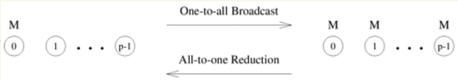
Network is bidirectional

- send/recv는 따로 계산안함?

One-All communication



- 'one'이 들어가면 single data 단위로 처리함
- broadcast … MPI_Bcast
  : send 1 message (size M) to (P-1) nodes
- reduction … MPI_Reduce
  : recv 1 message (size M) from each nodes
  combine with associative ops

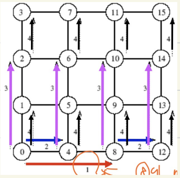
- Cost

  T = total time taken by procedure (whole)

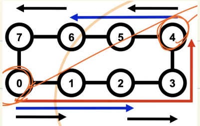
  = $(t_s + t_w \cdot m)(\log P)$ 배수

  start transfer-msg size

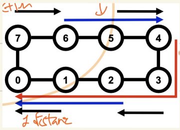
step 진행될수록 concurrency ↑



- Network Topologies
  - naive
    : source node가 다 처리함 (hotspot/bottleneck)
    → (P-1) stages
  - Ring
    → recursive doubling이 핵심!
    → $(\log_2 P)$ stages

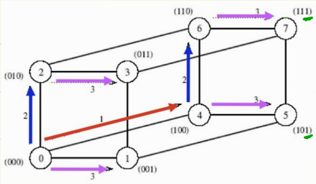


P/2 distance 부터 halving



1 distance 부터 doubling

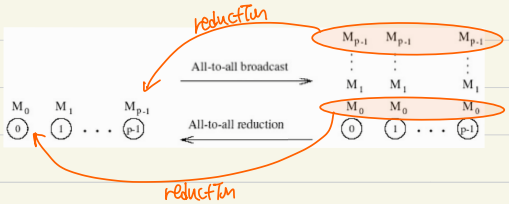
  - Mesh (2D)
    : square mesh of p nodes < $\sqrt{p}$ rows, $\sqrt{p}$ columns
    → steps
    ① row 안에서 ops
    : recursive doubling으로 broadcast/reduction
    ② column 안에서 ops
    : recursive doubling으로 broadcast/reduction
  - Hypercube (3D)
    : 2^d nodes 에 대해서 d-dimension으로 만들기
    : each dimension 마다 2nodes 만 존재함



→ communication은 MSB 부터 LSB order로 진행된다

All-All Collective Communication

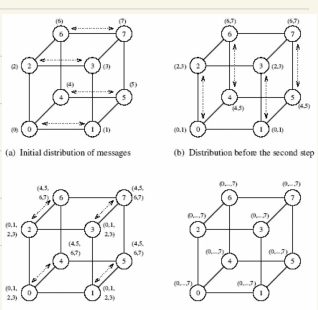
: 모든 node가 one-All communication 하는 것



- Topology 따라 time cost가 다르다!
  - step다름
  - msg 또한 다름

- Broadcast … MPI_Allgather
- Reduction … MPI_Reduce_scatter

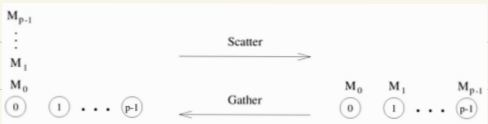
- network topology
  - naive
    → broadcast : 각 node가 1msg를 (p-1)-time send
    → reduction : 각 node가 (p-1)개의 msg reduce
  - ring
    * 이미 recursive doubling 한 경우 X
    → broadcast : 각 node가 neighbor node로 prev step에 recv한 msg를 전달한다.
    → Time : $(t_s + t_w m)(p-1)$
    → reduction : 각 node가 neighbor node로 msg send.
    recv node는 reduction하기.
    → Time : $(t_s + t_w \cdot m)(p-1)$
  - Mesh
    → broadcast
    Step ① row-wise (each msg size m) ring base broadcast
    → Time : $(t_s + t_w \cdot m)(\sqrt{p}-1)$
    ② column-wise (each msg size $m\sqrt{p}$) ring base broadcast
    → Time : $(t_s + t_w \cdot m\sqrt{p})(\sqrt{p}-1)$
    → reduction
    : mesh랑 오히려 반대로 반대로 함



