Collective Communication Implementations

Sathish Vadhiyar
Binomial Tree

**Definition (Binomial Tree)** The *binomial tree of order* $k \geq 0$ with root $R$ is the tree $B_k$ defined as follows:

1. If $k=0$, $B_k = \{R\}$. i.e., the binomial tree of order zero consists of a single node, $R$.
2. If $k > 0$, $B_k = \{R, B_0, B_1, \ldots B_{k-1}\}$. i.e., the binomial tree of order $k > 0$ comprises the root $R$, and $k$ binomial subtrees, $B_0 - B_{k-1}$.

- $B_k$ contains $2^k$ nodes
- The height of $B_k$ is $k$
- The number of nodes at level $l$ in $B_k$, where $0 \leq l \leq k$, is given by the *binomial coefficient* $^kC_l$
Binomial Trees

$B_0$

$B_1$

$B_2$

$B_3$
Binomial Trees
Binomial Trees

- Broadcast, Scatter and Gather usually implemented by binomial
- Takes $\log P$ communication steps instead of $2(\log P - 1)$ in binary
Barrier Algorithms

- **Butterfly barrier** by Eugene Brooks II
- In round $k$, $i$ synchronizes with $i \oplus 2^k$ pairwise.
- If $p$ not power of 2, existing procs. stand for missing ones.
- Worstcase – $2\log P$ pairwise synchronizations by a processor
Barrier Algorithms

- **Dissemination barrier** by Hensgen, Finkel and Manser
- In round $k$, $i$ signals $(i + 2^k) \bmod P$
- No pairwise synchronization
- Atmost $\log(\text{next power of 2} > P)$ on critical path irrespective of $P$

Stage 0

Stage 1

Stage 2

Stage 3

1 more round
Barrier Algorithms

- **MPICH Barrier** (pairwise exchange with recursive doubling)
  - Same as butterfly barrier.
  - If nodes not equal to power, find the nearest power of 2, i.e. $m = 2^n$
  - The last surfeit nodes, i.e. surfeit = size – m, initially send messages to the first surfeit number of nodes
  - The first m nodes then perform butterfly barrier
  - Finally, the first surfeit nodes send messages to the last surfeit nodes

```
Stage first
Stage 0
Stage 1
Stage 2
Stage last
```
Allgather implementation

- In general, optimized allxxx operations depend on hardware topology, network contentions etc.
- Circular/ring allgather
- Each process receives from left and sends to right
- P steps

![Diagram of allgather process]

Stage 0

Stage 1
Bruck’s Allgather

- Similar to dissemination barrier
- logP steps
AlltoAll

- The naive implementation

  for all procs. i in order{
    if i # my proc., then send to i and recv from i
  }

- MPICH implementation – similar to naïve, but doesn’t do it in order

  for all procs. i in order{
    dest = (my_proc+i)modP
    src = (myproc-i+P)modP
    send to dest and recv from src
  }
Reduce and AllReduce

- Reduce and allreduce can be implemented with tree algorithms, e.g. binary tree
- But in tree based algorithms, some processors are not involved in computation
- Rolf Rabenseifner of Stuttgart – algorithms for reduce and allreduce
Rabenseifner algorithm
from http://www.hlrs.de/organization/par/services/models/mpi/myreduce.c

- This algorithm is explained with the example of 13 nodes.
- The nodes are numbered 0, 1, 2, ... 12.
- The sendbuf content is a, b, c, ... m.
- Each buffer array is notated with ABCDEFGH, this means that e.g. 'C' is the third 1/8 of the buffer.
- size := number of nodes in the communicator.
- $2^{**n}$ := the power of 2 that is next smaller or equal to the size.
- $r := size - 2^{**n}$
- e.g., size=13, n=3, r=5
Rabenseifner algorithm
- Steps
from http://www.hlrs.de/organization/par/services/models/mpi/myreduce.c

