Collective Communication

John Mellor-Crummey

Department of Computer Science
Rice University

johnmc@rice.edu

Group Communication

- **Motivation: accelerate interaction patterns within a group**
- **Approach: collective communication**
  - —group works together *collectively* to realize a communication
  - —constructed from pairwise point-to-point communications
- **Implementation strategy**
  - —standard library of common collective operations
  - —leverage target architecture for efficient implementation
- **Benefits of standard library implementations**
  - —reduce development effort and cost for parallel programs
  - —improve performance through efficient implementations
  - —improve quality of scientific applications

Topics for Today

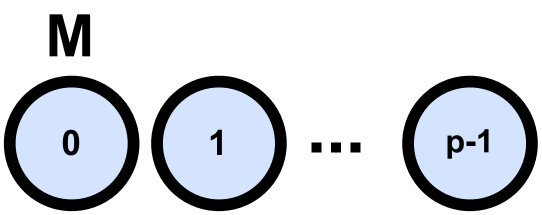
- One-to-all broadcast and all-to-one reduction
- All-to-all broadcast and reduction
- All-reduce and prefix-sum operations
- Scatter and gather
- All-to-all personalized communication
- Optimizing collective patterns

Assumptions

- Network is bidirectional
- Communication is single-ported
  - —node can receive only one message per step
- Communication cost model
  - —message of size *m,* no congestion, time = $t_s + t_w m$
  - —congestion: model by scaling t_w

One-to-All and All-to-One

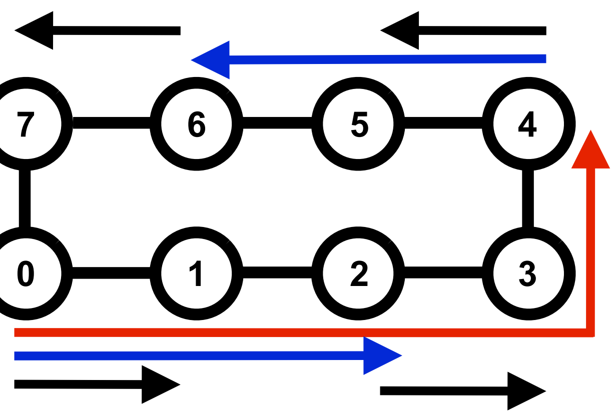
- One-to-all broadcast
  - —a processor has *M* units of data that it must send to everyone
- All-to-one reduction
  - —each processor has *M* units of data
  - —data items must be combined using some associative operator
    - – e.g. addition, min, max
  - —result must be available at a target processor



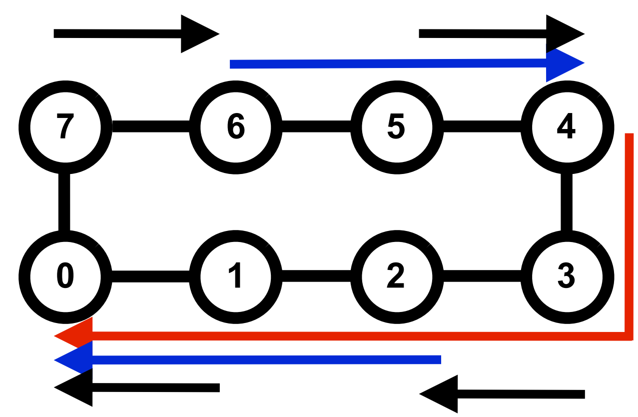
One-to-All and All-to-One on a Ring

- **Broadcast**
  - —naïve solution
    - – source sends send *p - 1* messages to the other *p - 1* processors
  - —use recursive doubling
    - – source sends a message to a selected processor

