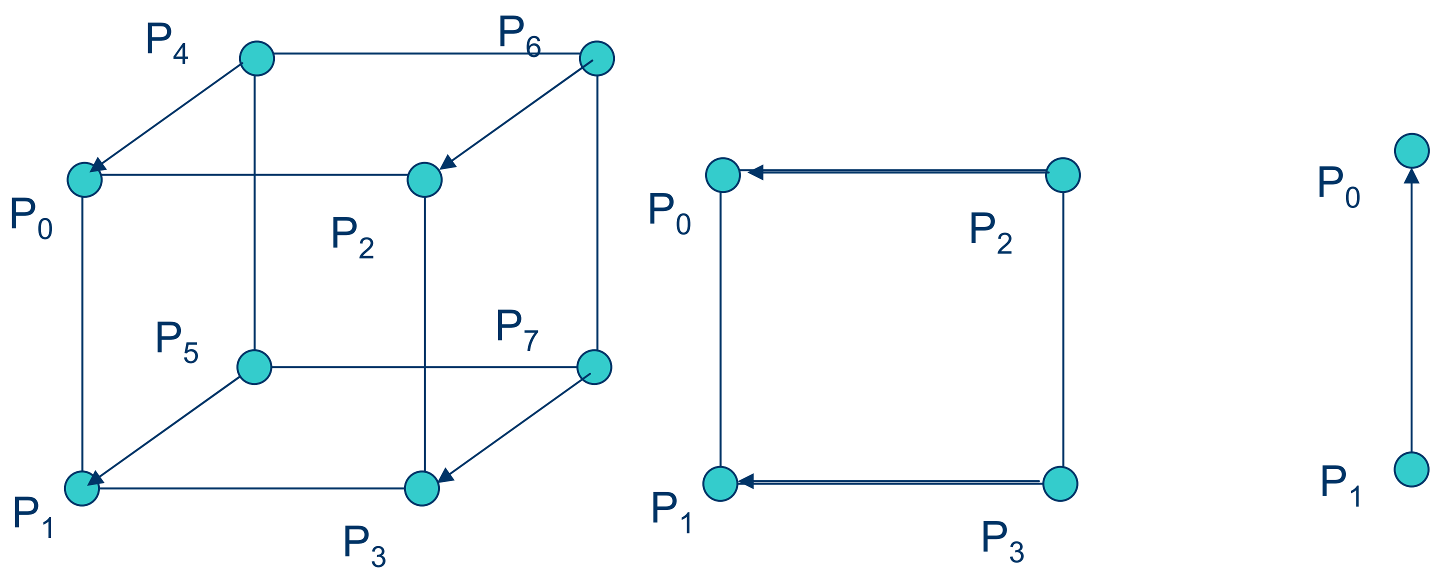
Cost Optimal PRAM Algorithm for the Reduction Problem(cont'd)

Using p= n div 2 processors to add n numbers:

Global a[0..n-1], n, i, j, p;
Begin
  spawn(P_0, P_1,… ,P_{p-1});
  for all P_i where $0 \le i \le$ p-1 do
    for j=0 to ceiling(logp)-1 do
      if i mod 2^j =0 and 2i + 2^j < n then
        a[2i] := a[2i] $\oplus$ a[2i + 2^j];
      endif;
    endfor j;
  endforall;
End.

