Algorithms for Collective Communication

Design and Analysis of Parallel Algorithms

Source

- A. Grama, A. Gupta, G. Karypis, and V. Kumar. Introduction to Parallel Computing, Chapter 4, 2003.

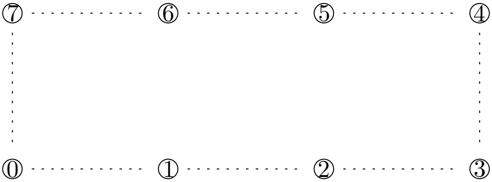
Outline

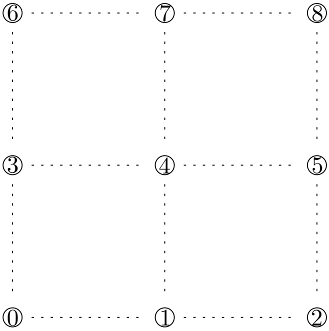
- One-to-all broadcast
- All-to-one reduction
- All-to-all broadcast
- All-to-all reduction
- All-reduce
- Prefix sum
- Scatter
- Gather
- All-to-all personalized
- Improved one-to-all broadcast
- Improved all-to-one reduction
- Improved all-reduce

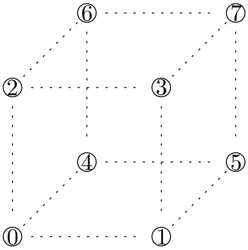
Corresponding MPI functions

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]`

Topologies







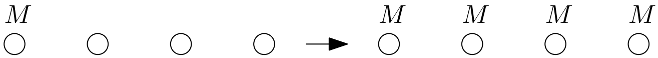
Linear model of communication overhead

- 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
- (Must use compatible units for m and t_w)

Contention

- Assuming bi-directional links
- Each node can send and receive simultaneously
- Contention if link is used by more than one message
- k-way contention means $t_w \rightarrow t_w/k$

One-to-all broadcast



Input:

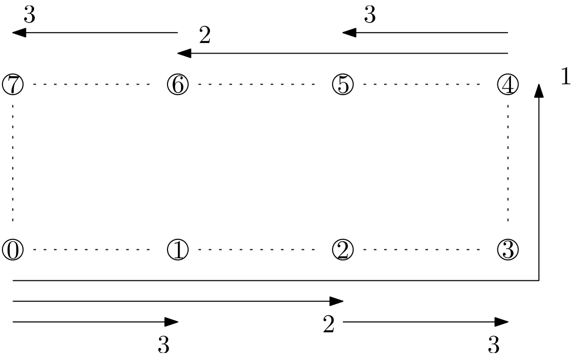
- The message M is stored locally on the root

Output:

- The message M is stored locally on all processes

One-to-all broadcast

Ring

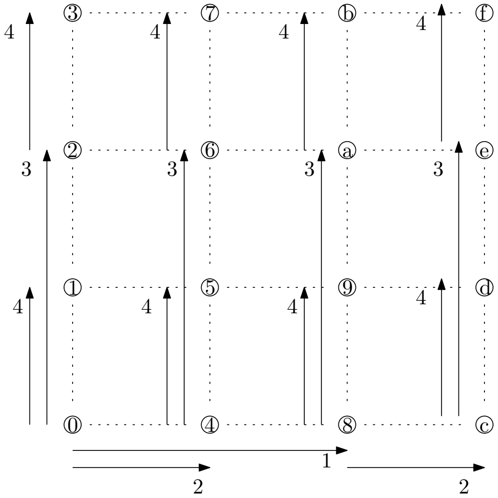


- Recursive doubling
- Double the number of active processes in each step

One-to-all broadcast

Mesh

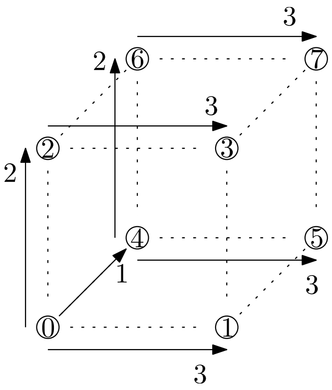
- Use ring algorithm on the root's mesh row
- Use ring algorithm on all mesh columns in parallel



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` $\leftarrow 2^d - 1$ (set all bits)
for $k = d-1, d-2, \ldots, 0$ **do**
    `mask` $\leftarrow$ `mask` XOR 2^k (clear bit k)
    if `me` AND `mask` $= 0$ **then**
        (lower k bits of `me` are 0)
        `partner` $\leftarrow$ `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

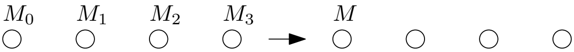
The given algorithm is not general.