      yields two independent problems over halves of the machine

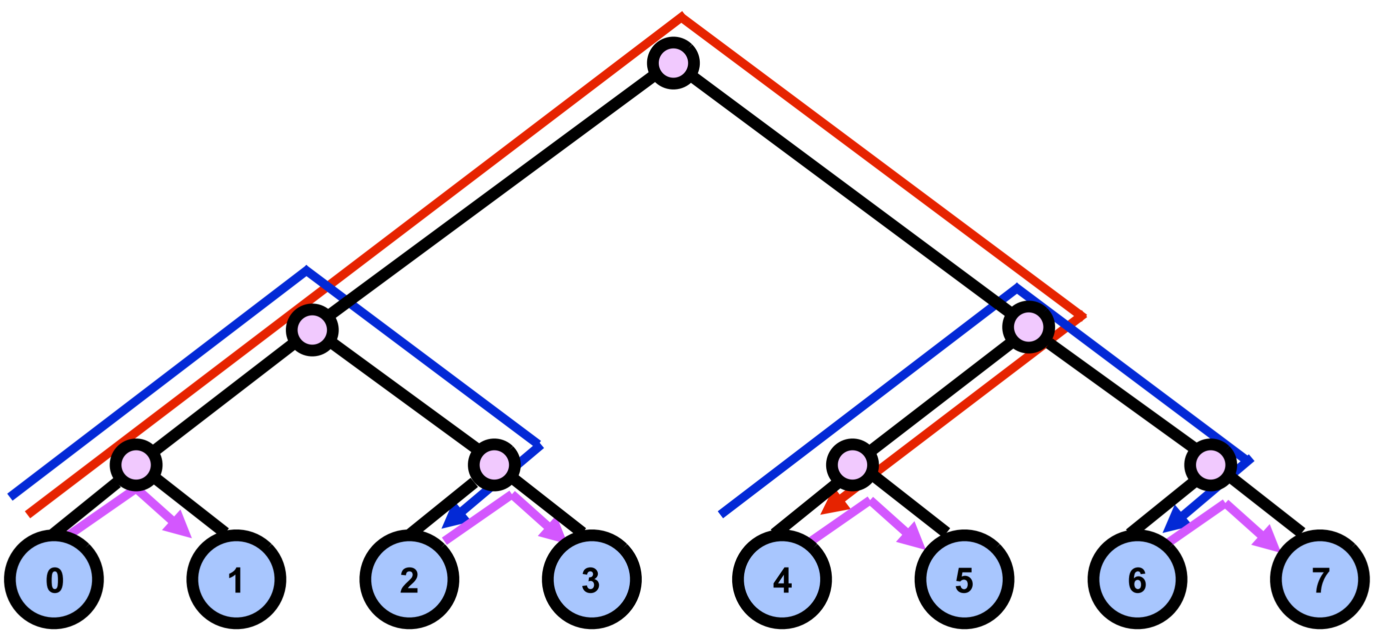


- **Reduction**
  - — invert the process



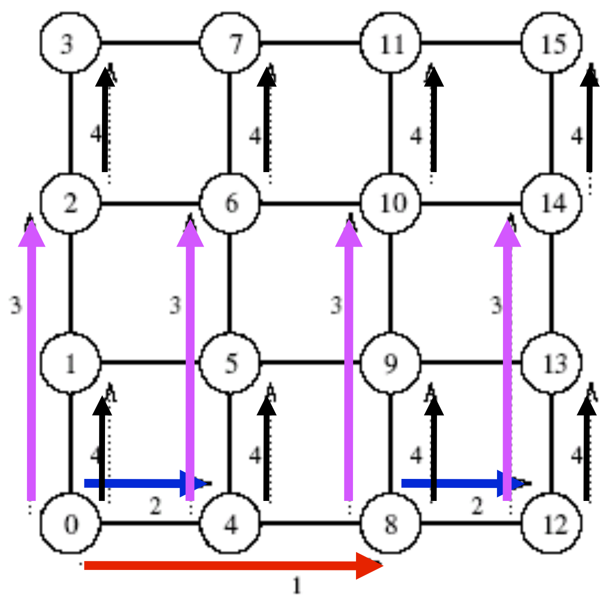
Broadcast on a Balanced Binary Tree

- **Consider processors arranged in a dynamic binary tree**
  - **—processors are at the leaves**
  - **—interior nodes are switches**
- **Assume leftmost processor is the root of the broadcast**
- **Use recursive doubling strategy: log *p* stages**



Broadcast and Reduction on a 2D Mesh

- **Consider a square mesh of p nodes**
  - — treat each row as a linear array of $p^{1/2}$ nodes
  - — treat each column as a linear array of $p^{1/2}$ nodes
- **Two step broadcast and reduction operations**
  1. perform the operation along a row
  2. perform the operation along each column concurrently

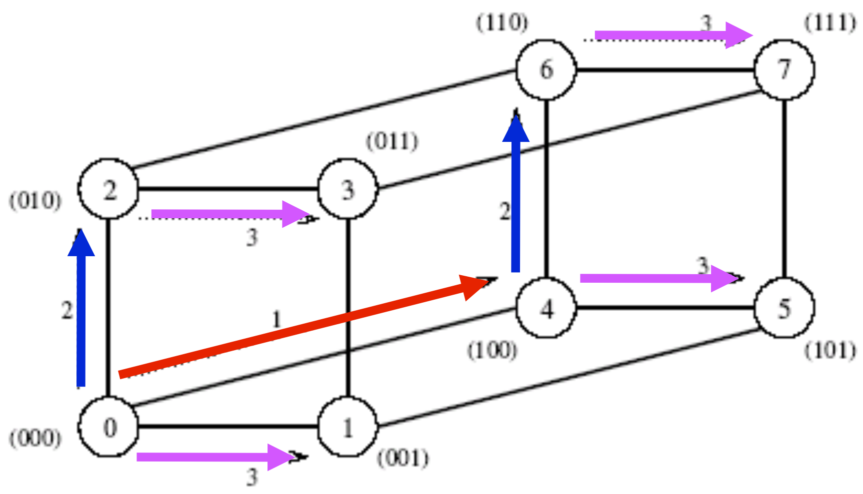


broadcast on 4 x 4 mesh

- **Generalizes to higher dimensional meshes**

Broadcast and Reduction on a Hypercube

- **Consider hypercube with *2^d* nodes**
  - **—view as *d*-dimensional mesh with two nodes in each dimension**
- **Apply mesh algorithm to a hypercube**
  - **—*d* (= *log p*) steps**