Notes: the processors communicate in a biominal-tree pattern

Solving Reducing Problem on Hypercube SIMD Computer



Step 1:

Reduce by dimension j=2

Step 2:

Reduce by dimension j=1

Step 3:

Reduce by dimension j=0

The total sum will be at P_0

Solving Reducing Problem on Hypercube SIMD Computer (cond't)

<u>Using p processors to add n numbers (p << n)</u>

Global j;

Local local.set.size, local.value[1..n div p +1], sum, tmp;

Begin

spawn(P_0, P_1,… ,P_{p-1});

for all P_i where $0 \le i \le$ p-1 do

if (i < n mod p) then local.set.size:= n div p + 1

else local.set.size := n div p;

endif;

sum[i]:=0;

endforall;

Allocate workload for each processors

Solving Reducing Problem on Hypercube SIMD Computer (cond't)

Calculate the partial sum for each processor

```
for j:=1 to (n div p +1) do
   for all P_i where 0 ≤ i ≤ p-1 do
         if local.set.size ≥ j then
  sum[i]:= sum ⊕ local.value [j];
      endforall;
endfor j;
```

Solving Reducing Problem on Hypercube SIMD Computer (cond't)

Calculate the total sum by reducing for each dimension of the hypercube

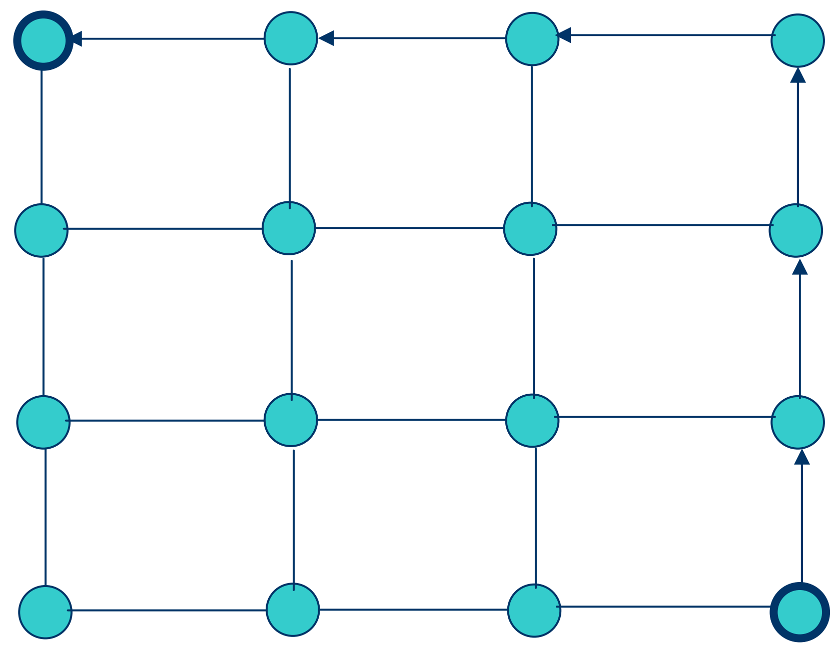
```
for j:=ceiling(logp)-1 downto 0 do
    for all P_i where 0 ≤ i ≤ p-1 do
    if i < 2^j then
        tmp := [i + 2^j]sum;
        sum := sum ⊕ tmp;
        endif;
  endforall;
endfor j;
```

Solving Reducing Problem on 2D-Mesh SIMD Computer

- ❑ A 2D-mesh with p*p processors need at least 2(p-1) steps to send data between two farthest nodes

