北京大学高能效计算与应用中心
Center for Energy-efficient Computing and Applications

Parallel & Distributed Computing

Lecture 5a: Message Passing and Collective Communications Basics

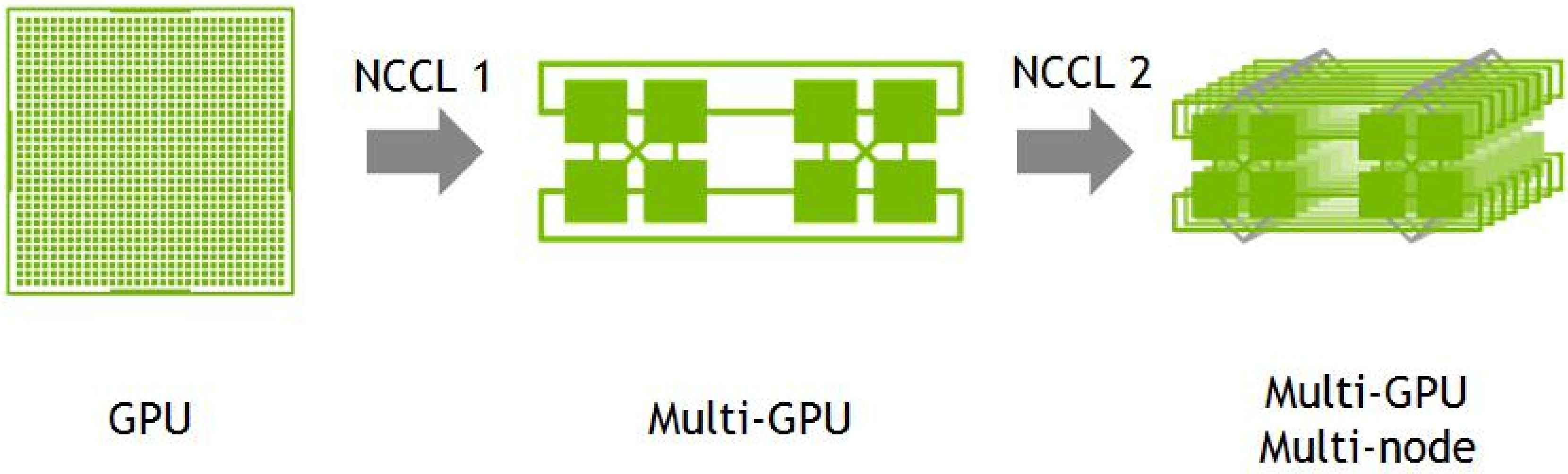
Spring 2025

Instructor: 罗国杰

gluo@pku.edu.cn

Collective Communications

- NVIDIA Collective Communications Library (NCCL)