Broadcast and Reduction Algorithms

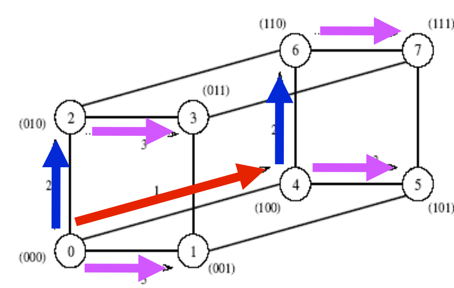
- **Each of aforementioned broadcast/reduction algorithms**
  —**adaptation of the same algorithmic template**
- **Next slide: a broadcast algorithm for a hypercube of 2^d nodes**
  —**can be adapted to other architectures**
  —**in the following algorithm**
  - ***my_id* is the label for a node**
  - ***X* is the message to be broadcast**

One-to-All Broadcast Algorithm

```
procedure GENERAL_ONE_TO_ALL_BC(d, my_id, source, X)
begin
    my_virtual_id := my_id XOR source;
    mask := 2^d - 1;
    for i := d - 1 downto 0 do          /* Outer loop */
        mask := mask XOR 2^i;           /* Set bit i of mask to 0 */
        if (my_virtual_id AND mask) = 0 then
            if (my_virtual_id AND 2^i) = 0 then        // even
                virtual_dest := my_virtual_id XOR 2^i;
                send X to (virtual_dest XOR source);
    /* Convert virtual_dest to the label of the physical destination */
            else                                        // odd
                virtual_source := my_virtual_id XOR 2^i;
                receive X from (virtual_source XOR source);
    /* Convert virtual_source to the label of the physical source */
            endelse;
    endfor;
end GENERAL_ONE_TO_ALL_BC
```

position relative to source (annotation on `my_virtual_id := my_id XOR source;`)

I am communicating on behalf of a 2^i subcube (annotation on `if (my_virtual_id AND mask) = 0 then`)



One-to-all broadcast of a message ***X*** from ***source*** on a hypercube

All-to-One Reduction Algorithm

```
procedure ALL_TO_ONE_REDUCE(d, my_id, m, X, sum)
begin
    for j := 0 to m - 1 do sum[j] := X[j];
    mask := 0;
    for i := 0 to d - 1 do
        /* Select nodes whose lower i bits are 0 */
        if (my_id AND mask) = 0 then
            if (my_id AND 2^i) ≠ 0 then      // odd
                msg_destination := my_id XOR 2^i;
                send sum to msg_destination;
            else                              // even
                msg_source := my_id XOR 2^i;
                receive X from msg_source;
                for j := 0 to m - 1 do
                    sum[j] := sum[j] + X[j];
            endelse;
        mask := mask XOR 2^i;    /* Set bit i of mask to 1 */
    endfor;
end ALL_TO_ONE_REDUCE
```

I am communicating on behalf of a 2^i subcube (refers to line: if (my_id AND mask) = 0 then)

All-to-One sum reduction on a *d*-dimensional hypercube

Each node contributes msg *X* containing *m* words, and node 0 is the destination

Broadcast/Reduction Cost Analysis

Hypercube

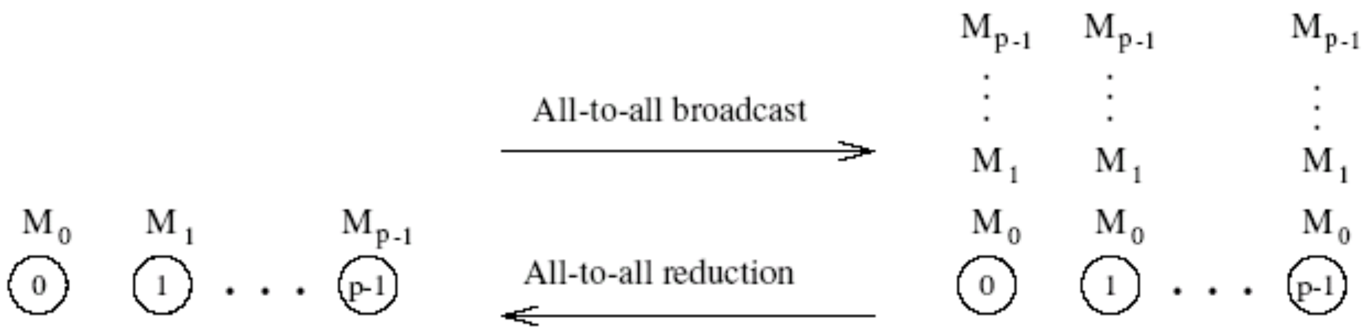
- **Log p point-to-point simple message transfers**
  - **each message transfer time: $t_s + t_w m$**
- **Total time**

$$T = (t_s + t_w m) \log p.$$

All-to-All Broadcast and Reduction

Each processor is the source as well as destination

- **Broadcast**
  - —each process broadcasts its own *m*-word message all others
- **Reduction**
  - —each process gets a copy of the result