1. compute n and r
2. if myrank < 2*r
   split the buffer into ABCD and EFGH
   even myrank:
   send buffer EFGH to myrank+1
   receive buffer ABCD from myrank+1
   compute op for ABCD
   receive result EFGH
   odd myrank:
   send buffer ABCD to myrank-1
   receive buffer EFGH from myrank-1
   compute op for EFGH send result EFGH

Result:
node: 0  2  4  6  8  10  11  12
value: a+b  c+d  e+f  g+h  i+j  k  l  m
Rabenseifner algorithm
- Steps

3. if(myrank is even && myrank < 2*r) || (myrank >= 2*r)
4. define NEWRANK(old) := (old < 2*r ? old/2 : old - r)
   define OLDRENK(new) := (new < r ? new*2 : new + r)

Result:
old:  0   2   4   6   8   10  11  12
new:  0   1   2   3   4   5   6   7
val: a+b  c+d  e+f  g+h  i+j  k   l   m
Rabenseifner algorithm
- Steps
from http://www.hlrs.de/organization/par/services/models/mpi/myreduce.c

5.1 Split the buffer (ABCDEFGH) in the middle,
the lower half (ABCD) is computed on even (new) ranks,
the upper half (EFGH) is computed on odd (new) ranks.

exchange:
ABCD from 1 to 0, from 3 to 2, from 5 to 4 and from 7 to 6
EFGH from 0 to 1, from 2 to 3, from 4 to 5 and from 6 to 7
compute op in each node on its half

Result:
node 0: (a+b)+(c+d) for ABCD
node 1: (a+b)+(c+d) for EFGH
node 2: (e+f)+(g+h) for ABCD
node 3: (e+f)+(g+h) for EFGH
node 4: (i+j)+k for ABCD
node 5: (i+j)+k for EFGH
node 6: l+m for ABCD
node 7: l+m for EFGH
Rabenseifner algorithm
- Steps
from http://www.hlrs.de/organization/par/services/models/mpi/myreduce.c

5.2 Same with double distance and one more the half of the buffer.

Result:
node 0: \([(a+b)+(c+d)] + [(e+f)+(g+h)] \) for AB
node 1: \([(a+b)+(c+d)] + [(e+f)+(g+h)] \) for EF
node 2: \([(a+b)+(c+d)] + [(e+f)+(g+h)] \) for CD
node 3: \([(a+b)+(c+d)] + [(e+f)+(g+h)] \) for GH
node 4: \([(i+j)+ k ] + [ l + m ] \) for AB
node 5: \([(i+j)+ k ] + [ l + m ] \) for EF
node 6: \([(i+j)+ k ] + [ l + m ] \) for CD
node 7: \([(i+j)+ k ] + [ l + m ] \) for GH
Rabenseifner algorithm
- Steps

from http://www.hlrs.de/organization/par/services/models/mpi/myreduce.c

5.3 Same with double distance and one more the half of the buffer.

Result:

node 0: \{ [(a+b)+(c+d)] + [(e+f)+(g+h)] \} + \{ [(i+j)+k] + [l+m] \} for A
node 1: \{ [(a+b)+(c+d)] + [(e+f)+(g+h)] \} + \{ [(i+j)+k] + [l+m] \} for E
node 2: \{ [(a+b)+(c+d)] + [(e+f)+(g+h)] \} + \{ [(i+j)+k] + [l+m] \} for C
node 3: \{ [(a+b)+(c+d)] + [(e+f)+(g+h)] \} + \{ [(i+j)+k] + [l+m] \} for G
node 4: \{ [(a+b)+(c+d)] + [(e+f)+(g+h)] \} + \{ [(i+j)+k] + [l+m] \} for B
node 5: \{ [(a+b)+(c+d)] + [(e+f)+(g+h)] \} + \{ [(i+j)+k] + [l+m] \} for F
node 6: \{ [(a+b)+(c+d)] + [(e+f)+(g+h)] \} + \{ [(i+j)+k] + [l+m] \} for D
node 7: \{ [(a+b)+(c+d)] + [(e+f)+(g+h)] \} + \{ [(i+j)+k] + [l+m] \} for H
Rabenseifner algorithm
- Steps (for reduce)

from http://www.hlrs.de/organization/par/services/models/mpi/myreduce.c

6. Last step is gather
a. Gather is by an algorithm similar to tournament algorithm.
b. In each round, one process gathers from another
c. The players are P/2 at the initial step and the distance is halved every step till 1.