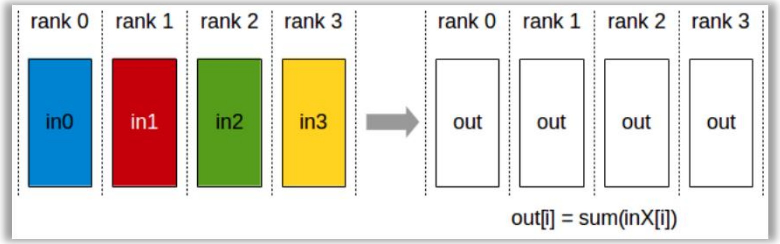


- Concepts in a modern library for collective communication

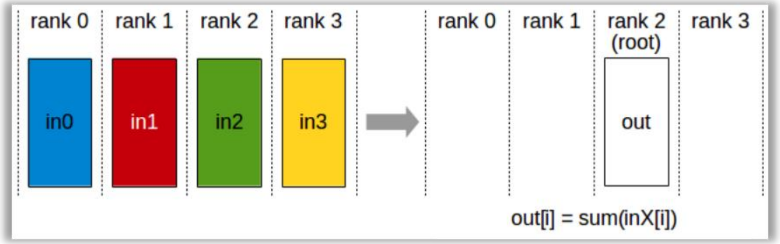
Example Collective Communications

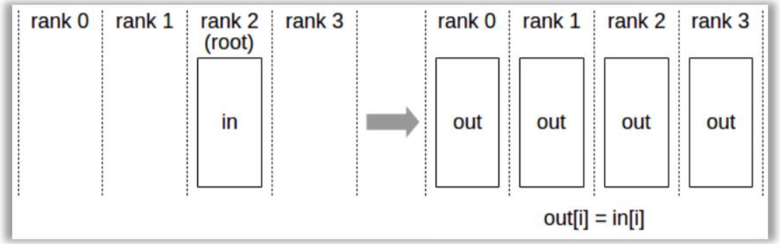
- NVIDIA Collective Communications Library (NCCL)

ncclAllReduce()

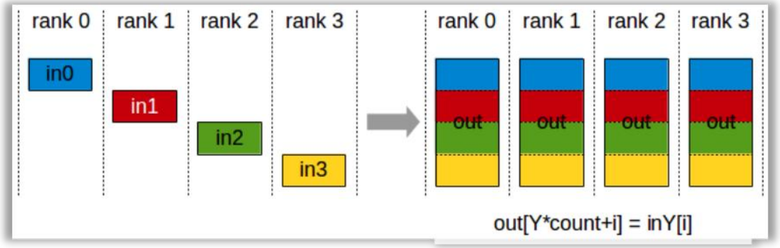


ncclReduce

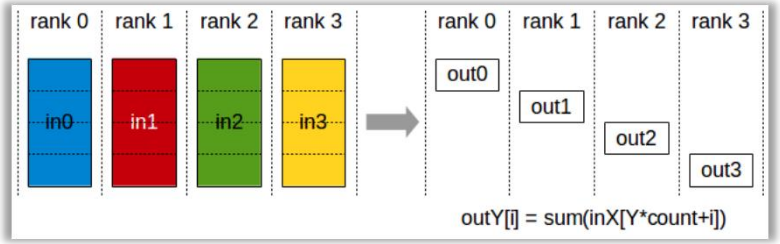




ncclBroadcast()



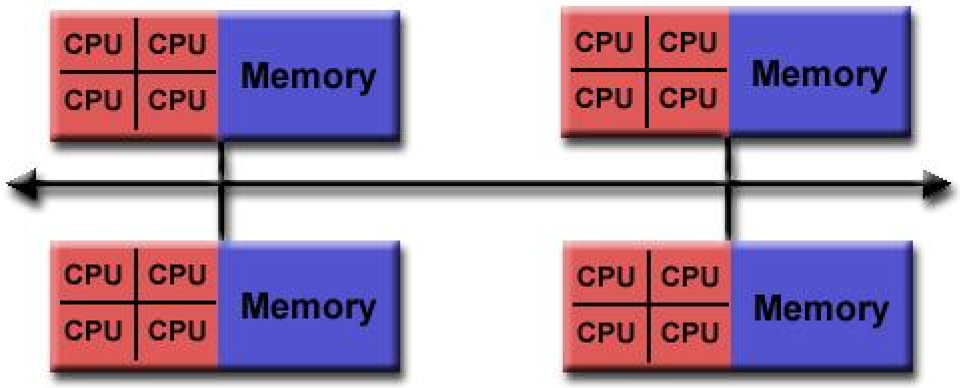
ncclAllGather()



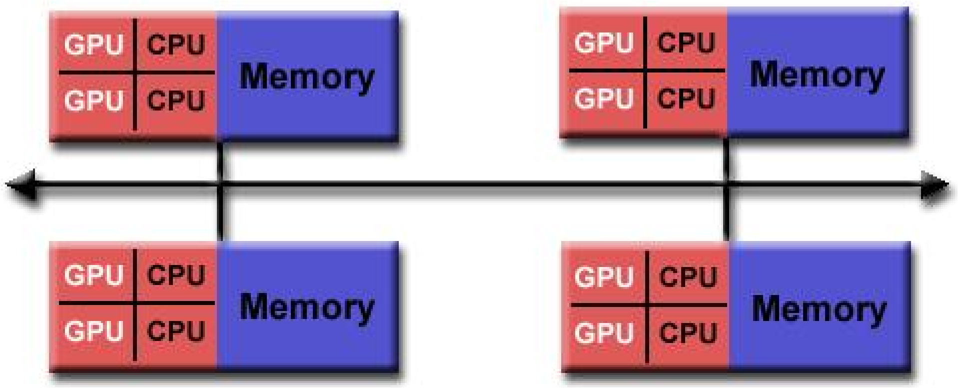
ncclReduceScatter()

Parallelization in Distributed-Memory System?