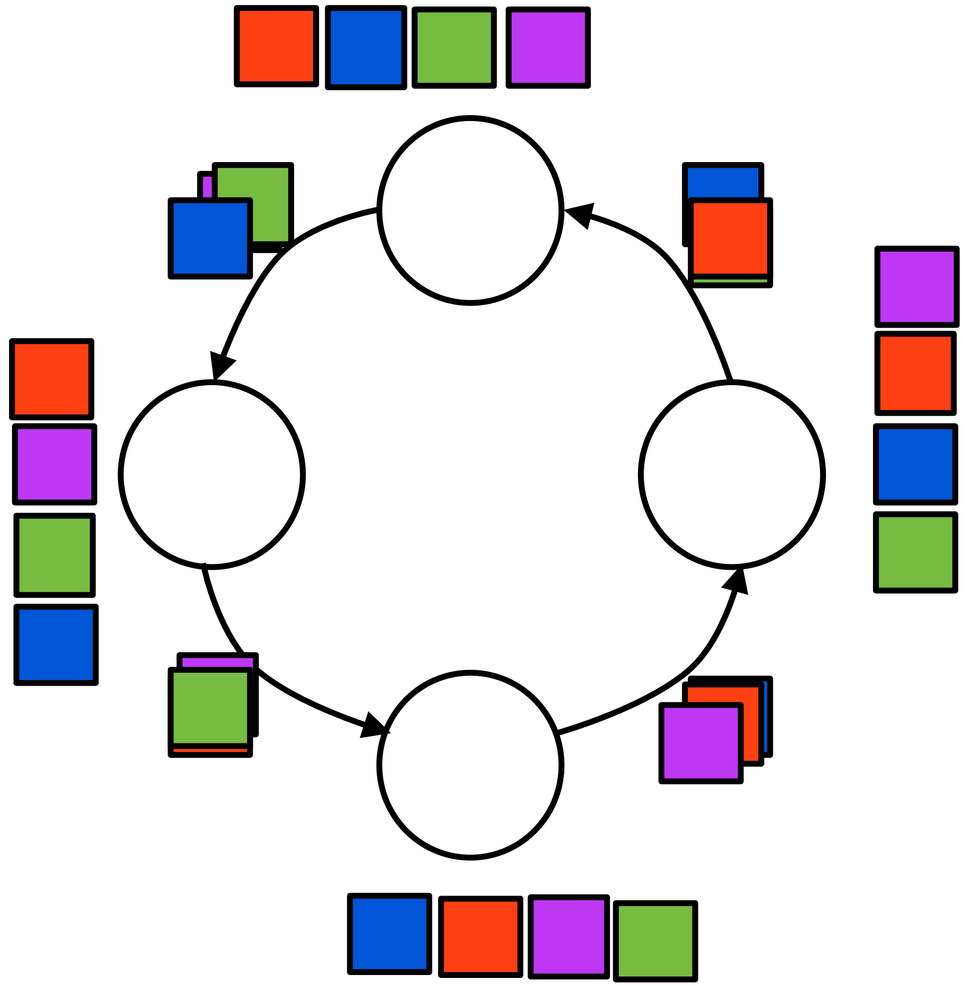
All-to-All Broadcast/Reduction on a Ring

```
procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
begin
    left := (my_id − 1) mod p;
    right := (my_id + 1) mod p;
    result := my_msg;
    msg := result;
    for i := 1 to p − 1 do
        send msg to right;
        receive msg from left;
        result := result ∪ msg;
    endfor;
end ALL_TO_ALL_BC_RING
```