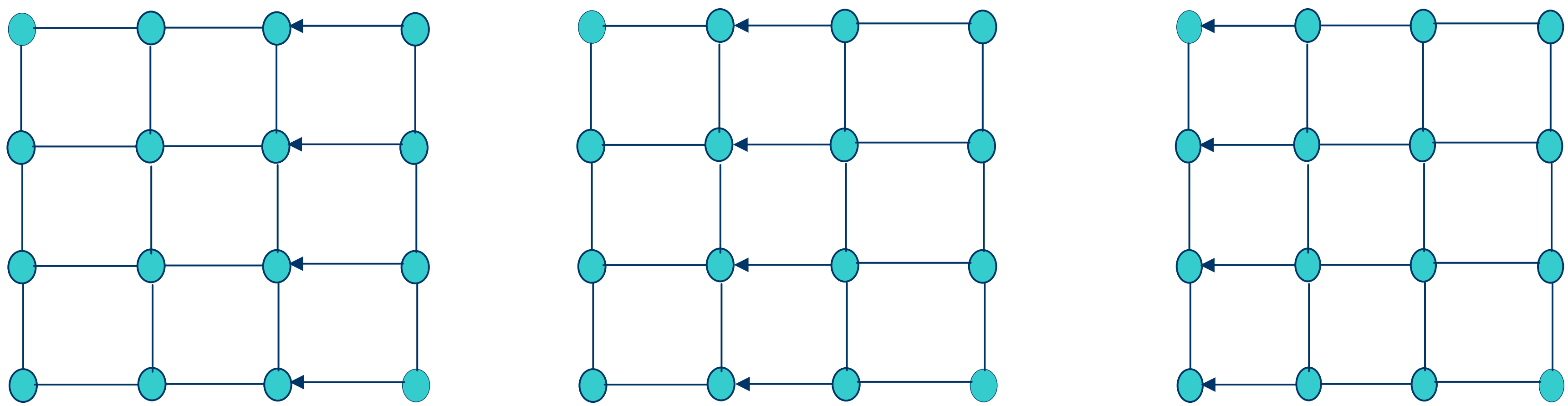
➔ The lower bound of the complexity of any reduction sum algorithm is $0(n/p^2 + p)$

Example: a 4*4 mesh need 2*3 steps to get the subtotals from the corner processors



Solving Reducing Problem on 2D-Mesh SIMD Computer(cont'd)

- Example: compute the total sum on a 4*4 mesh



Stage 1

Step i = 3

Stage 1

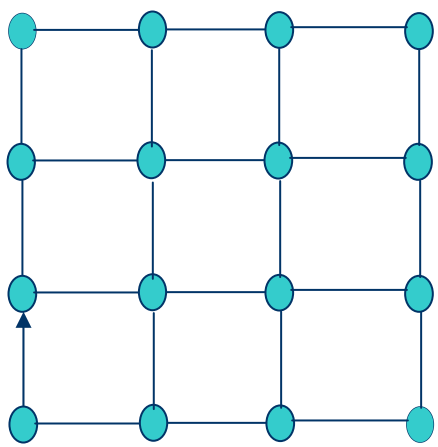
Step i = 2

Stage 1

Step i = 1

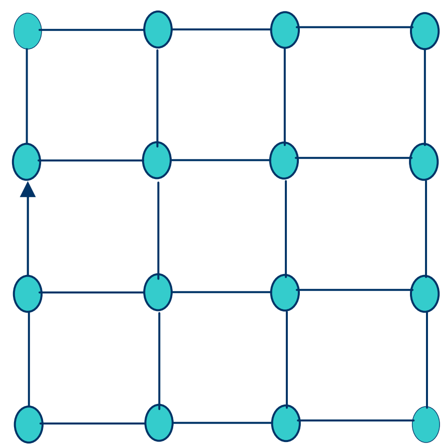
Solving Reducing Problem on 2D-Mesh SIMD Computer(cont'd)

- Example: compute the total sum on a 4*4 mesh



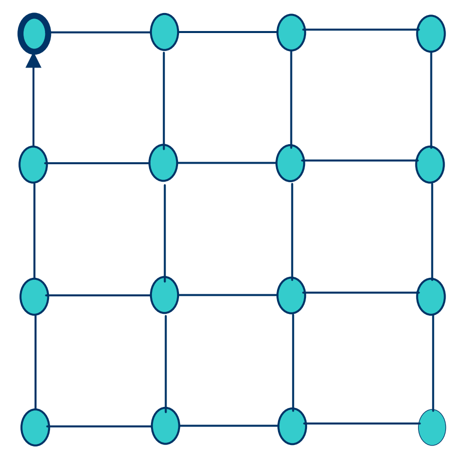
Stage 2

Step i = 3



Stage 2

Step i = 2



Stage 2

Step i = 1

(the sum is at $P_{1,1}$)