Hybrid & Homogeneous



Hybrid & Heterogeneous



Source: Blaise Barney (LLNL), Introduction to Parallel Computing

Inter-Process Communications within/across System

- within a shared-memory system
  - POSIX shared memory
  - POSIX message queues
  - etc.
- across shared-memory systems
  - POSIX socket
  - **message passing**
  - RPC (w/ protobuf, etc.)
  - REST API
  - etc.

Distributed Shared Memory

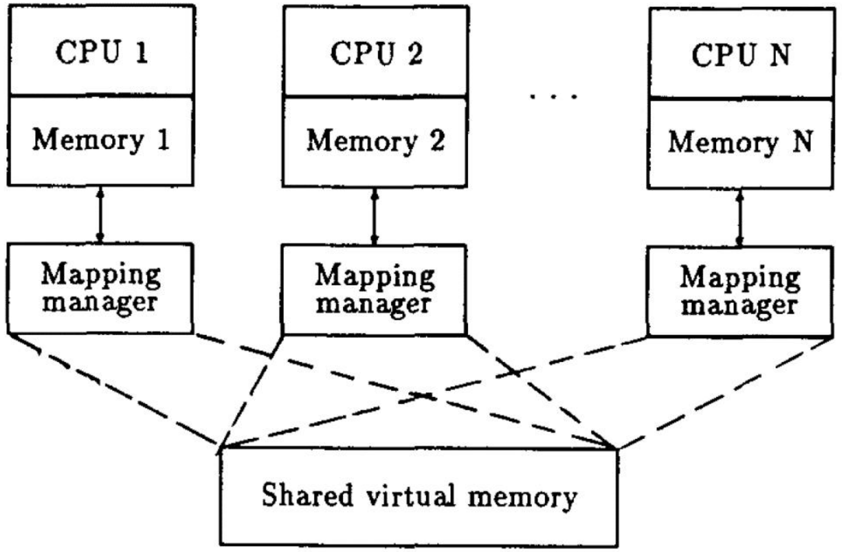
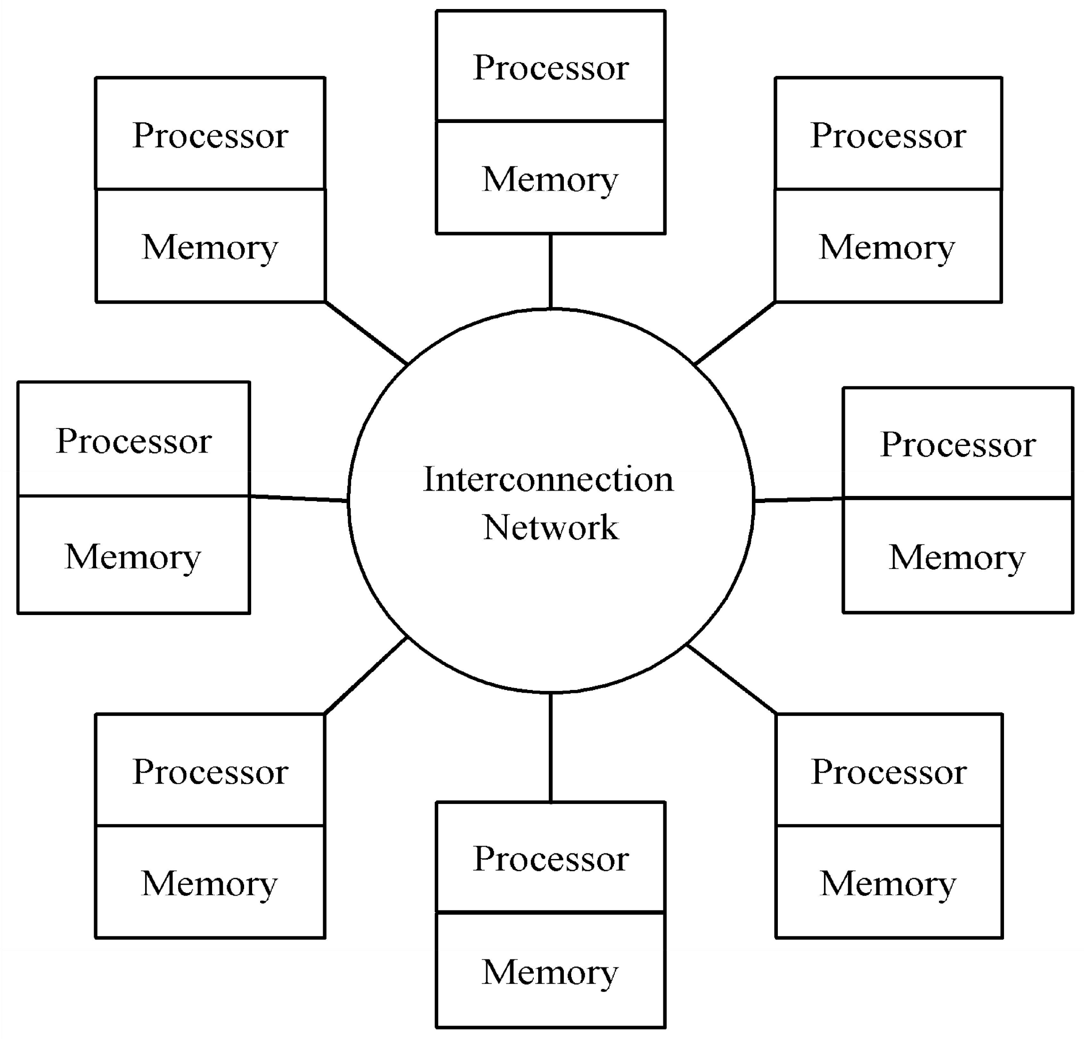