message size stays constant

Also works for a linear array with bidirectional communication channels

All-to-All Broadcast on a Ring



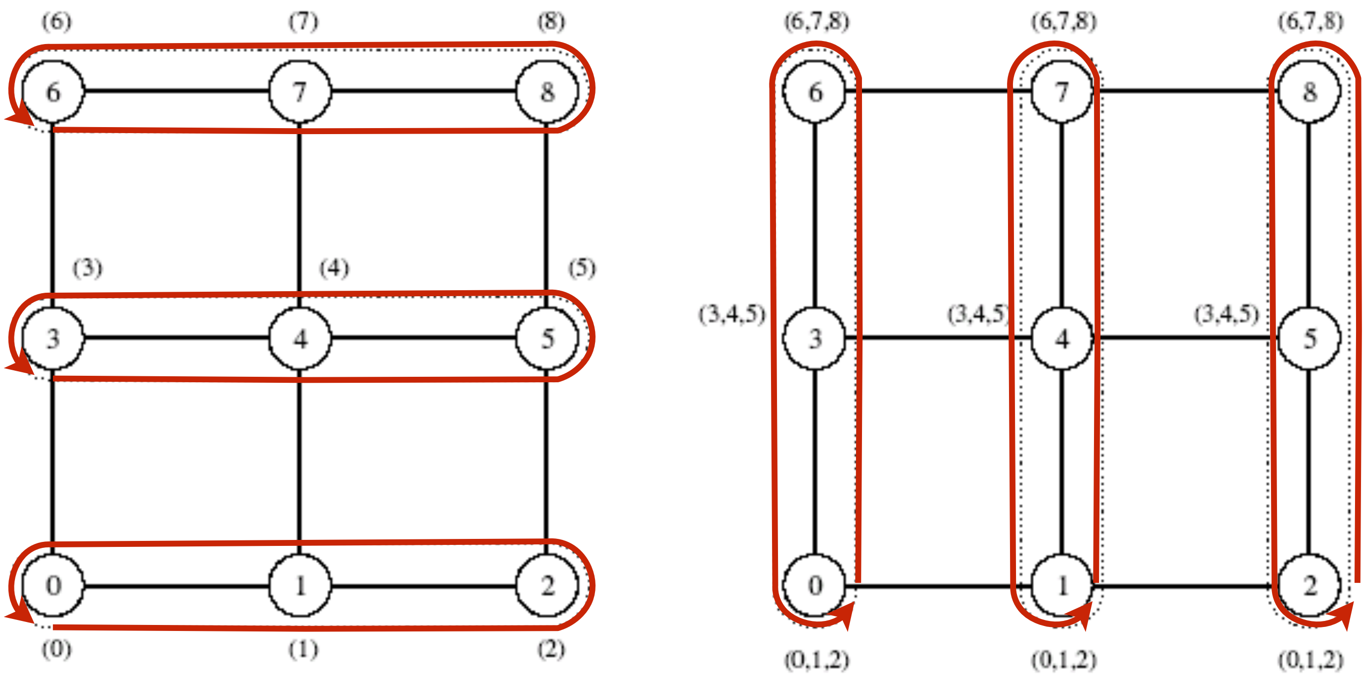
For an all-to-all reduction

- combine (rather than append) each incoming message into your local result
- at each step, forward your incoming msg to your successor

All-to-all Broadcast on a Mesh

Two phases

- **Perform row-wise all-to-all broadcast as for linear array/ring**
  - **—each node collects $p^{1/2}$ messages for nodes in its own row**
  - **—consolidates into a single message of size $mp^{1/2}$**
- **Perform column-wise all-to-all broadcast of merged messages**

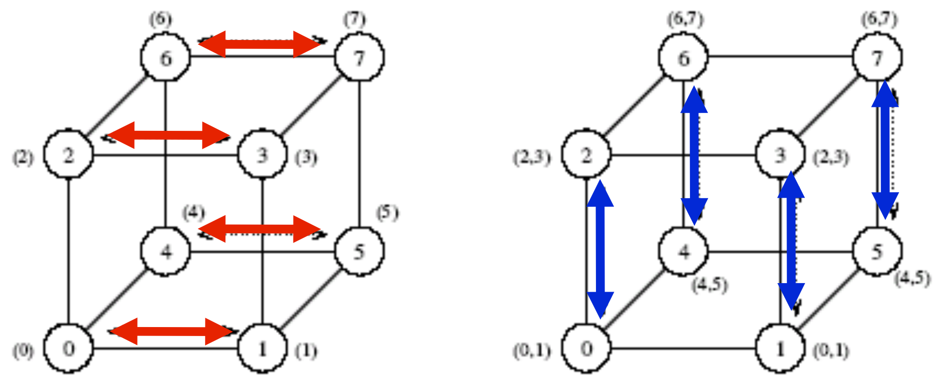


(a) Initial data distribution (b) Data distribution after rowwise broadcast

All-to-all Broadcast on a Hypercube

- Generalization of the mesh algorithm to *log p* dimensions
- Message size doubles in each of *log p* steps

1 value @ each

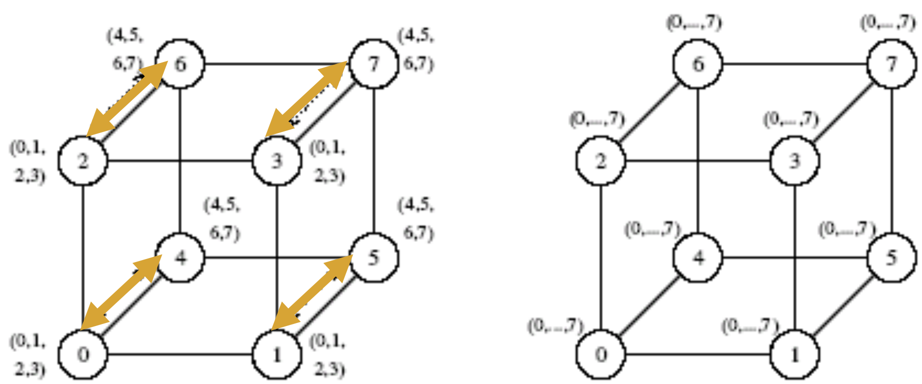


(a) Initial distribution of messages

(b) Distribution before the second step

2 values @ each

4 values @ each



(c) Distribution before the third step

(d) Final distribution of messages

8 values @ each

All-to-all Broadcast on a Hypercube

```
procedure ALL_TO_ALL_BC_HCUBE(my_id, my_msg, d, result)
begin
    result := my_msg;
    for i := 0 to d - 1 do
        partner := my_id XOR 2^i;
        send result to partner;
        receive msg from partner;
        result := result ∪ msg;
    endfor;
end ALL_TO_ALL_BC_HCUBE
```

