(How to Implement) Basic Communication Operations

Alexandre David
1.2.05
Overview

- One-to-all broadcast & all-to-one reduction (4.1).
- All-to-all broadcast and reduction (4.2).
- All-reduce and prefix-sum operations (4.3).
- Scatter and Gather (4.4).
- All-to-All Personalized Communication (4.5).
- Circular Shift (4.6).
- Improving the Speed of Some Communication Operations (4.7).
Collective Communication Operations

- Represent regular communication patterns.
- Used extensively in most data-parallel algorithms.
- Critical for efficiency.
- Available in most parallel libraries.
- Very useful to “get started” in parallel processing.

Collective: involve group of processors.

The efficiency of data-parallel algorithms depends on the efficient implementation of these operations.

Recall: $t_s + mt_w$ time for exchanging a $m$-word message with cut-through routing.

All processes participate in a single global interaction operation or subsets of processes in local interactions.

Goal of this chapter: good algorithms to implement commonly used communication patterns.
Reminder

- Result from previous analysis:
  - Data transfer time is roughly the same between all pairs of nodes.
  - Homogeneity true on modern hardware (randomized routing, cut-through routing...).
    - $t_s + mt_w$
    - Adjust $t_w$ for congestion: effective $t_w$.
- Model: bidirectional links, single port.
- Communication with point-to-point primitives.
Broadcast/Reduction

- One-to-all broadcast:
  - Single process sends identical data to all (or subset of) processes.
- All-to-one reduction:
  - Dual operation.
  - $P$ processes have $m$ words to send to one destination.
  - Parts of the message need to be combined.

Reduction can be used to find the sum, product, maximum, or minimum of sets of numbers.
This is the logical view, what happens from the programmer’s perspective.
One-to-All Broadcast—Ring/Linear Array

- Naïve approach: send sequentially.
  - Bottleneck.
  - Poor utilization of the network.
- Recursive doubling:
  - Broadcast in $\log p$ steps (instead of $p$).
  - Divide-and-conquer type of algorithm.
  - Reduction is similar.

Source process is the bottleneck. Poor utilization: Only connections between single pairs of nodes are used at a time.

Recursive doubling: All processes that have the data can send it again.
Note:

• The nodes do not snoop the messages going “through” them. Messages are forwarded but the processes are not notified of this because they are not destined to them.

• Choose carefully destinations: furthest.

• Reduction symmetric: Accumulate results and send with the same pattern.
Example: Matrix*Vector

Although we have a matrix & a vector the broadcast are done on arrays.
One-to-All Broadcast – Mesh

- Extensions of the linear array algorithm.
  - Rows & columns = arrays.
  - Broadcast on a row, broadcast on columns.
  - Similar for reductions.
  - Generalize for higher dimensions (cubes...).
1. Broadcast like linear array.
2. Every node on the linear array has the data and broadcast on the columns with the linear array algorithm, *in parallel*.
One-to-All Broadcast – Hypercube

- Hypercube with $2^d$ nodes = $d$-dimensional mesh with 2 nodes in each direction.
- Similar algorithm in $d$ steps.
- Also in $\log p$ steps.
- Reduction follows the same pattern.
Better for congestion: Use different links every time. Forwarding in parallel again.
All-to-One Broadcast – Balanced Binary Tree

- Processing nodes = leaves.
- Hypercube algorithm maps well.
- Similarly good w.r.t. congestion.
Broadcast on a Balanced Binary Tree

Divide-and-conquer type of algorithm again.
Algorithms

- So far we saw pictures.
- Not enough to implement.
- Precise description
  - to implement.
  - to analyze.
- Description for hypercube.
- Execute the following procedure on all the nodes.