  - hypercube
    → 1 node당 3 neighbor가 있는데,
    3 steps를 거치면서 neighbor랑 모두 communicate 해야함
    broadcast → step 진행할수록 msg size doubling 됨
    → Time
    $= \sum_{i=1}^{\log p} (t_s + 2^{i-1} \cdot m t_w)$

Scatter/gather

- Scatter ··· MPI_Scatter
  : 하나 가진 mp msg를 재분할 p node에게 나눠줄다
- gather ··· MPI_Gather
  : (p-1) node로 부터 1 msg씩 append 한다

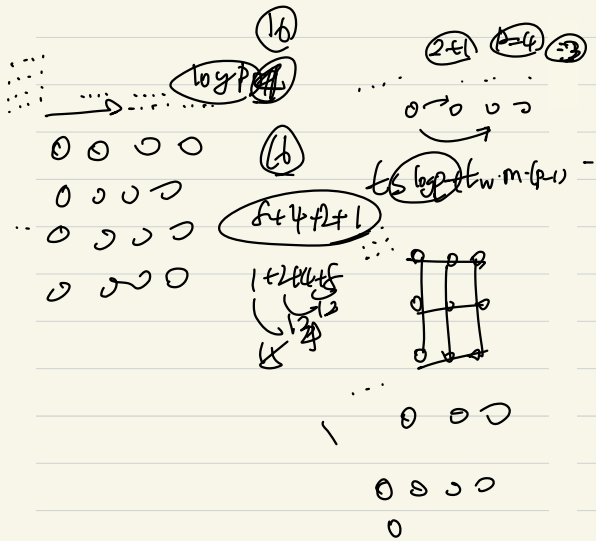


→ 모든 msg는 unique하다.

- Network Topology

- naive
  → Scatter : home node가 하나 다 보내줌
    · Time = $(t_s + t_w \cdot m)(p-1)$
  → gather : 각각 home node로 1개씩 append (contention)
    · Time = $(t_s + t_w \cdot m)(p-1)$

- Ring
  → Scatter : recursive doubling 방식 msg size 줄이!
    · Time = $\sum_{i=1}^{\log p} (t_s + t_w \cdot m \cdot 2^{i-1})$
  → gather : recursive doubling 방식 msg size 늘이!
    · Time = $\sum_{i=1}^{\log p} (t_s + t_w \cdot m \cdot 2^{i-1})$



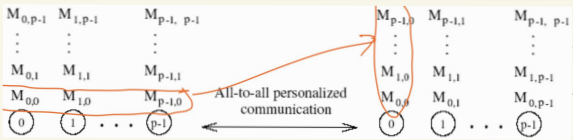
- Mesh
  → Scatter
  step① row wise로 data halving
    · Time = $\sum_{i=1}^{\log\sqrt{p}} (t_s + t_w \cdot m \cdot p/2^i) = \log p \cdot t_s + t_w m(p-1)$
  step② column wise로 data halving
    · Time = $\sum_{i=1}^{\log\sqrt{p}} (t_s + t_w \cdot (p/\sqrt{p}) \cdot 2^i)$
  → gather
  같은듯! 방향만 반대

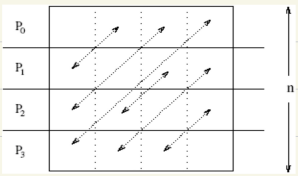
- hypercube
  : by step마다 msg 받은 node는 sender가 된다.
  → Time : $\sum_{i=1}^{\log p} t_s + t_w \cdot m \cdot 2^{i-1}$

All-All Personalized Communication

- matrix transpose 할 때 필요!