Solving Reducing Problem on 2D-Mesh SIMD Computer(cont'd)

<u>Summation (2D-mesh SIMD with l*l processors</u>

```
Global i;
Local tmp, sum;
Begin
  {Each processor finds sum of its local value →
    code not shown}
  for i:=l-1 downto 1 do
    for all P(j,i) where 1 ≤ i ≤ l do
          {Processing elements in colum i active}
          tmp := right(sum);
          sum:= sum ⊕ tmp;
       end forall;
  endfor;
```

Stage 1:

$P_{i,1}$ computes the sum of all processors in row i-th

Solving Reducing Problem on 2D-Mesh SIMD Computer(cont'd)

Stage2:

Compute the total sum and store it at $P_{1,1}$

```
for i:= l-1 downto 1 do
   for all  Pi,1 do
      {Only a single processing element active}
      tmp:=down(sum);
      sum:=sum ⊕ tmp;
   end forall;
 endfor;
End.
```

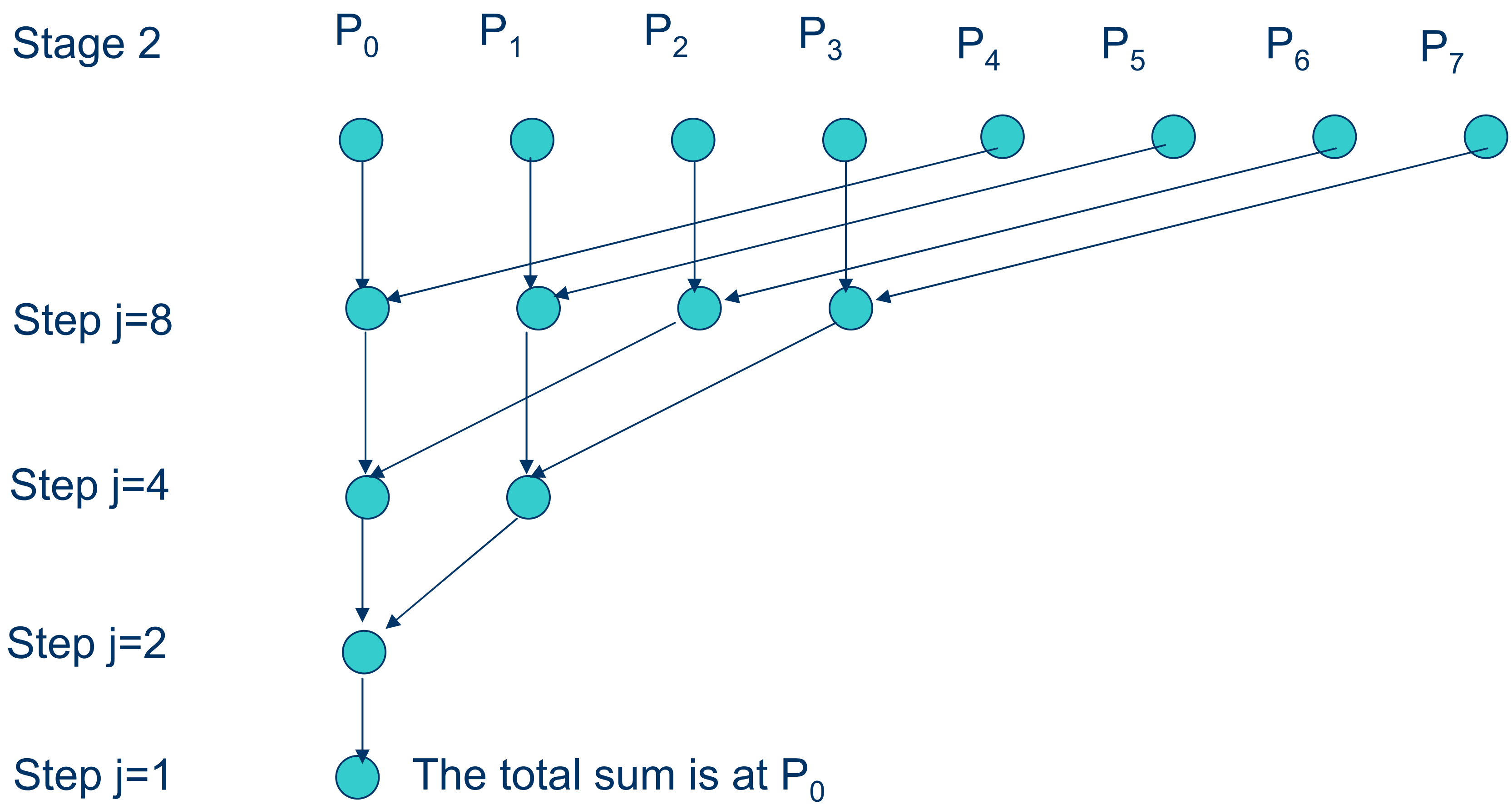
Solving Reducing Problem on UMA Multiprocessor Model(MIMD)