- **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: $\texttt{me} \rightarrow \texttt{me}\,\mathrm{XOR}\,\texttt{root}$

One-to-all broadcast

- 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 ($\log_2 p^2 = 2 \log_2 p$)

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 $\oplus$
- E.g., $\oplus \in \{+, \times, \max, \min\}$

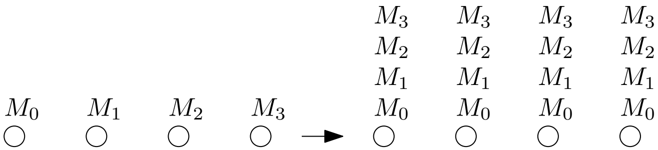
Output:

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

All-to-one reduction
Algorithm

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

All-to-all broadcast



Input:

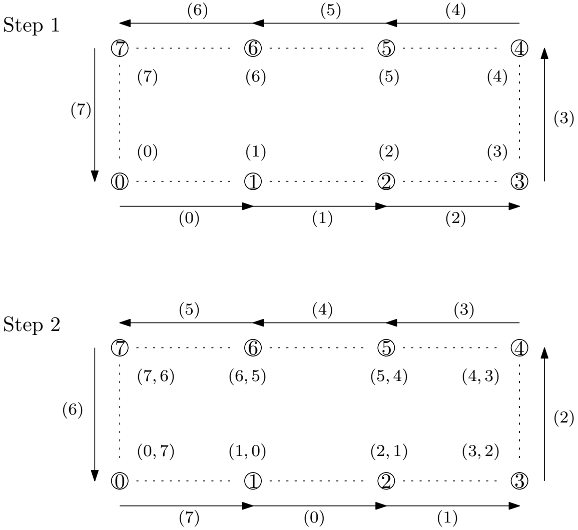
- The p messages M_k for $k = 0, 1, \ldots, p-1$
- The message M_k is stored locally on process k

Output:

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

All-to-all broadcast

Ring



and so on...

All-to-all broadcast

Ring algorithm

1: $\texttt{left} \leftarrow (\texttt{me} - 1) \bmod p$
2: $\texttt{right} \leftarrow (\texttt{me} + 1) \bmod p$
3: $\texttt{result} \leftarrow M_{\texttt{me}}$
4: $M \leftarrow \texttt{result}$
5: **for** $k = 1, 2, \ldots, p-1$ **do**
6:     Send M to `right`
7:     Receive M from `left`
8:     $\texttt{result} \leftarrow \texttt{result} \cup M$
9: **end for**

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

All-to-all broadcast

Time of ring algorithm

- 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

Time of mesh algorithm

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

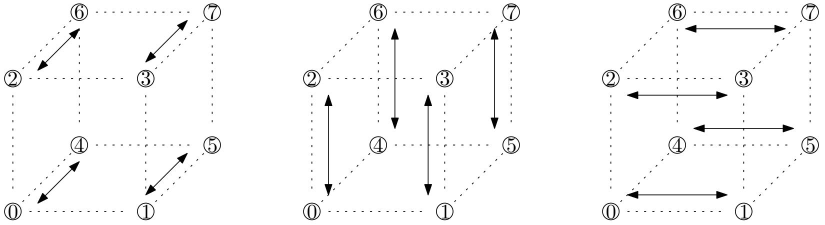
- Total time: $2(\sqrt{p} - 1)t_s + (p - 1)t_w m$

All-to-all broadcast

Hypercube algorithm

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

- For each dimension d of the hypercube in sequence:
- Apply the **ring** algorithm to the 2^{d-1} links in the current dimension in parallel



All-to-all broadcast

Time of hypercube algorithm

- 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	$\log_2 p$	$(p-1)m$

- Same transfer time (t_w term)
- But the number of messages differ

All-to-all reduction

$$
\begin{array}{cccc}
M_{0,3} & M_{1,3} & M_{2,3} & M_{3,3} \\
M_{0,2} & M_{1,2} & M_{2,2} & M_{3,2} \\
M_{0,1} & M_{1,1} & M_{2,1} & M_{3,1} \\
M_{0,0} & M_{1,0} & M_{2,0} & M_{3,0}
\end{array}
\longrightarrow
\begin{array}{cccc}
M_0 & M_1 & M_2 & M_3
\end{array}
$$

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

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:

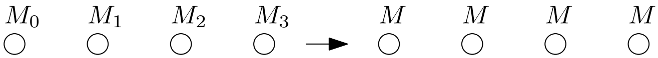
- The "sum" $M_r := M_{0,r} \oplus M_{1,r} \oplus \cdots \oplus M_{p-1,r}$ stored locally on each process r

All-to-all reduction

Algorithm

- Analogous to all-to-all broadcast algorithm
- Analogous time (plus the time for computing $a \oplus b$)
- Reverse order of communications
- Reverse direction of communications
- Combine incoming message with part of 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:

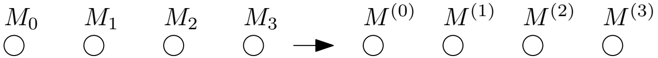
- The "sum" $M := M_0 \oplus M_1 \oplus \cdots \oplus M_{p-1}$ stored locally on all processes

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:

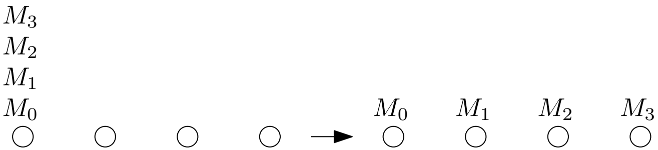
- The "sum" $M^{(k)} := M_0 \oplus M_1 \oplus \cdots \oplus M_k$ stored locally on process k for all k

Prefix sum

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:

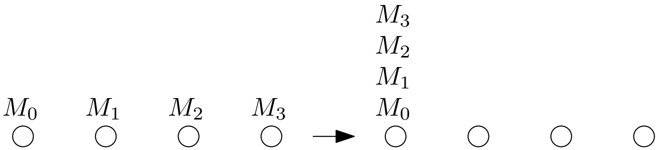
- The message M_k stored locally on process k for all k

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:

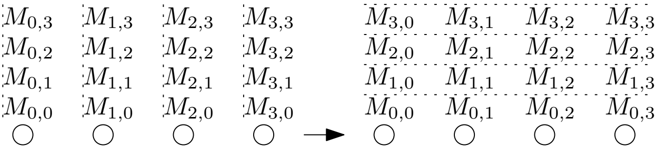
- The p messages M_k stored locally on the root

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

Time:

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

Q:

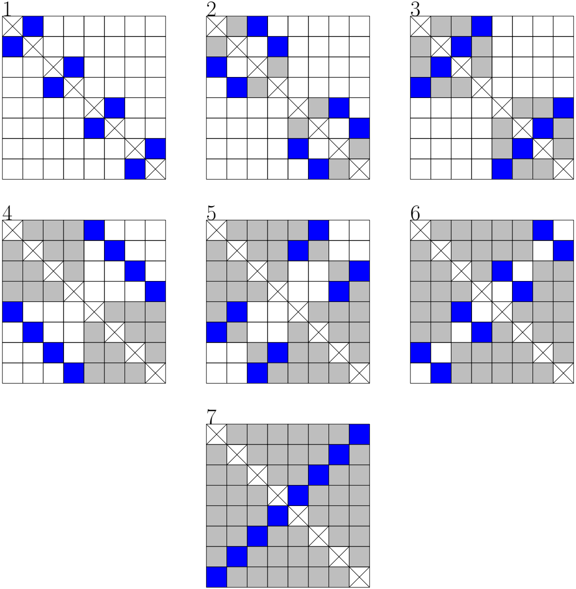
- In which order do we pair the processes?

A:

- In step k, let `me` exchange messages with `me` XOR k
- This can be done without contention!

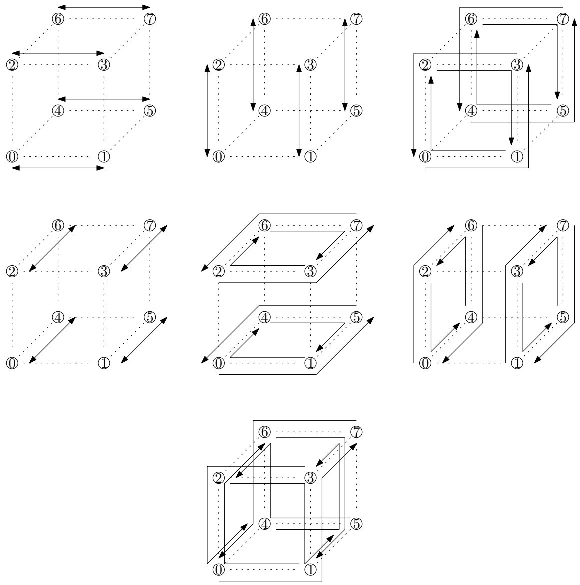
All-to-all personalized

An optimal hypercube algorithm



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 fixed (arbitrary) ordering imposed on 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:
$$r \mapsto r \text{ XOR } k_1 \text{ XOR} \cdots \text{XOR } k_n = r \text{ XOR } k$$