All이 communicate 하여 처리하는 법!





: 여러 개 (neighbor로) 한 번에 처리하기

- Time = $\sum_{T=1}^{P-1} (t_s + t_w \cdot m \cdot (P-T))$

One / All Collective Communication

Assumption

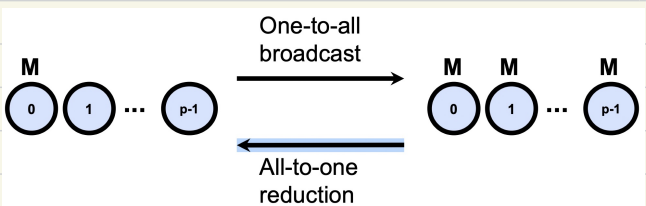
- Network is bidirectional
- communication is single-ported
  ↳ stepp에서 1 message만 recv 할 수 있음
  Multi-message 불가능

One-to-All

- broadcast
  : Processor가 M units of data를 갖고있고, 그 data를 other 모두에게 send

All-to-one

- reduction
  : each processor가 M units of data 갖고있고,
  associative operator로 combine해서
  target processor의 result 계산함



One / All Collective Communication (Cont.)

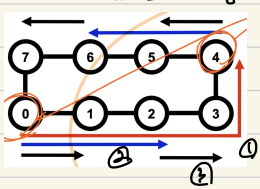
Ring Protocol

: network가 ring 형태로 구성되어 있을 때!

Broadcast

- naive solution : $(P-1)$ stages
  2명 1명이 순차적으로 다 전달해주면 됨!
  → $8-1=7$ stage 필요

- recursive doubling : $\log_2 P$ stages
  send distance를 halving 해가면서 communication 진행

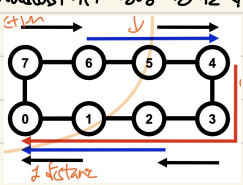


- 맨처음 sender : No
  No → N4 (8/2)
- data 가진 node는 본인 sender 역할 수행함
  ① → ② → ③

→ $\log_2 8$ = 3 stage 필요

Reduction

: Broadcast에서 방향 반대로 수행

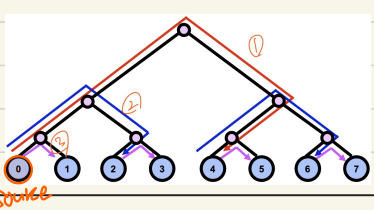


- 이번 반대로 distance 1씩 늘려가면서 수행

Balanced Binary Tree Protocol

Broadcast

- Processor가 leaves에 있음.
- left-most leaf = source of broadcast
- recursive doubling 사용해줌 (Chaining)



$\log_2 P$ steps 사용함

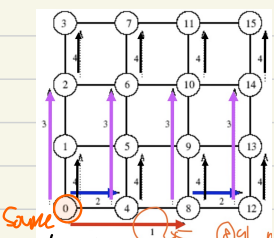
One/All Collective Communication (Cont.)

2D Mesh Protocol

- Square mesh (2차원) of p nodes
  - row : $\sqrt{p}$ 개의 linear array of node
  - column : $\sqrt{p}$ 개의 linear array of node

Steps

① Perform operation along a row
② Perform operation each column concurrently) the steps의 Concurrency 최대화되도록!



- T_1 : n_0 send to n_8 (row / recursive doubling)
- T_2 : $n_0 \to n_4$ / $n_8 \to n_{12}$ degree of concurrency = 2
  Every row has data!
- T_3 : $n_0 \to n_2$ / $n_4 \to n_6$ / $n_8 \to n_{10}$ / $n_{12} \to n_{14}$
- T_4 : column 마다 data 있는 애들 다 각각

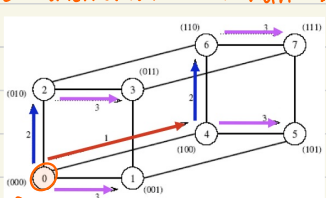
T_1 갈수록 degree of concurrency ↑

*reduction은 거꾸로 대칭적으로! (degree of concurrency 초반에 제한)