- Easily to access data like PRAM
- Processors execute asynchronously, so we must ensure that no processor access an "unstable" variable
- Variables used:

Global	a[0..n-1],	{values to be added}
	p,	{number of proeessor, a power of 2}
	flags[0..p-1],	{Set to 1 when partial sum available}
	partial[0..p-1],	{Contains partial sum}
	global_sum;	{Result stored here}
Local	local_sum;	

Solving Reducing Problem on UMA Multiprocessor Model(cont'd)

- Example for UMA multiprocessor with p=8 processors



Solving Reducing Problem on UMA Multiprocessor Model(cont'd)

Summation (UMA multiprocessor model)

Stage 1:

Each processor computes the partial sum of n/p values

```
Begin
  for k:=0 to p-1 do flags[k]:=0;
  for all Pi where 0 ≤ i < p do
     local_sum :=0;
     for j:=i to n-1 step p do
          local_sum:=local_sum ⊕ a[j];
```

Solving Reducing Problem on UMA Multiprocessor Model(cont'd)

Stage 2:

Compute the total sum

Each processor waits for the partial sum of its partner available

```
   j:=p;
   while j>0 do begin
     if i ≥  j/2 then
       partial[i]:=local_sum;
       flags[i]:=1;
       break;
     else
       while (flags[i+j/2]=0) do;
       local_sum:=local_sum ⊕ partial[i+j/2];
     endif;
     j=j/2;
   end while;
   if i=0 then global_sum:=local_sum;
 end forall;
End.
```

Solving Reducing Problem on UMA Multiprocessor Model(cont'd)

- Algorithm complexity 0(n/p+p)
- What is the advantage of this algorithm compared with another one using critical-section style to compute the total sum?
- **Design strategy 2:**
  - Look for a data-parallel algorithm before considering a control-parallel algorithm

➔ On MIMD computer, we should exploit both data parallelism and control parallelism

