Algorithms for Collective Communication

Design and Analysis of Parallel Algorithms

5DV050 Spring 2012

Outline

- One-to-all broadcast
- All-to-one reduction
- All-to-all broadcast
- All-to-all reduction
- All-reduce
- Prefix sum
- Scatter
- Gather
- All-to-all personalized
- *Improved* one-to-all broadcast
- *Improved* all-to-one reduction
- *Improved* all-reduce

Corresponding MPI functions

Operation	MPI function(s)
One-to-all broadcast	MPI_Bcast
All-to-one reduction	MPI_Reduce
All-to-all broadcast	MPI_Allgather[v]
All-to-all reduction	MPI_Reduce_scatter[_block]
All-reduce	MPI_Allreduce
Prefix sum	MPI_Scan / MPI_Exscan
Scatter	MPI_Scatter[v]
Gather	MPI_Gather[v]
All-to-all personalized	MPI_Alltoall[v|w]

Static interconnection networks

Examples of static interconnection topologies:

- Ring/linear array
- Mesh/torus
- Hypercube
- Tree

Examples of evaluation criteria:

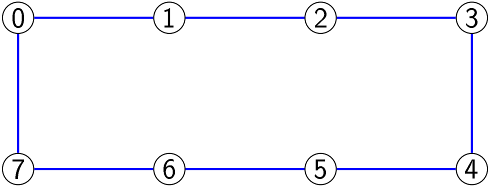
Diameter Maximum distance between any pair of processors. (Low is good.)

(Arc) Connectivity Minimum number of links that must be removed in order to disconnect at least one processor. Measures the mutliplicity of paths between nodes. (High is good.)

Bisection width Minimum number of links that must be removed to partition the network into two equal parts.

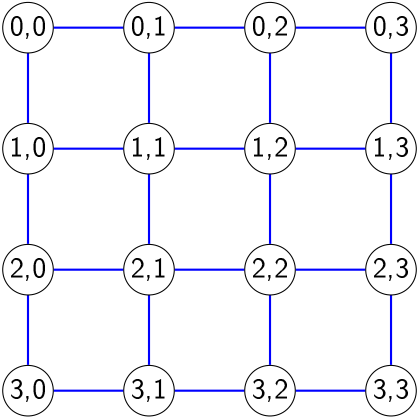
Bisection bandwidth Minimum volume of communication allowed between any two halves of the network.

Ring topology



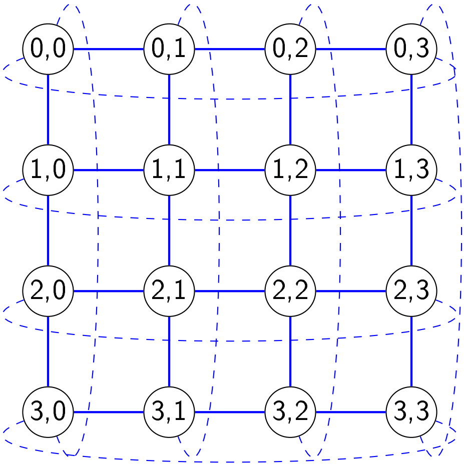
- Diameter: $\lfloor p/2 \rfloor$
- Connectivity: 2
- Bisection width: 2

(2D) Mesh topology



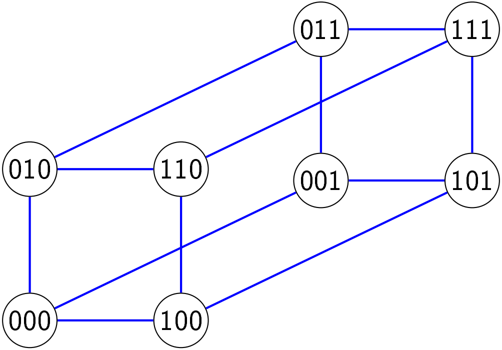
- Diameter: $2(\sqrt{p} - 1)$
- Connectivity: 2
- Bisection width: $\sqrt{p}$

(2D) Torus topology



- Diameter: $2\left\lfloor\sqrt{p}/2\right\rfloor$
- Connectivity: 4
- Bisection width: $2\sqrt{p}$