For sake of simplicity, the number of nodes is a power of 2.
my\_id is the label of the node the procedure is executed on. The procedure performs \( d \) communication steps, one along each dimension of the hypercube. Nodes with zero in \( i \) least significant bits (of their labels) participate in the communication.
**Broadcast Algorithm**

1. procedure ONE_TO_ALL_BC(d, my_id, X)
2. begin
3. \[\text{mask} := 2^d - 1;\] /* Set all d bits of mask to 1 */
4. for \(i := d - 1\) downto 0 do /* Outer loop */
5. \[\text{mask} := \text{mask XOR} 2^i;\] /* Set bit i of mask to 0 */
6. if \((\text{my_id AND mask}) = 0\) then /* If lower i bits of my_id are 0 */
7. \[\text{msg\_destination} := \text{my_id XOR} 2^i;\]
8. \[\text{send} X \text{ at} \text{msg\_destination};\]
9. else
10. \[\text{msg\_source} := \text{my_id XOR} 2^i;\]
11. \[\text{receive} X \text{ from} \text{msg\_source};\]
12. endif;
13. endif;
14. end ONE_TO_ALL_BC

*my_id* is the label of the node the procedure is executed on. The procedure performs \(d\) communication steps, one along each dimension of the hypercube. Nodes with zero in \(i\) least significant bits (of their labels) participate in the communication.
my_id is the label of the node the procedure is executed on. The procedure performs $d$ communication steps, one along each dimension of the hypercube.

Nodes with zero in $i$ least significant bits (of their labels) participate in the communication.

Notes:

• Every node has to know when to communicate, i.e., call the procedure.
• The procedure is distributed and requires only point-to-point synchronization.
• Only from node 0.
XOR the source = renaming relative to the source. Still works because of the sub-cube property: changing 1 bit = navigate on one dimension, keep a set of equal bits = sub-cube.
Reduce Algorithm

1. procedure ALL_TO_ONE_REDUCE(d, my_id, m, X, sum)
2. begin
3. for \( j := 0 \) to \( m - 1 \) do \( sum[j] := X[j] \);
4. \( mask := 0 \);
5. for \( i := 0 \) to \( d - 1 \) do
   /* Select nodes whose lower \( i \) bits are 0 */
   if \( (my_id \text{ AND } mask) = 0 \) then
      if \( (my_id \text{ AND } 2^i) \neq 0 \) then
         \( msg\_destination := my\_id \text{ XOR } 2^i \);
   end if
9. In a nutshell:
   reverse the previous one.
11. receive \( X \) from \( msg\_source \);
12. for \( j := 0 \) to \( m - 1 \) do
15. end for;
16. \( mask := mask \text{ XOR } 2^i \); /* Set bit \( i \) of \( mask \) to 1 */
17. end ALL_TO_ONE_REDUCE
Cost Analysis

\[ \rho \text{ processes } \rightarrow \log \rho \text{ steps (point-to-point transfers in parallel).} \]
Each transfer has a time cost of \( t_s + t_w m \).
Total time: \( T = (t_s + t_w m) \log \rho \).
All-to-All Broadcast and Reduction

- Generalization of broadcast:
  - Each processor is a source and destination.
  - Several processes broadcast different messages.
- Used in matrix multiplication (and matrix-vector multiplication).
- Dual: all-to-all reduction.

How to do it?
If performed naively, it may take up to $p$ times as long as a one-to-all broadcast (for $p$ processors).
Possible to concatenate all messages that are going through the same path (reduce time because fewer $t_s$).
All-to-All Broadcast and Reduction

Figure 4.8 All-to-all broadcast and all-to-all reduction.
All-to-All Broadcast – Rings

All communication links can be kept busy until the operation is complete because each node has some information to pass. One-to-all in \( \log p \) steps, all-to-all in \( p-1 \) steps instead of \( p \log p \) (naïve).

How to do it for linear arrays? If we have bidirectional links (assumption from the beginning), we can use the same procedure.
All-to-All Broadcast Algorithm

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

Algorithm 4.4 All-to-all broadcast on a \( p \)-node ring.
All-to-All Reduce Algorithm

1. procedure ALL_TO_ALL_RED_RING(my_id, my_msg, p, result)
2. begin
3. left := (my_id - 1) mod p;
4. right := (my_id + 1) mod p;
5. recv := 0;
6. for i := 1 to p - 1 do
7. j := (my_id + i) mod p;
8. temp := my[j] + recv;
9. send temp to left;
10. receive recv from right;
11. endfor;
12. result := sum[my_id] + next;
13. end ALL_TO_ALL_RED_RING

Accumulate and forward.