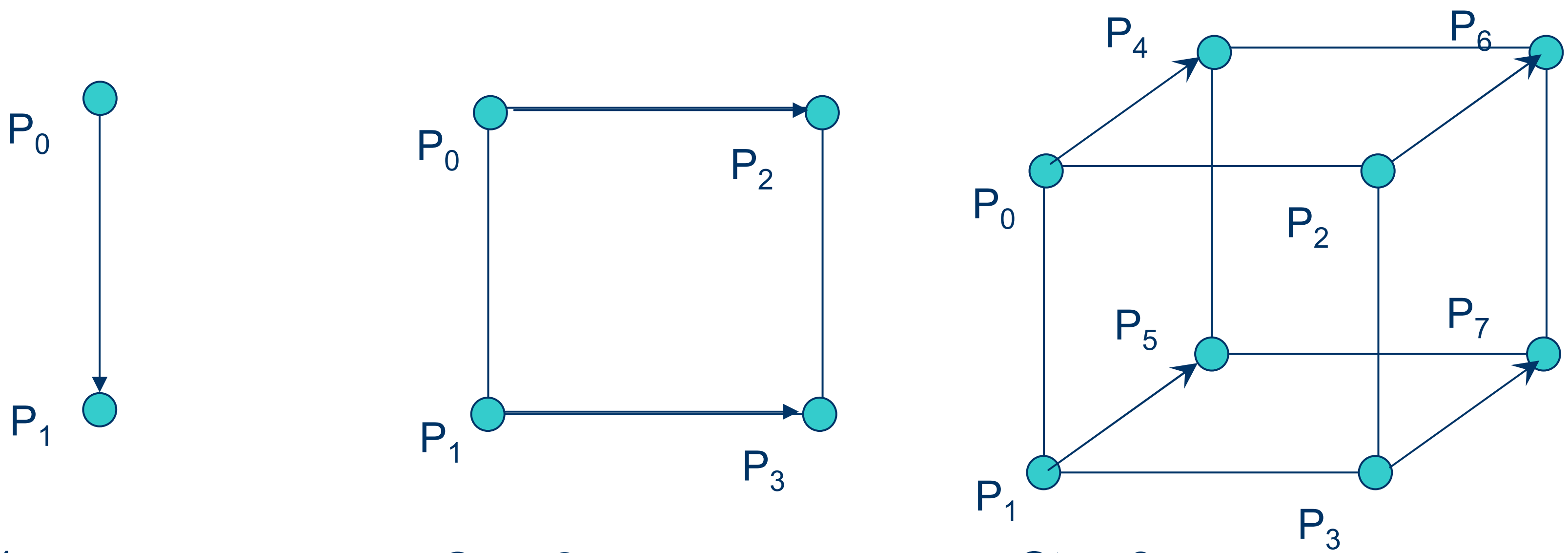
(try to develop SPMD program if possible)

Broadcast

- Description:
  - Given a message of length M stored at one processor, let's send this message to all other processors
- Things to be considered:
  - Length of the message
  - Message passing overhead and data-transfer time

Broadcast Algorithm on Hypercube SIMD

- If the amount of data is small, the best algorithm takes **logp** communication steps on a **p-node** hypercube
- Examples: broadcasting a number on a **8-node** hypercube



Step 1:

Send the number via the 1st dimension of the hypercube

Step 2:

Send the number via the 2nd dimension of the hypercube

Step 3:

Send the number via the 3rd dimension of the hypercube

Broadcast Algorithm on Hypercube SIMD(cont'd)