(3D) Hypercube topology



- Diameter: $\log_2 p$
- Connectivity: $\log_2 p$
- Bisection width: $p/2$

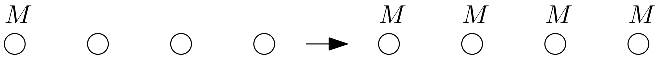
Latency/bandwidth communication model

- Point-to-point message takes time $t_s + t_w m$
- t_s is the latency
- t_w is the per-word transfer time (inverse bandwidth)
- m is the message size in # words

Contention

- Assume bi-directional links
- Each node can send and receive simultaneously
- Contention if a link is used by more than one message going in the same direction
- k-way contention is modeled by dividing the available bandwidth by k (continuous model) $t_w \mapsto t_w/k$

One-to-all broadcast



Input:

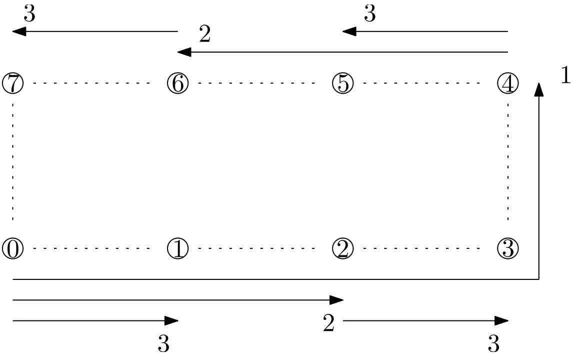
- The message M is stored locally on the root

Output:

- The message M is stored locally on all processes

One-to-all broadcast

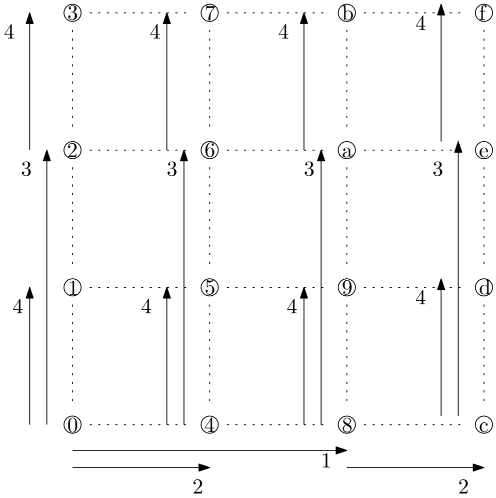
Ring



One-to-all broadcast

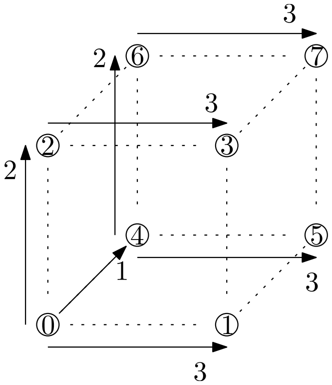
Mesh

- Use ring algorithm on the root's mesh row
- Use ring algorithm on all mesh columns in parallel



One-to-all broadcast
Hypercube

- Generalize mesh algorithm to d dimensions



One-to-all broadcast

Algorithm

The algorithms described above are identical on all three topologies

```
Assume that p = 2^d
mask ← 2^d − 1 (set all bits)
for k = d − 1, d − 2, . . . , 0 do
    mask ← mask XOR 2^k (clear bit k)
    if me AND mask = 0 then
        (lower k bits of me are 0)
        partner ← me XOR 2^k (partner has opposite bit k)
        if me AND 2^k = 0 then
            Send M to partner
        else
            Receive M from partner
        end if
    end if
end for
```

One-to-all broadcast

The preceeding broadcast algorithm is not general.

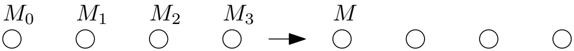
- **What if $p \neq 2^d$?**
  - Set $d = \lceil \log_2 p \rceil$ and don't communicate if $\texttt{partner} \geq p$
- **What if the root is not process 0?**
  - Relabel the processes: $\texttt{me} \rightarrow \texttt{me}\,\mathrm{XOR}\,\texttt{root}$
- Can we use both fixes simultaneously or not?