Hypercube protocol

: hypercube with 2^d nodes

- d-dimensional mesh with 2 nodes in each dimension

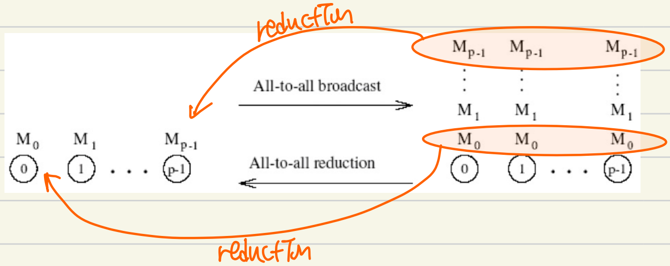


: d steps 만큼 완료됨 ($d = \log_2 2^d = \log_2 p$)

→ total cost = $\sum_{i=1}^{\log p} (t_s + 2^{i-1} t_w m)$ (broadcast 개념이 size of message)

All / All Collective Communication

	· each processor는 sender이자 receiver
All-to-All Broadcast	: 각자가 자신의 data를 모두에게 copy
All-to-All Reduction	: each processor는 copy of result를 가짐 (one to N의 2개인!) — 값은 다를수있음. data 교환은 동일함



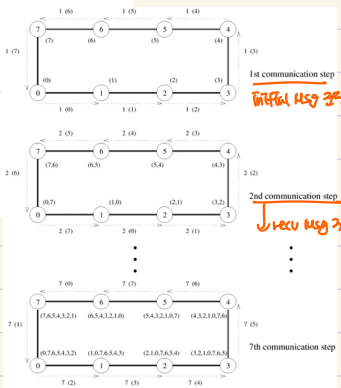
All / All Collective Communication (Cont.)

Ring Protocol

Broadcast : 매 timestep마다 right로 send하고 left로부터 recv한 msg의 내용을 1씩 늘림

```
1.   procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
2.   begin
3.       left := (my_id - 1) mod p;
4.       right := (my_id + 1) mod p;
5.       result := my_msg;
6.       msg := result;
7.       for i := 1 to p - 1 do
8.           send msg to right;
9.           receive msg from left;
10.          result := result ∪ msg;
11.      endfor;
12.  end ALL_TO_ALL_BC_RING
```

(초기값 msg, 시계 방향으로, append)



(초기값 msg 보냄, recv msg 보냄)

- cost per node
  $= t_s + t_w \cdot m$
  (t_s: startup time, t_w: single-word transfer time, m: # of message)

Reduction : combine each incoming message into local result

- for each step, recv msg를 result와 send 한다

```
procedure ALL_TO_ALL_RED_RING(my_id, my_msg, p, result)
begin
    left := (my_id - 1) mod p;
    right := (my_id + 1) mod p;
    recv := 0;
    for i := 1 to p - 1 do
        j := (my_id + i) mod p;
        temp := msg[j] + recv;
        send temp to left;
        receive recv from right;
    endfor;
    result := msg[my_id] + recv;
end ALL_TO_ALL_RED_RING
```

(즉, node마다 (p-1) communication 반복함, reduce)

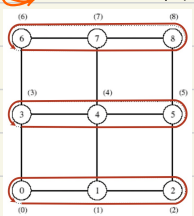
All / All Collective Communication (Cont.)

Mesh

broadcast

- 2 phases 로 이루어졌다.

① row-wise All-to-All Communication



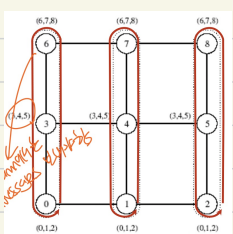
→ each node collects $\sqrt{p}$ messages (each size m)

→ 전송 message 크기는 기존에 $m\sqrt{p}$ 가 된다.