Table I. Spectrum of Solutions to the Memory Coherence Problem

	Page ownership strategy			
		Dynamic		
			Distributed manager	
Page synchronization method	Fixed	Centralized manager	Fixed	Dynamic
Invalidation	Not allowed	Okay	Good	Good
Write-broadcast	Very expensive	Very expensive	Very expensive	Very expensive

K. Li and P. Hudak, “Memory coherence in shared virtual memory systems,” ACM Trans. Comput. Syst., vol. 7, no. 4, pp. 321–359, Nov. 1989.

Message-Passing Model



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

The Message Passing Interface

- Late 1980s: vendors had unique libraries
- 1989: Parallel Virtual Machine (PVM) developed at Oak Ridge National Lab
- MPI Standards
  - 1992: Work on standard begun
  - 1994: Version 1.0
  - 1997/2008/2009: Version 2.0/2.1/2.2
  - 2012/2015: Version 3.0/3.1
  - 2021/2023: Version 4.0/4.1
  - WIP: Version 5.0
- Today: MPI is dominant massage passing library standard

https://www.mpi-forum.org/

MPI: Hello World!

```c
#include <mpi.h>

int main(int argc, char *argv[])
{
    int npes, myrank;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &npes);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    printf("From process %d out of %d, Hello World!\n",
        myrank, npes);
    MPI_Finalize();
    return 0;
}
```

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

MPI: the Message Passing Interface

The minimal set of MPI routines.

`MPI_Init`	Initializes MPI.
`MPI_Finalize`	Terminates MPI.
`MPI_Comm_size`	Determines the number of processes.
`MPI_Comm_rank`	Determines the label of calling process.
`MPI_Send`	Sends a message.
`MPI_Recv`	Receives a message.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Compiling and Running MPI Programs

- Compiling examples
  - `mpicc -o foo foo.c`
  - `mpic++ -o bar bar.cpp`
- Running examples
  - `mpirun -np 4 foo`
  - `mpirun -np 2 foo : -np 4 bar`
- Specifying host processors
  - see “--hostfile” and “--host” options

Better Understanding of MPI (1/5)