One-to-all broadcast

Runtime

- Number of steps: $d = \log_2 p$
- Time per step: $t_s + t_w m$
- Total time: $(t_s + t_w m) \log_2 p$
- In particular, note that broadcasting to p^2 processes is only twice as expensive as broadcasting to p processes since $\log_2 p^2 = 2 \log_2 p$

All-to-one reduction



$$M := M_0 \oplus M_1 \oplus M_2 \oplus M_3$$

Input:

- The p messages M_k for $k = 0, 1, \ldots, p-1$
- The message M_k is stored locally on process k
- An associative reduction operator $\oplus$
- Typically, $\oplus \in \{+, \times, \max, \min\}$

Output:

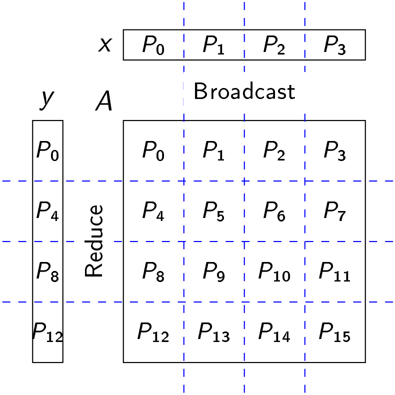
- The "sum" $M := M_0 \oplus M_1 \oplus \cdots \oplus M_{p-1}$ stored on the root

All-to-one reduction
Algorithm

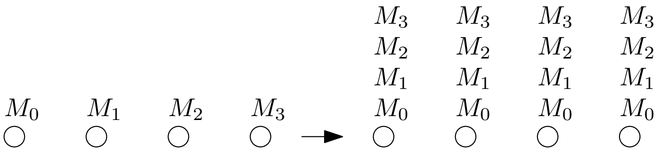
- Analogous to all-to-one broadcast algorithm
- Analogous runtime (plus the, often negligible, time to compute $a \oplus b$)
- Reverse order of communications
- Reverse direction of communications
- Combine incoming message with local message using $\oplus$

Matrix–vector multiplication

```
for i = 0, 1, ..., n − 1 do
    y(i) ← 0
    for k = 0, 1, ..., n − 1 do
        y(i) ← y(i) + A(i, k) ∗ x(k)
    end for
end for
```



All-to-all broadcast



Input:

- The p messages M_k for $k = 0, 1, \ldots, p-1$
- The message M_k is stored locally on process k

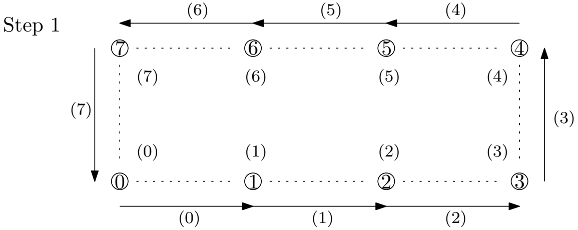
Output:

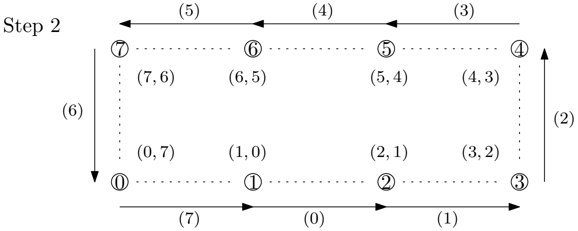
- The p messages M_k for $k = 0, 1, \ldots, p-1$ are stored locally on all processes

Equivalent to p concurrent one-to-all broadcasts

All-to-all broadcast

Ring





and so on...

All-to-all broadcast

Ring algorithm

```
1: left ← (me − 1) mod p
2: right ← (me + 1) mod p
3: result ← M_me
4: M ← result
5: for k = 1, 2, ..., p − 1 do
6:     Send M to right
7:     Receive M from left
8:     result ← result ∪ M
9: end for
```

- The "send" is assumed to be non-blocking
- Lines 6–7 can be implemented via MPI_Sendrecv

All-to-all broadcast

Run time (ring)

- Number of steps: $p - 1$
- Time per step: $t_s + t_w m$
- Total time: $(p - 1)(t_s + t_w m)$

All-to-all broadcast

Mesh algorithm

The **mesh** algorithm is based on the **ring** algorithm:

- Apply the **ring** algorithm to all mesh rows in parallel
- Apply the **ring** algorithm to all mesh columns in parallel

All-to-all broadcast

Run time (mesh)

(Assuming a $\sqrt{p} \times \sqrt{p}$ mesh for simplicity)