→ Cost per node

$= (t_s + t_w m)(\sqrt{p} - 1)$

② Column-wise all-to-all communication



→ each node collects $\sqrt{p}$ messages (each size $m\sqrt{p}$)

→ 전송 message 크기는 mp 가 된다.

→ Cost per node

$= (t_s + t_w m\sqrt{p})(\sqrt{p} - 1)$

total $= 2t_s(\sqrt{p}-1) + t_w m(p-1)$

reduction

: 전반적으로 broadcast랑 비슷한데, reduction에서 msg 크기는 1로 ?

```
my_result = local_value

for each round
  send my_result to partner
  receive msg
  my_result = my_result ⊕ msg   ← associative op.

post condition: each my_result now contains global result
```

Prefix Sum

데이터를 누적하여 합하는 것

Pre-condition	• n_k indicates that k^{th} node
Problem statement	• Compute sums $s_k = \sum_{i=0}^{k} n_i$ for $\forall$ k between 0 and p-1 → ex) <3, 1, 4, 0, 2> sequence의 Prefix sum은 <3, 4, 8, 8, 10>
Post-Condition	• n_k node contains s_k

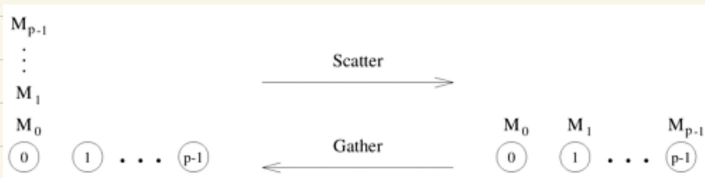
• All-to-All Reduction 사용할 수 있음

```
1.  procedure PREFIX_SUMS_HCUBE(my_id, my number, d, result)
2.  begin
3.      result := my_number;
4.      msg := result;
5.      for i := 0 to d - 1 do
6.          partner := my_id XOR 2^i;
7.          send msg to partner;
8.          receive number from partner;
9.          msg := msg + number;
10.         if (partner < my_id) then result := result + number;
11.     endfor;
12. end PREFIX_SUMS_HCUBE
```

All-to-one 로 더 많은 필요한 data만 사용하기

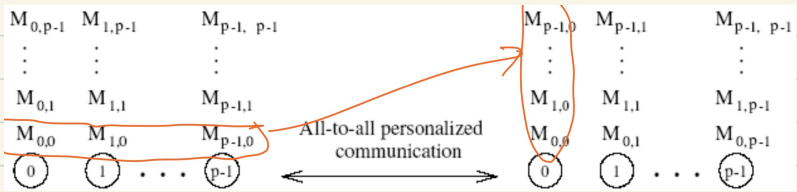
Scatter · Gather

Scatter	: node sends unique msg of size m to every other node (one-to-all personalized)
	· broadcast랑 다른점: msg 사이즈가 점점 작아진다.
	(broadcast: msg size constant) (recursive halving 이용할 수 있음)
Gather	: single node collects unique msg from each node
	· reduction은 arithmetic 이고 gather는 걍 appending임



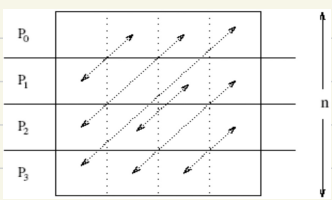
All-to-All Personalized Communication

Total exchange



: matrix transpose랑 똑같은 거

matrix transpose



- each node holds dense row of matrix
- do transpose

- ring protocol
  - each node는 send all pieces of data of size $m(p-1)$를 neighbor로 보낸다
  - recv node는 그중에 needed 1 msg만 extract 하고 $m(p-2)$를 보낸다.

    skip하면 msg size 감소

    :

    $(p-1)$ step 반복!

  - cost

$$T = \sum_{i=1}^{p-1} (t_s + t_w m(p-i))$$