node 0: recv B => AB
node 1: recv F => EF
node 2: recv D => CD
node 3: recv H => GH
node 4: send B
node 5: send F
node 6: send D
node 7: send H

node 0: recv CD => ABCD
node 1: recv GH => EFGH
node 2: send CD
node 3: send GH
node 0: recv EFGH => ABCDEFGH
Rabenseifner algorithm
- Steps (for allreduce)
from http://www.hlrs.de/organization/par/services/models/mpi/myreduce.c

6. Similar to reduce, but
a. Instead of gather by one process, both the processes exchange
b. Both the players move to the next round
c. Finally, transfer the result from the even to odd nodes in the old ranks.

<table>
<thead>
<tr>
<th>node 0: AB</th>
<th>node 0: ABCD</th>
<th>node 0: ABCDEFGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>node 1: EF</td>
<td>node 1: EFGH</td>
<td>node 1: ABCDEFGH</td>
</tr>
<tr>
<td>node 2: CD</td>
<td>node 2: ABCD</td>
<td>node 2: ABCDEFGH</td>
</tr>
<tr>
<td>node 3: GH</td>
<td>node 3: EFGH</td>
<td>node 3: ABCDEFGH</td>
</tr>
<tr>
<td>node 4: AB</td>
<td>node 4: ABCD</td>
<td>node 4: ABCDEFGH</td>
</tr>
<tr>
<td>node 5: EF</td>
<td>node 5: EFGH</td>
<td>node 5: ABCDEFGH</td>
</tr>
<tr>
<td>node 6: CD</td>
<td>node 6: ABCD</td>
<td>node 6: ABCDEFGH</td>
</tr>
<tr>
<td>node 7: GH</td>
<td>node 7: EFGH</td>
<td>node 7: ABCDEFGH</td>
</tr>
</tbody>
</table>
General Notes on Optimizing Collectives

- 2 components for collective communications – latency and bandwidth
- Latency(α) – time when the collective completes with the first byte (or) number of time steps
- Bandwidth(β) – rate at which collective proceeds after the first byte transmission (or) total time for all messages
- Cost for communication – α+nβ
- Latency is critical for small message sizes and bandwidth for large message sizes
Example - Broadcast

- Binomial Broadcast
  - log \( p \) steps
  - Amount of data communicated at each step - \( n \)
  - cost = \( \log p \) (\( \alpha + n\beta \))

- scatter and allgather
  - Divide message into \( p \) segments
  - Scatter the \( p \) segments to \( p \) processes using binomial scatter
    - \( \log p \) \( \alpha \) + \( \frac{n}{p}(p-1) \) \( \beta \)
  - Scattered data collected at all processes using ring allgather
    - \( (p-1) \) \( \alpha \) + \( \frac{n}{p}(p-1) \) \( \beta \)
  - cost = \( \log p + p-1 \) \( \alpha \) + \( 2\frac{n}{p}(p-1) \) \( \beta \)

- Hence binomial broadcast for small messages and (scatter+allgather) for long messages
MPICH Algorithms

- Allgather
  - Bruck Algorithm (variation of dissemination) (< 80 KB) and non-power-of-two
  - Recursive doubling (< 512 KB) for power-of-2 processes
  - ring (> 512 KB) and (80-512 KB) for any processes

- Broadcast
  - Binomial (< 12 KB), binomial scatter + ring all_gather (> 512 KB)