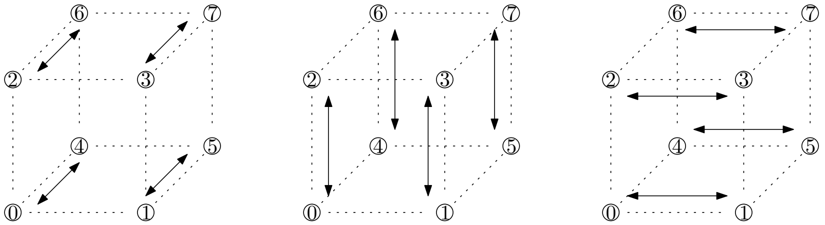
- Apply the **ring** algorithm to all mesh rows in parallel
  - Number of steps: $\sqrt{p} - 1$
  - Time per step: $t_s + t_w m$
  - Total time: $(\sqrt{p} - 1)(t_s + t_w m)$
- Apply the **ring** algorithm to all mesh columns in parallel
  - Number of steps: $\sqrt{p} - 1$
  - Time per step: $t_s + t_w \sqrt{p} m$
  - Total time: $(\sqrt{p} - 1)(t_s + t_w \sqrt{p} m)$
- Total time: $2(\sqrt{p} - 1)t_s + (p - 1)t_w m$

All-to-all broadcast

Hypercube algorithm

The **hypercube** algorithm is also based on the **ring** algorithm:

- For each dimension k of the hypercube in sequence:
- Apply the **ring** algorithm (which reduces to a pairwise exchange) to the 2^{d-1} links in the current dimension in parallel



All-to-all broadcast

Run time (hypercube)

- Number of steps: $d = \log_2 p$
- Time for step $k = 0, 1, \ldots, d-1$: $t_s + t_w 2^k m$
- Total time: $\sum_{k=0}^{d-1}(t_s + t_w 2^k m) = t_s \log_2 p + t_w (p-1) m$

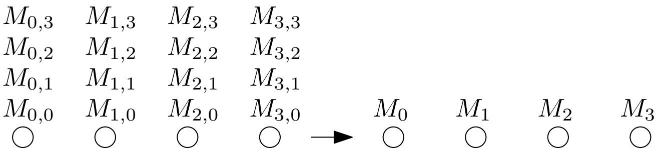
All-to-all broadcast

Summary

Topology	t_s	t_w
Ring	$p-1$	$(p-1)m$
Mesh	$2(\sqrt{p}-1)$	$(p-1)m$
Hypercube	$\log_2 p$	$(p-1)m$

- Note 1: Same transfer time (t_w term)
- Note 2: The t_w term is optimal since each process must receive $(p-1)m$ words

All-to-all reduction



$$M_r := M_{0,r} \oplus M_{1,r} \oplus M_{2,r} \oplus M_{3,r}$$

Input:

- The p^2 messages $M_{r,k}$ for $r, k = 0, 1, \ldots, p-1$
- The message $M_{r,k}$ is stored locally on process r
- An associative reduction operator $\oplus$

Output:

- The "sum" $M_r := M_{0,r} \oplus M_{1,r} \oplus \cdots \oplus M_{p-1,r}$ stored locally on each process r

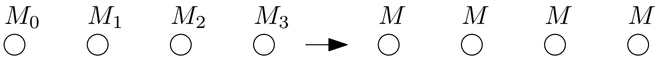
Equivalent to p concurrent all-to-one reductions

All-to-all reduction

Algorithm

- Analogous to all-to-all broadcast algorithm
- Analogous time (plus the time, which is often negligible, for computing $a \oplus b$)
- Reverse order of communications
- Reverse direction of communications
- Combine incoming message with a subset of the local message using $\oplus$

All-reduce



$$M := M_0 \oplus M_1 \oplus M_2 \oplus M_3$$

Input:

- The p messages M_k for $k = 0, 1, \ldots, p-1$
- The message M_k is stored locally on process k
- An associative reduction operator $\oplus$

Output:

- The "sum" $M := M_0 \oplus M_1 \oplus \cdots \oplus M_{p-1}$ stored locally on all processes

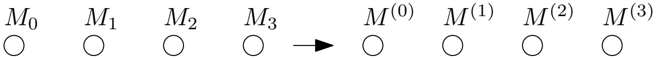
Equivalent to one-to-all reduction followed by one-to-all broadcast

All-reduce

Algorithm

- Analogous to all-to-all broadcast algorithm
- Combine incoming message with local message using $\oplus$
- Cheaper since the message size does not grow
- Total time: $(t_s + t_w m) \log_2 p$

Prefix sum



$$M^{(k)} := M_0 \oplus M_1 \oplus \cdots \oplus M_k$$

Input:

- The p messages M_k for $k = 0, 1, \ldots, p-1$
- The message M_k is stored locally on process k
- An associative reduction operator $\oplus$

Output:

- The "sum" $M^{(k)} := M_0 \oplus M_1 \oplus \cdots \oplus M_k$ stored locally on process k for all k

Prefix sum: Example

Input

1	2	3	4	5	6	7	8	9
1	3	6	10	15	21	28	36	45

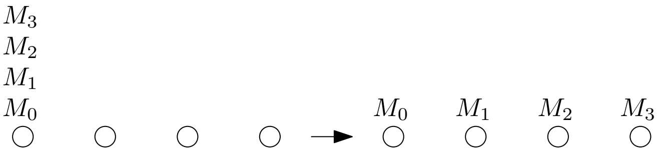
Output

Prefix sum

Algorithm

- Analogous to all-reduce algorithm
- Analogous time
- Locally store only the corresponding partial sum

Scatter



Input:

- The p messages M_k for $k = 0, 1, \ldots, p-1$ stored locally on the root

Output:

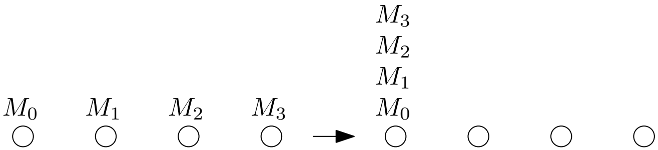
- The message M_k stored locally on process k for all k

Scatter

Algorithm

- Analogous to one-to-all broadcast algorithm
- Send half of the messages in the first step, send one quarter in the second step, and so on
- More expensive since several messages are sent in each step
- Total time: $t_s \log_2 p + t_w(p-1)m$

Gather



Input:

- The p messages M_k for $k = 0, 1, \ldots, p-1$
- The message M_k is stored locally on process k

Output:

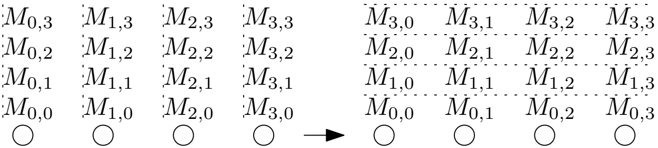
- The p messages M_k stored locally on the root

Gather

Algorithm

- Analogous to scatter algorithm
- Analogous time
- Reverse the order of communications
- Reverse the direction of communications

All-to-all personalized



Input:

- The p^2 messages $M_{r,k}$ for $r, k = 0, 1, \ldots, p-1$
- The message $M_{r,k}$ is stored locally on process r

Output:

- The p messages $M_{r,k}$ stored locally on process k for all k

All-to-all personalized

Summary

Topology	t_s	t_w
Ring	$p-1$	$(p-1)mp/2$
Mesh	$2(\sqrt{p}-1)$	$p(\sqrt{p}-1)m$
Hypercube	$\log_2 p$	$m(p/2)\log_2 p$

- The hypercube algorithm is not optimal with respect to communication volume (the lower bound is $t_w m(p-1)$)

All-to-all personalized

An optimal (w.r.t. volume) hypercube algorithm

Idea:

- Let each pair of processes exchange messages directly

Run time:

- $(p-1)(t_s + t_w m)$

Question:

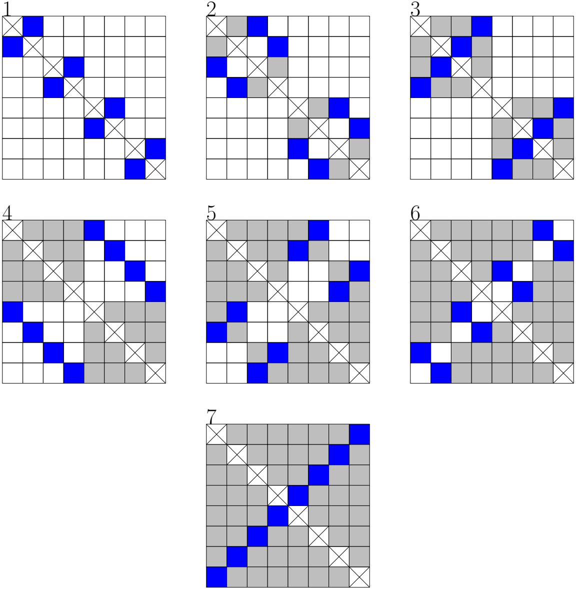
- In which order do we pair the processes?