All-to-all Reduction

- Similar to all-to-all broadcast, except for the merge
- Algorithm sketch

```
my_result = local_value

for each round
    send my_result to partner
    receive msg
    my_result = my_result ⊕ msg

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

Cost Analysis for All-to-All Broadcast

- **Ring**
  - $(t_s + t_w m)(p-1)$
- **Mesh**
  - phase 1: $(t_s + t_w m)(p^{1/2} - 1)$
  - phase 2: $(t_s + t_w m p^{1/2})(p^{1/2} - 1)$
  - total: $2t_s(p^{1/2} - 1) + t_w m(p - 1)$
- **Hypercube**

$$T = \sum_{i=1}^{\log p} (t_s + 2^{i-1} t_w m)$$

$$= t_s \log p + t_w m(p-1).$$

Above algorithms are asymptotically optimal in msg size

Prefix Sum

- **Pre-condition**
  - given p numbers $n_0, n_1, \ldots, n_{p-1}$ (one on each node)
    - node labeled k contains n_k
- **Problem statement**
  - compute the sums $s_k = \sum_{i=0}^{k} n_i$ for all k between 0 and $p-1$
- **Post-condition**
  - node labeled k contains s_k

Prefix Sum

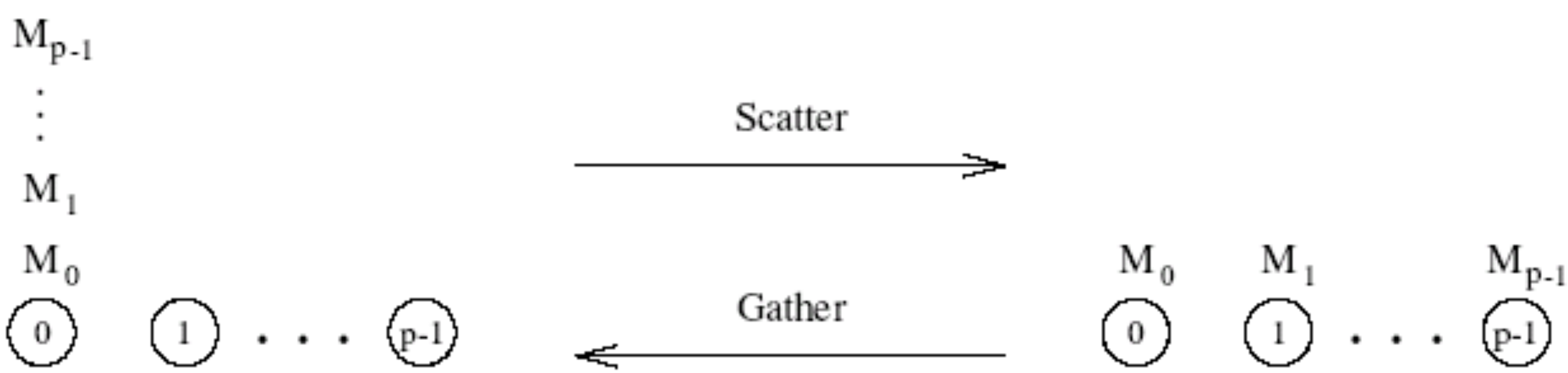
- **Can use all-to-all reduction kernel to implement prefix sum**
- **Constraint**
  - **prefix sums on node *k*: values from *k*-node subset with labels ≤ *k***
- **Strategy**
  - **implemented using an additional result buffer**
  - **add incoming value to result buffer on node *k***
    - **only if the msg from a node ≤ *k***

Prefix Sum on a Hypercube

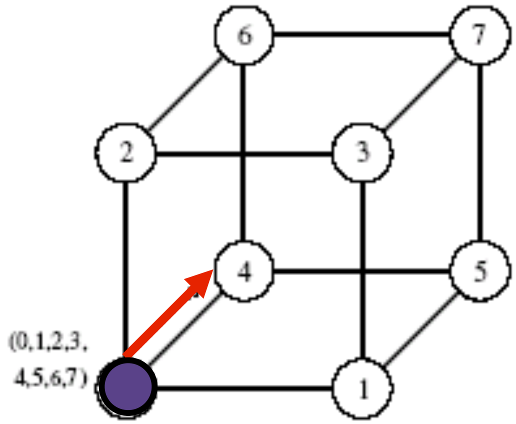
```
procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
begin
    result := my_number;
    msg := result;
    for i := 0 to d - 1 do