Last message for my_id.

Algorithm 4.5 All-to-all reduction on a p-node ring.
All-to-All Reduce – Rings

Diagram showing a network of nodes and connections, illustrating the communication pattern for an all-to-all reduce operation using rings. The nodes are labeled with numbers, and the connections show how data is transferred from one node to another.
All-to-All Reduce – Rings

$p-1$ steps.
All-to-All Broadcast – Meshes

Two phases:

- All-to-all on rows – messages size $m$.
  - Collect $\sqrt{p}$ messages.
- All-to-all on columns – messages size $\sqrt{p} \times m$. 
All-to-All Broadcast – Meshes

(0) (1) (2)

(3) (4) (5)

(6) (7) (8)

0 1 2

3 4 5

6 7 8

(01,2) (01,2) (01,2)

(03,5) (03,5) (03,5)

(67,8) (67,8) (67,8)
Algorithm

1. procedure ALL-TO-ALL_BC_MESH(my_id, my_msg, p, result)
2. begin

/* Communication along rows */
3. left := my_id - (my_id mod √p) + (my_id - 1) mod √p;
4. right := my_id - (my_id mod √p) + (my_id + 1) mod √p;
5. result := result ∪ 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;

/* Communication along columns */
12. up := (my_id - √p) mod p;
13. down := (my_id + √p) mod p;
14. msg := result;
15. for i := 1 to √p - 1 do
16. send msg to down;
17. receive msg from up;
18. result := result ∪ msg;
19. endfor;
20. end ALL-TO-ALL_BC_MESH
All-to-All Broadcast - Hypercubes

- Generalization of the mesh algorithm to $\log p$ dimensions.
- Message size doubles at every step.
- Number of steps: $\log p$.

Remember the 2 extremes:
- Linear array: $p$ nodes per (1) dimension – $p^1$.
- Hypercubes: 2 nodes per $\log p$ dimensions – $2^{\log p}$.

And in between 2-D mesh $\sqrt{p}$ nodes per (2) dimensions – $\sqrt{p}^2$. 
All-to-All Broadcast – Hypercubes

(a) Initial distribution of messages

(b) Distribution before the second step

(c) Distribution before the third step

(d) Final distribution of messages
At every step we have a broadcast on sub-cubes. The size of the sub-cubes doubles at every step and all the nodes exchange their messages.
All-to-All Reduction – Hypercubes

1. procedure ALL_TO_ALL_RED_HCUBE(my_id, msg, d, result)
2. begin
3.   recloc := 0;
4.   for i := d − 1 to 0 do
5.     partner := my_id XOR 2^i;
6.     j := my_id AND 2^i;
7.     k := (my_id XOR 2^i) AND 2^i;
8.     senloc := recloc + k;
9.     recloc := recloc + j;
10.    send msg[senloc .. senloc + 2^i − 1] to partner;
11.   receive temp[] .. 2^i − 1 from partner;
12.   for j := 0 to 2^i − 1 do
14.   endfor;
15. end ALL_TO_ALL_RED_HCUBE

Algorithm 4.8 All-to-all broadcast on a d-dimensional hypercube. AND and XOR are bitwise logical-and and exclusive-or operations, respectively.

Similar pattern in reverse order.

Combine results
Cost Analysis (Time)

- **Ring:**
  \[ T = (t_s + twm)(p-1). \]

- **Mesh:**
  \[ T = (t_s + twm)\sqrt{p-1} + (t_s + twm/p) \sqrt{p-1} \]
  \[ = 2ts\sqrt{p - 1} + twm(p-1). \]

- **Hypercube:**
  \[ T = \sum_{i=1}^{\log_p} (t_s + 2^{i-1}twm) \log_p \text{ steps} \]
  \[ = t_s \log p + twm(p-1). \]
  \[ \text{message of size } 2^{i-1}m. \]

Lower bound for the communication time of all-to-all broadcast for parallel computers on which a node can communicate on only one of its ports at a time = \( twm(p-1) \). Each node receives at least \( m(p-1) \) words of data. That's for **any** architecture.