Answer:

- In step k, let `me` exchange messages with `me` XOR k
- Amazingly, the messages can be routed without contention!

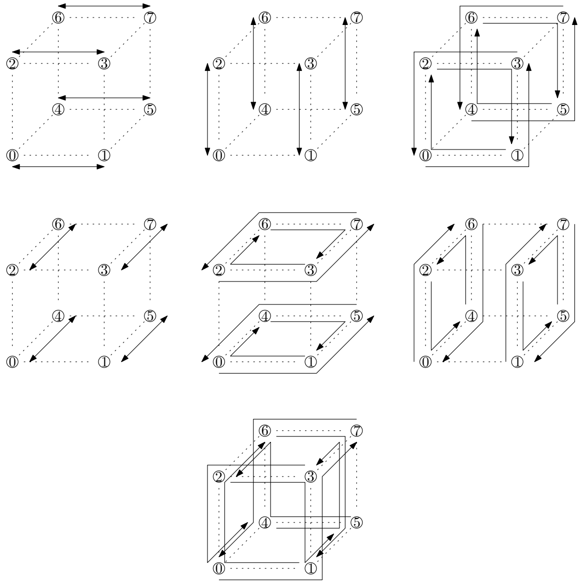
All-to-all personalized: Optimal algorithm

Communication pattern for $p = 8$



All-to-all personalized

An optimal hypercube algorithm based on E-cube routing



All-to-all personalized

E-cube routing

- Routing from s to $t := s \, \mathrm{XOR} \, k$ in step k
- The difference between s and t is

$$s \, \mathrm{XOR} \, t = s \, \mathrm{XOR}(s \, \mathrm{XOR} \, k) = k$$

- The number of links to traverse equals the number of 1's in the binary representation of k (the so-called Hamming distance)
- E-cube routing: route through the links according to some arbitrary (but fixed) ordering of the dimensions

All-to-all personalized

E-cube routing

Why does E-cube routing work?

- Write

$$k = k_1 \text{ XOR } k_2 \text{ XOR} \cdots \text{XOR } k_n$$

  such that

  - k_j has exactly one set bit
  - $k_i \neq k_j$ for all $i \neq j$

- Step i:

$$r \mapsto r \text{ XOR } k_i$$

  and hence uses the links in one dimension without congestion.

- After all n steps we have as desired:

$$s \mapsto s \text{ XOR } k_1 \text{ XOR} \cdots \text{XOR } k_n = s \text{ XOR } k = t$$

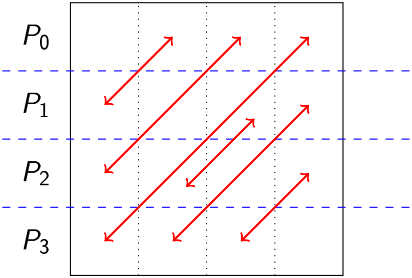
All-to-all personalized

E-cube routing example

- Route from $s = 100_2$ to $t = 001_2 = s \,\mathrm{XOR}\, 101_2$
- Hamming distance (i.e., # links): 2
- Write
$$k = 101_2 = 001_2 \,\mathrm{XOR}\, 100_2 = k_1 \,\mathrm{XOR}\, k_2$$
- E-cube route:
$$s = 100_2 \rightarrow 101_2 \rightarrow 001_2 = t$$

Matrix transposition

```
for i = 0, 1, ..., n − 1 do
    for k = 0, 1, ..., n − 1 do
        B(k, i) ← A(i, k)
    end for
end for
```