Broadcasting a number from P_0 to all other processors

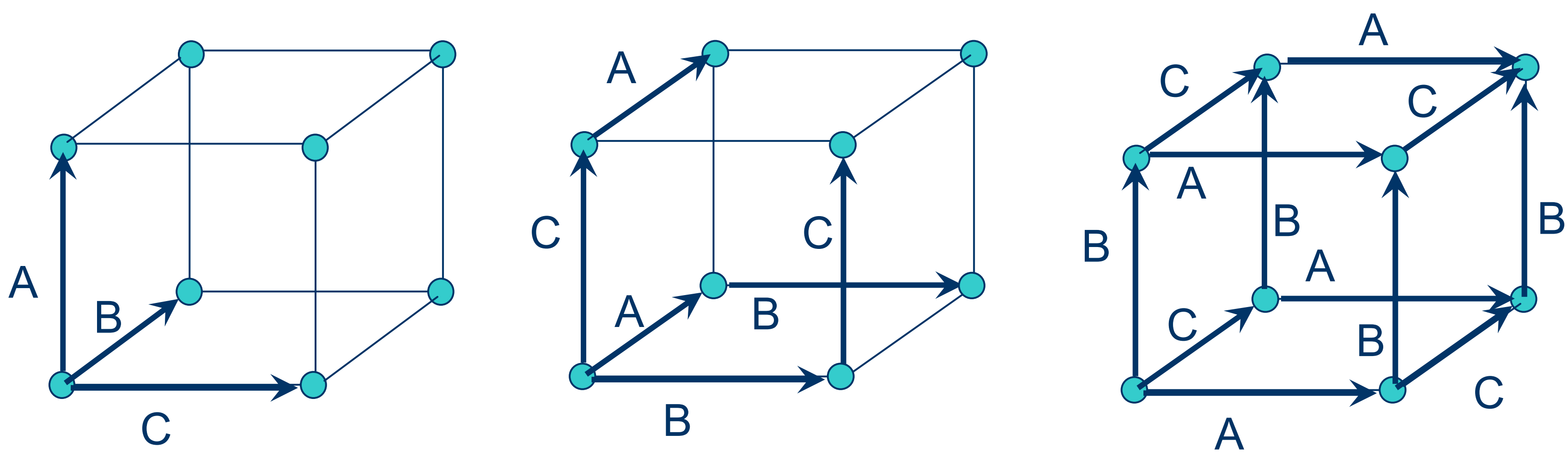
```
Local    i,          {Loop iteration}
         p,          {Partner processor}
         position;   {Position in broadcast tree}
         value;      {Value to be broadcast}
Begin
  spawn(P0, P1,... ,Pp-1);
  for j:=0 to logp-1 do
    for all Pi where 0 ≤ i ≤  p-1 do
      if i < 2^j  then
           partner := i+2^j;
         [partner]value:=value;
           endif;
    endforall;
 end forj;
End.
```

Broadcast Algorithm on Hypercube SIMD(cont'd)

- The previous algorithm
  - Uses at most p/2 out of plogp links of the hypercube
  - Requires time Mlogp to broadcast a length M msg

  ➔not efficient to broadcast long messages
- Johhsson and Ho (1989) have designed an algorithm that executes logp times faster by:
  - Breaking the message into logp parts
  - Broadcasting each parts to all other nodes through a different biominal spanning tree

Johnsson and Ho's Broadcast Algorithm on Hypercube SIMD



- Time to broadcast a msg of length M is Mlogp/logp = M
- The maximum number of links used simultaneously is plogp, much greater than that of the previous algorithm

Johnsson and Ho's Broadcast Algorithm on Hypercube SIMD(cont'd)

- Design strategy 3
  - As problem size grow, use the algorithm that makes best use of the available resources

Prefix SUMS Problem

- Description:
  - Given an associative operation ⊕ and an array A containing n elements, let's compute the n quantities
    - A[0]
    - A[0] ⊕ A[1]
    - A[0] ⊕ A[1] ⊕ A[2]
    - …
    - A[0] ⊕ A[1] ⊕ A[2] ⊕ … ⊕ A[n-1]
- Cost-optimal PRAM algorithm:
  - "Parallel Computing: Theory and Practice", section 2.3.2, p. 32

Prefix SUMS Problem on Multicomputers

- Finding the prefix sums of 16 values

	Processor 0				Processor 1				Processor 2				Processor 3			
(a)	3	2	7	6	0	5	4	8	2	0	1	5	2	3	8	6
(b)	18				17				8				19			
(c)	18	35	43	62	18	35	43	62	18	35	43	62	18	35	43	62
(d)	3	5	12	18	18	23	27	35	37	37	38	43	45	48	56	62

Prefix SUMS Problem on Multicomputers(cont'd)

- Step (a)
  - Each processor is allocated with its share of values
- Step (b)
  - Each processor computes the sum of its local elements
- Step (c)
  - The prefix sums of the local sums are computed and distributed to all processor
- Step (d)
  - Each processor computes the prefix sum of its own elements and adds to each result the sum of the values held in lower-numbered processors