- Alltoall
  - Bruck’s algorithm (for < 256 bytes)
  - Post all irecvs and isends (for medium size messages. 256 bytes – 32 KB)
  - Pairwise exchange (for long messages and power-of-2 processors) – p-1 steps. In each step k, each process i exchanges data with (i xor k)
  - For non-power of 2, an algorithm in which in each step, k, process i sends data to (i+k) and receives from (i-k)
MPICH Algorithms

- **Reduce-scatter**
  - For commutative operations:
    - Recursive halving (< 512 KB), pairwise exchange (> 512 KB; \(p-1\) steps; rank+i at step i)
  - For non-commutative:
    - Recursive doubling (< 512 bytes), pairwise exchange (> 512 bytes)

- **Reduce**
  - For pre-defined operations
    - Binomial algorithm (< 2 KB), Rabenseifner (> 2 KB)
  - For user-defined operations
    - Binomial algorithm

- **AllReduce**
  - For pre-defined operations
    - Recursive doubling (short), Rabenseifner (long messages)
  - For user-defined operations
    - Recursive doubling
On Real Network Topologies

- Two communicating processes may be mapped onto two processors that are more than 1-hop away.
- Or different edges in the algorithm graph can map onto a same shared link, leading to contention.
Mapping Process Topologies Onto Network Topologies

- **Definition:** **Dilation** – Let $\varnothing$ be the function that embeds graph $G=(V,E)$ into graph $G'=(V',E')$. The dilation of the embedding is defined as
  \[
  \text{dil}(\varnothing) = \max\{\text{dist}(\varnothing(u), \varnothing(v)) | (u,v) \in E\}
  \]

- **Ring onto 2-D mesh:**
  - A dilation-1 embedding exists if the mesh has an even number of rows and/or columns
Binary Tree Onto 2-D mesh

- A dilation-1 embedding can exist for a tree of height 3 or less.
- H-tree is a common way of embedding a binary tree onto a mesh.
- A complete binary tree of height n has a dilation cap(n/2) embedding in a 2-D mesh.
- Similarly, a binomial tree of height n has a dilation cap(n/2) embedding in a 2-D mesh.
References

- Section 5.1 in the book by Quinn
Barrier Algorithms

- **Tournament barrier** by Hensgen, Finkel and Manser
- In the 1\textsuperscript{st} round, each pair of nodes (players) synchronize (play a game)
- The receiver will be considered as the winner of the game
- In the 2\textsuperscript{nd} round, the winners of the 1\textsuperscript{st} round will synchronize (play games)
- The receiver in the 2\textsuperscript{nd} round will advance to the 3\textsuperscript{rd} round
- This process continues till there is 1 winner left in the tournament
- The single winner then broadcasts a message to all the other nodes
- At each round $k$, proc. $j$ receives a message from proc. $i$, where $i = j - 2^k$
AlltoAll implementation

- Circular alltoall
- For step k in \( \{1..P\} \), proc. i sends to \((i+k) \mod P\) and receives from \((i-k+P) \mod P\)
Reduce-Scatter for commutative operations: Recursive halving algorithm

- Recursive doubling – in the first step, communication is with the neighboring process. In each step, the communication distance doubles.
- Recursive halving – reverse of recursive doubling
- At the first step:
  - a process communicates with another process P/2 away
  - sends data needed by the other half
  - Receives data needed by its half
  - Performs operation
- Next step – distance P/4 away and so on...
- \( \log P \) steps
Reduce-Scatter for non-commutative operations: Recursive doubling algorithm

- In the first step, data (all data except the one needed for its result) is exchanged with the neighboring process.
- In the next step, \((n-2n/p)\) data (all except the one needed by it and the one needed by process it communicated with the previous step) is communicated with process that is distance 2 apart.
- In the third step \((n-4n/p)\) data with process that is distance 4 apart and so on...
- \(\lg P\) steps