Maps to an all-to-all personalized operation

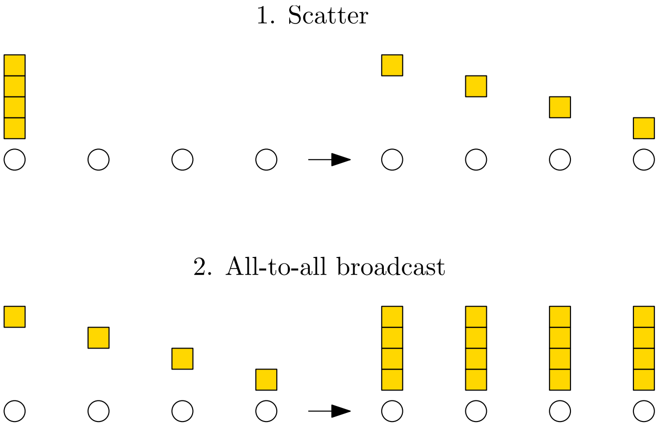
Summary

Hypercube

Operation	Time
One-to-all broadcast	$(t_s + t_w m) \log_2 p$
All-to-one reduction	$(t_s + t_w m) \log_2 p$
All-reduce	$(t_s + t_w m) \log_2 p$
Prefix sum	$(t_s + t_w m) \log_2 p$
All-to-all broadcast	$t_s \log_2 p + t_w (p-1) m$
All-to-all reduction	$t_s \log_2 p + t_w (p-1) m$
Scatter	$t_s \log_2 p + t_w (p-1) m$
Gather	$t_s \log_2 p + t_w (p-1) m$
All-to-all personalized	$(t_s + t_w m)(p-1)$

Improved one-to-all broadcast

Applies when $m \gg p$



Improved one-to-all broadcast

Run time

Old algorithm:

- Total time: $(t_s + t_w m) \log_2 p$

New algorithm:

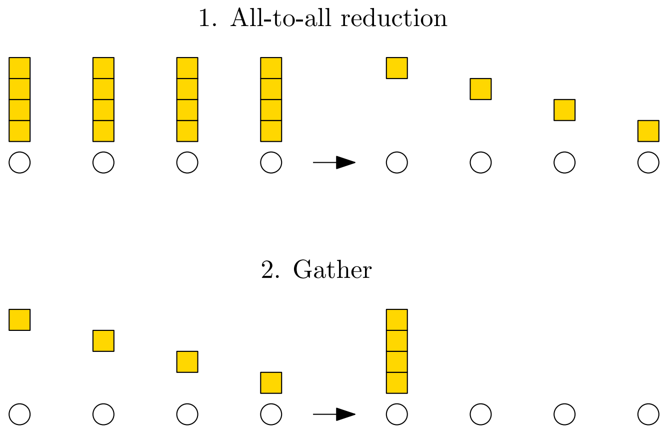
- Scatter: $t_s \log_2 p + t_w (p-1)(m/p)$
- All-to-all broadcast: $t_s \log_2 p + t_w (p-1)(m/p)$
- Total time: $2t_s \log_2 p + 2t_w (p-1)(m/p) \approx 2t_s \log_2 p + 2t_w m$

Comparison with previous algorithm:

- t_s term: twice as large
- t_w term: reduced by a factor $\approx (\log_2 p)/2$

Improved all-to-one reduction

Applies when $m \gg p$



Improved all-to-one reduction

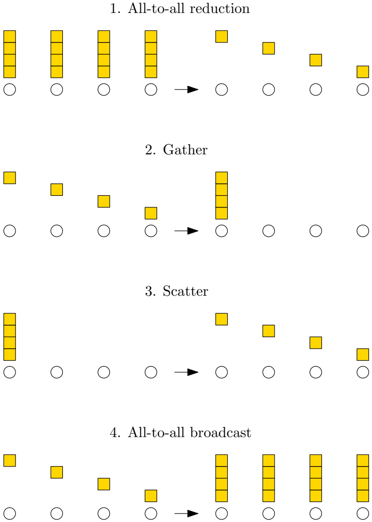
Run time

- Analogous to improved one-to-all broadcast
- t_s term: twice as large
- t_w term: reduced by a factor $\approx (\log_2 p)/2$

Improved all-reduce

Applies when $m \gg p$

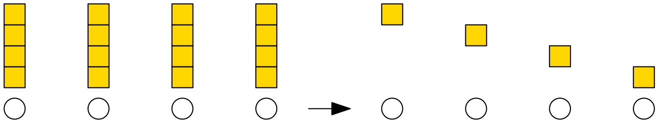
All-reduce = One-to-all reduction + All-to-one broadcast



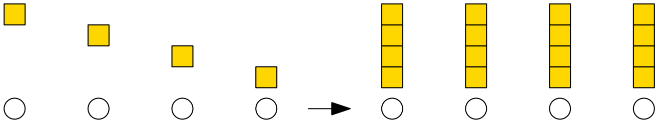
...but gather followed by scatter cancel out!

Improved all-reduce

1. All-to-all reduction



2. All-to-all broadcast



Improved all-reduce

Run time

- Analogous to improved one-to-all broadcast
- t_s term: twice as large
- t_w term: reduced by a factor $\approx (\log_2 p)/2$