The straight-forward algorithm for the simple ring architecture is interesting: It is a sequence of \( p \) one-to-all broadcasts with different sources every time. The broadcasts are pipelined. That's common in parallel algorithms.

We cannot use the hypercube algorithm on smaller dimension topologies because of congestion.
Contention because communication is done on links with single ports. Contention is in the sense of the access to the link. The result is congestion on the traffic.
**All-Reduce**

- Each node starts with a buffer of size $m$.
- The final result is the same combination of all buffers on every node.
- Same as all-to-one reduce + one-to-all broadcast.
- Different from all-to-all reduce.

All-to-all reduce combines $p$ different messages on $p$ different nodes. All-reduce combines 1 message on $p$ different nodes.
All-Reduce Algorithm

- Use all-to-all broadcast but
  - Combine messages instead of concatenating them.
  - The size of the messages does not grow.
  - Cost (in $\log p$ steps): $T = (t_s + t_w \cdot m) \log p$. 
Prefix-Sum

- Given \( p \) numbers \( n_0, n_1, \ldots, n_{p-1} \) (one on each node), the problem is to compute the sums \( s_k = \sum_{i=0}^{k} n_i \) for all \( k \) between 0 and \( p-1 \).
- Initially, \( n_k \) is on the node labeled \( k \), and at the end, the same node holds \( S_k \).

This is a reminder.
Prefix-Sum Algorithm

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. end for;
12. end PREFIX_SUMS_HCUBE

Algorithm 4.9 Prefix sums on a d-dimensional hypercube.
Figure in the book is messed up.
Scatter and Gather

- Scatter: A node sends a unique message to every other node – *unique per node*.
- Gather: Dual operation but the target node does not combine the messages into one.

\[
\begin{array}{c}
M_2 \\
M_1 \\
M_0 \\
0 \\
1 \\
2 \\
\ldots
\end{array} \quad \xrightarrow{\text{Scatter}} \quad \begin{array}{c}
M_0 \\
M_1 \\
M_2 \\
0 \\
1 \\
2 \\
\ldots
\end{array}
\]

Do you see the difference with one-to-all broadcast and all-to-one reduce? Communication pattern similar.

Scatter = one-to-all personalized communication.
The pattern of communication is identical with one-to-all broadcast but the size and the content of the messages are different. Scatter is the reverse operation. This algorithm can be applied for other topologies.

How many steps? What’s the cost?
Cost Analysis

- Number of steps: $\log p$.
- Size transferred: $pm/2$, $pm/4$, $\ldots$, $m$.

- Geometric sum

  \[
  p + \frac{p}{2} + \frac{p}{4} + \ldots + \frac{p}{2^n} = p \frac{1 - \frac{1}{2^{n+1}}}{1 - \frac{1}{2}}
  \]

  \[
  \frac{p}{2} + \frac{p}{4} + \ldots + \frac{p}{2^n} = 2p(1 - \frac{1}{2^{n+1}}) - p = 2p(1 - \frac{1}{2p}) - p = p - 1
  \]

  \[
  (2^{n+1} = 2^{1+\log p} = 2p)
  \]

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

The term $t_w m(p-1)$ is a lower bound for any topology because the message of size $m$ has to be transmitted to $p-1$ nodes, which gives the lower bound of $m(p-1)$ words of data.
All-to-All Personalized Communication

- Each node sends a distinct message to every other node.

See the difference with all-to-all broadcast?
All-to-all personalized communication = total exchange.
Result = transpose of the input (if seen as a matrix).
Example: Transpose

Figure 4.17  All-to-all personalized communication in transposing a $4 \times 4$ matrix using four processes.
Total Exchange on a Ring

19+26-03-2008 Alexandre David, MVP'08
Total Exchange on a Ring
Cost Analysis

- Number of steps: \( p-1 \).
- Size transmitted: \( m(p-1), m(p-2), \ldots, m \).

\[
T = t_s (p - 1) + \sum_{i=1}^{p-1} it_w m = (t_s + t_w mp / 2)(p - 1)
\]

Optimal

In average we transmit \( mp/2 \) words, whereas the linear all-to-all transmits \( m \) words. If we make this substitution, we have the same cost as the previous linear array procedure. To really see optimality we have to check the lowest possible needed data transmission and compare it to \( T \).