        partner := my_id XOR 2^i;
        send msg to partner;
        receive number from partner;
        msg := msg + number;
        if (partner < my_id) then result := result + number;
    endfor;
end PREFIX_SUMS_HCUBE
```

Scatter and Gather

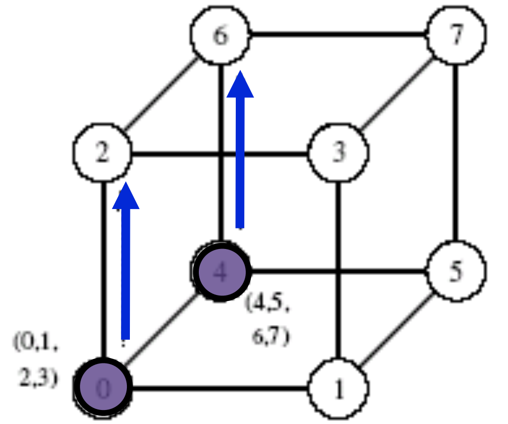
- **Scatter**
  - —a node sends a unique message of size *m* to every other node
    - – AKA one-to-all personalized communication
  - —algorithmic structure is similar to broadcast
    - – scatter: message size get smaller at each step
    - – broadcast: message size stay constant
- **Gather**
  - —single node collects a unique message from each node
  - —inverse of the scatter operation; can be executed as such



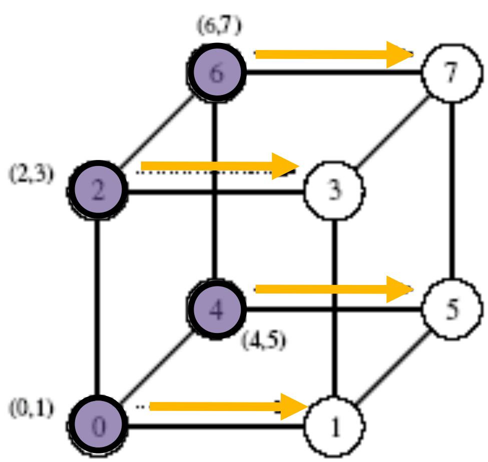
Scatter on a Hypercube



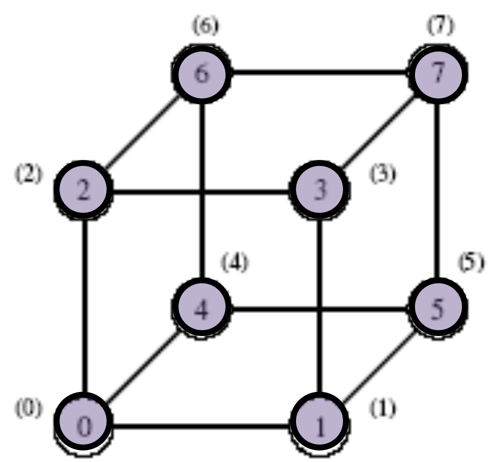
(a) Initial distribution of messages



(b) Distribution before the second step



(c) Distribution before the third step



(d) Final distribution of messages

Cost of Scatter and Gather

- Log *p* steps
  - —in each step
    - machine size halves
    - message size halves
- Time

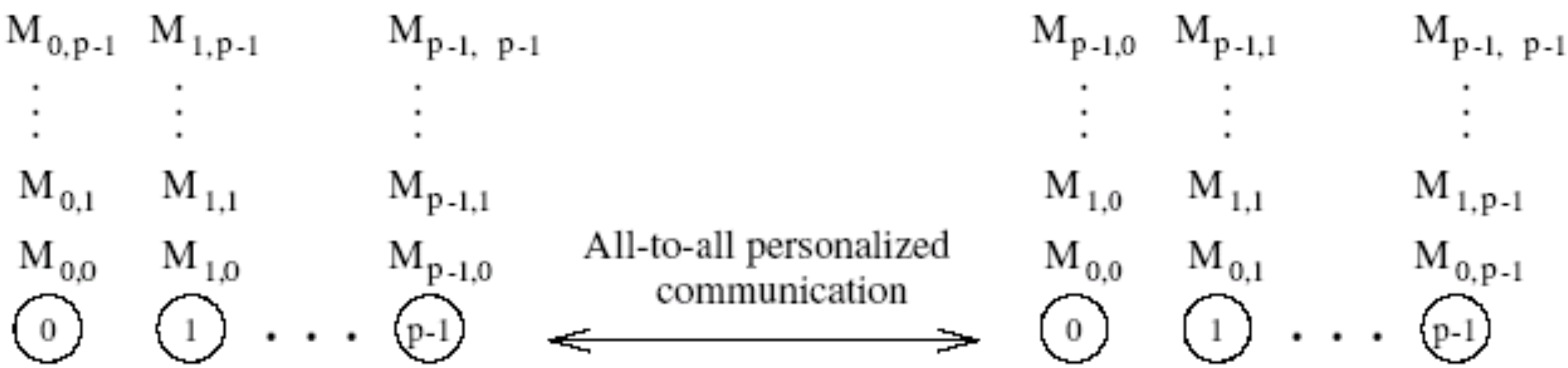
$$T = t_s \log p + t_w m(p-1).$$

- Note: time is asymptotically optimal in message size

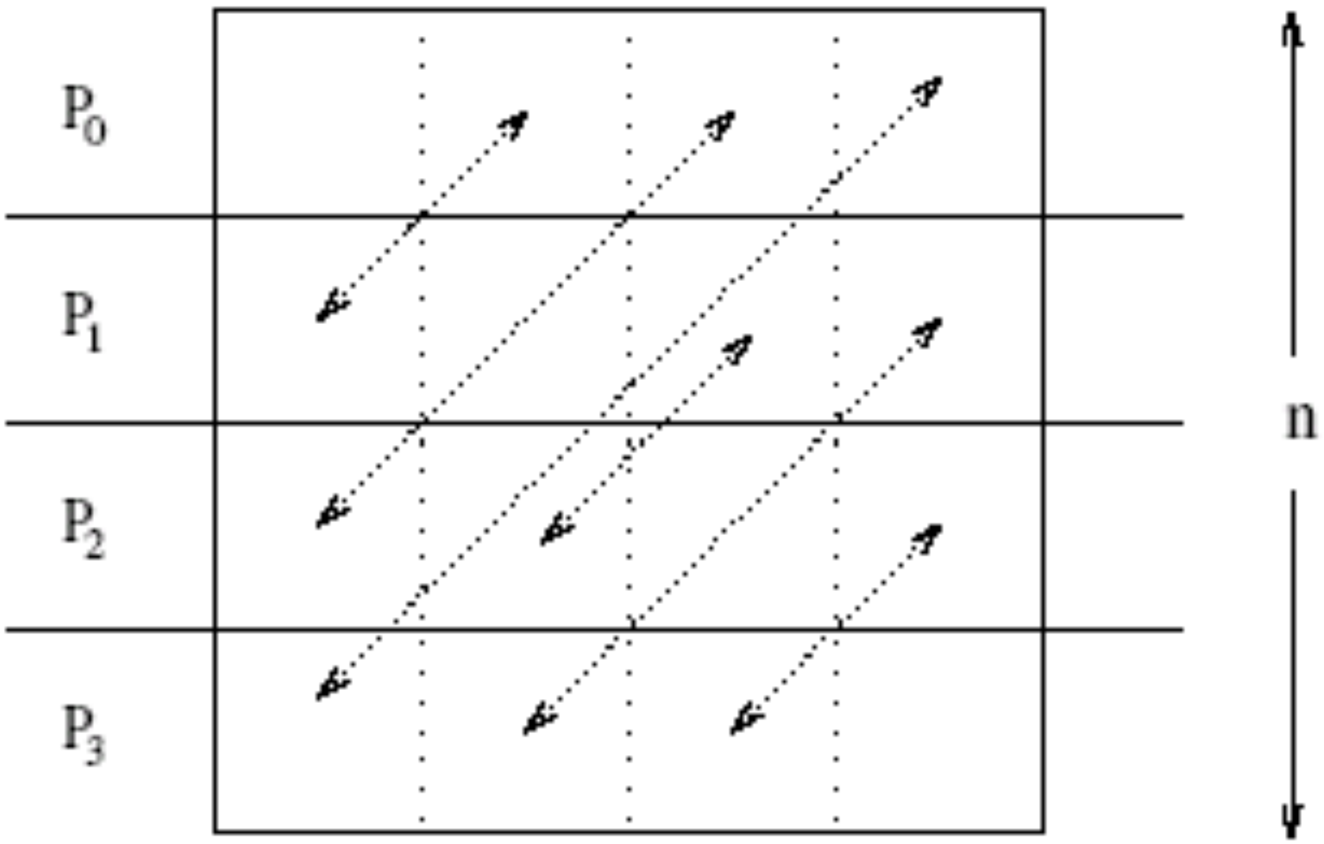
All-to-All Personalized Communication

Total exchange

- **Each node: distinct message of size *m* for every other node**



All-to-All Personalized Communication



All-to-All Personalized Communication

- Every node has p pieces of data, each of size m
- Algorithm sketch for a ring