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
$$k = k_1 \,\mathrm{XOR}\, k_2 = 001_2 \,\mathrm{XOR}\, 100_2$$
- E-cube route:
$$t = 100_2 \rightarrow 10\textcolor{red}{1}_2 \rightarrow \textcolor{red}{0}01_2 = s$$

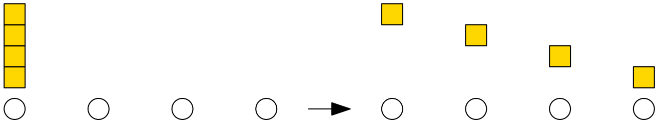
Summary

Hypercube

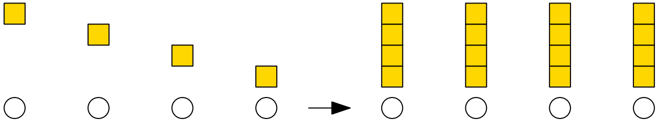
Operation	Time
One-to-all broadcast	$(t_s + t_w m)\log_2 p$
All-to-one reduction	$(t_s + t_w m)\log_2 p$
All-reduce	$(t_s + t_w m)\log_2 p$
Prefix sum	$(t_s + t_w m)\log_2 p$
All-to-all broadcast	$t_s \log_2 p + t_w(p-1)m$
All-to-all reduction	$t_s \log_2 p + t_w(p-1)m$
Scatter	$t_s \log_2 p + t_w(p-1)m$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

1. Scatter



2. All-to-all broadcast



Improved one-to-all broadcast

Time analysis

Old algorithm:

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

New algorithm:

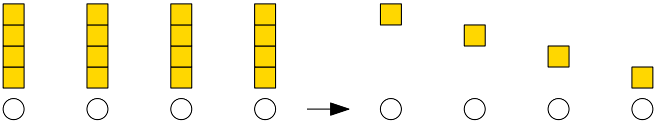
- Scatter: $t_s \log_2 p + t_w(p-1)(m/p)$
- All-to-all broadcast: $t_s \log_2 p + t_w(p-1)(m/p)$
- Total time: $2t_s \log_2 p + 2t_w(p-1)(m/p) \approx 2t_s \log_2 p + 2t_w m$

Effect:

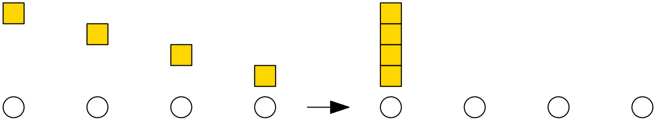
- t_s term: twice as large
- t_w term: reduced by a factor $\approx (\log_2 p)/2$

Improved all-to-one reduction

1. All-to-all reduction



2. Gather



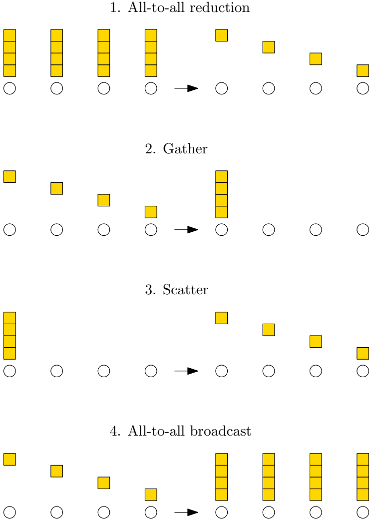
Improved all-to-one reduction

Time analysis

- 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

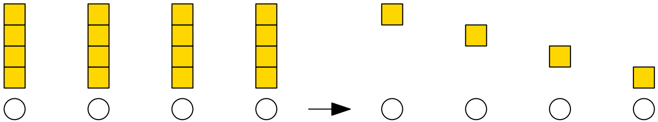
All-reduce = One-to-all reduction + All-to-one broadcast



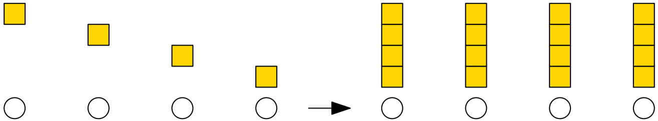
...but gather followed by scatter cancel out!

Improved all-reduce

1. All-to-all reduction



2. All-to-all broadcast



Improved all-reduce

Time analysis

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