Average distance a packet travels = \( p/2 \). There are \( p \) nodes that need to transmit \( m(p-1) \) words. Total traffic = \( m(p-1)^*p/2^*p \). Number of link that support the load = \( p \), to communication time \( \geq t_w m(p-1)p/2 \).
We use the procedure of the ring/array.
We use the procedure of the ring/array.
We use the procedure of the ring/array.
Cost Analysis

- Substitute $p$ by $\sqrt{p}$ (number of nodes per dimension).
- Substitute message size $m$ by $m\sqrt{p}$.
- Cost is the same for each dimension.
- $T=(2t_s+t_wmp)(\sqrt{p}-1)$

We have $p(\sqrt{p}-1)m$ words transferred, looks worse than lower bound in $(p-1)m$ but no congestion. Notice that the time for data rearrangement is not taken into account. It is almost optimal (by a factor 4), see exercise.
Total Exchange on a Hypercube

- Generalize the mesh algorithm to $\log p$ steps = number of dimensions, with 2 nodes per dimension.
- Same procedure as all-to-all broadcast.
Total Exchange on a Hypercube
Total Exchange on a Hypercube
Total Exchange on a Hypercube
Total Exchange on a Hypercube
Cost Analysis

- Number of steps: $\log p$.
- Size transmitted per step: $pm/2$.
- Cost: $T=(t_s+t_wmp/2)\log p$.
- Optimal? **NO**
  - Each node sends and receives $m(p-1)$ words.
  - Average distance = $(\log p)/2$. Total traffic = $p*m(p-1)*\log p/2$.
  - Number of links = $p \log p/2$.
  - Time lower bound = $t_wm(p-1)$.

Notes:

1. No congestion.
2. Bi-directional communication.
3. How to conclude if an algorithm is optimal or not: Check the possible lowest bound and see if the algorithm reaches it.
An Optimal Algorithm

- Have every pair of nodes communicate directly with each other – p-1 communication steps – but **without congestion**.
- At $j^{th}$ step node $i$ communicates with node $(i \text{xor } j)$ with **E-cube routing**.
Total Exchange on a Hypercube
Total Exchange on a Hypercube
Total Exchange on a Hypercube
Total Exchange on a Hypercube
Total Exchange on a Hypercube
Point: Transmit less, only to the needed node, and avoid congestion with E-cube routing.
Cost Analysis

- Remark: Transmit less, only what is needed, but more steps.
- Number of steps: $p-1$.
- Transmission: size $m$ per step.
- Cost: $T = (t_s + t_w m)(p-1)$.
- Compared with $T = (t_s + t_w m/2)\log p$.
- Previous algorithm better for small messages.

This algorithm is now optimal: It reaches the lowest bound.
A permutation = a redistribution in a set. 
You can call the shift a rotation in fact.
Circular 5-shift on a mesh.

\[ q \mod \sqrt{p} \text{ on rows} \]
\[ \lfloor q / \sqrt{p} \rfloor \text{ on columns} \]
Circular Shift on a Hypercube

- Map a linear array with $2^d$ nodes onto a hypercube of dimension $d$.
- Expand q shift as a sum of powers of 2 (e.g. 5-shift = $2^0 + 2^2$).
- Perform the decomposed shifts.
- Use bi-directional links for “forward” (shift itself) and “backward” (rotation part) ... log$p$ steps.

Backward and forward may be misleading in the book.

Interesting but not best solution, no idea why it’s mentioned if the optimal solution is simpler.
Exercise: Check the E-cube routing and convince me that there is no congestion.
Communication time = $t_s + t_wm$ in one step.
Improving Performance

- So far messages of size $m$ were not split.
- If we split them into $p$ parts:
  - One-to-all broadcast = scatter + all-to-all broadcast of messages of size $m/p$.
  - All-to-one reduction = all-to-all reduce + gather of messages of size $m/p$.
  - All-reduce = all-to-all reduction + all-to-all broadcast of messages of size $m/p$. 