```
for k = 1 to p - 1
  send message of size m(p - k) to neighbor
  select piece of size m out of message for self
```

- Cost analysis

$$
\begin{aligned}
T &= \sum_{i=1}^{p-1} (t_s + t_w m(p-i)) \\
&= t_s(p-1) + t_w m \sum_{i=1}^{p-1} i \\
&= (t_s + t_w mp/2)(p-1)
\end{aligned}
$$

Optimizing Collective Patterns

Example: one-to-all broadcast of large messages on a hypercube

- **Consider broadcast of message *M* of size *m*, where *m* is large**
- **Cost of straightforward strategy** $T = (t_s + t_w m)\log p$
- **Optimized strategy**
  - —split *M* into *p* parts $M_0, M_1, \ldots M_p$ of size *m/p* each
    - – want to place $M_0 \cup M_1 \cup \ldots \cup M_p$ on all nodes
  - —scatter M_i to node *i*
  - —have nodes collectively perform all-to-all broadcast
    - – each node *k* broadcasts its M_k
- **Cost analysis**
  - —scatter time = $t_s \log p + t_w(m/p)(p-1)$ *(slide 27)*
  - —all-to-all broadcast time = $t_s \log p + t_w(m/p)(p-1)$ *(slide 21)*
  - —total time = $2(t_s \log p + t_w(m/p)(p-1)) \approx 2(t_s \log p + t_w m)$ *(faster than slide 13)*

References

- **Adapted from slides "Principles of Parallel Algorithm Design" by Ananth Grama**
- **Based on Chapter 4 of "Introduction to Parallel Computing" by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Addison Wesley, 2003**