- Compilation
  - mpicc or mpic++ is just a wrapper
  - Try “mpicc --show” in Open MPI or MPICH. For example,

```
gcc -I/usr/lib64/mpi/gcc/openmpi/include/openmpi
-I/usr/lib64/mpi/gcc/openmpi/include –pthread
-L/usr/lib64/mpi/gcc/openmpi/lib64 -lmpi -lopen-rte -lopen-pal
-ldl -Wl,--export-dynamic -lnsl -lutil -lm –ldl
```

- Execution
  - mpirun or mpiexec. For example,

```
> mpirun -np 8 ./a.out
```

Better Understanding of MPI (2/5)

```c
/* hello.c */
#include <stdio.h>
#include <mpi.h>

void main(int argc, char *argv[]) {
  MPI_Status status;
  int rank, size;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(
    MPI_COMM_WORLD, &rank);
  MPI_Comm_size(
    MPI_COMM_WORLD, &size);

  /* see code segments on right */

  MPI_Finalize();
}
```

```c
char message[20];
int tag=11;

if (rank == 0) {
  strcpy(message, "Hello,World!");
  for (i=1; i<size; i++)
  MPI_Send(message, 13, MPI_CHAR,
     i, tag, MPI_COMM_WORLD);
}

else {
  MPI_Recv(message, 20, MPI_CHAR,
    0, tag, MPI_COMM_WORLD,
    &status);
  printf( "Process %d : %.13s\n",
    rank, message);
}
```

Better Understanding of MPI (3/5)

```c
/* master.c */

#include <stdio.h>
#include <mpi.h>

void main(int argc, char *argv[]) {
  MPI_Status status;
  MPI_Init(&argc, &argv);

  char message[20];
  int i, tag=11;

  strcpy(message, "Hello,World!");
  for (i=1; i<size; i++)
    MPI_Send(message, 13, MPI_CHAR,
      i, tag, MPI_COMM_WORLD);

  MPI_Finalize();
}
```

```c
/* slave.c */

#include <stdio.h>
#include <mpi.h>

void main(int argc, char *argv[]) {
  MPI_Status status;
  MPI_Init(&argc, &argv);

  char message[20];
  int i, tag=11;

  MPI_Recv(message, 20, MPI_CHAR,
    0, tag, MPI_COMM_WORLD,
    &status);
  printf( "Process %d : %.13s\n",
    rank,message);

  MPI_Finalize();
}
```

Better Understanding of MPI (4/5)

- Single program multiple data (SPMD) mode

```
> mpirun -np 8 ./hello
```

- Multiple program multiple data (MPMD) mode

```
> mpirun -np 1 ./master : -np 7 ./slave
```

Better Understanding of MPI (5/5)

- Example hostfle

  host_ada slots=2 max_slots=8

  host_barbara slots=2 max_slots=8

- mpirun -hostfile <file> -np 3 ./hello
  - 2 processes on host_ada, and 1 process on host_barbara
- mpirun -hostfile <file> -np 4 ./hello
  - 2 processes on host_ada, and 2 process on host_barbara
- mpirun -hostfile <file> -np 5 ./hello
  - 3 processes on host_ada, and 2 process on host_barbara
- mpirun -hostfile <file> -np 17 ./hello
  - There are not enough slots available in the system

MPI vs. Socket APIs

- Send and receive data using socket APIs

```
void Sender(int fd, char *buf, int count) {
  int n;
  while (count > 0) {
    n = write(fd, buf, count);
    if (n < 0) { ... special handling ... }
    count -= n;
    buf += n;
  }
}
void Receiver(int fd, char *buf, int count) {
  int n;
  while (count > 0) {
    n = read(fd, buf, count);
    if (n < 0) { ... special handling ... }
    count -= n;
    buf += n;
  }
}
```

- MPI
  - process startup and shutdown
    - one manager process starts other processes using ssh
    - OR contacts demons to start processes (more scalable)
  - blocking and nonblocking send/recv.
    - MPI_Send, MPI_Recv
    - MPI_ISend, MPI_IRecv, MPI_Test*, MPI_Wait*
  - collective communications

Collective Communication: Introduction

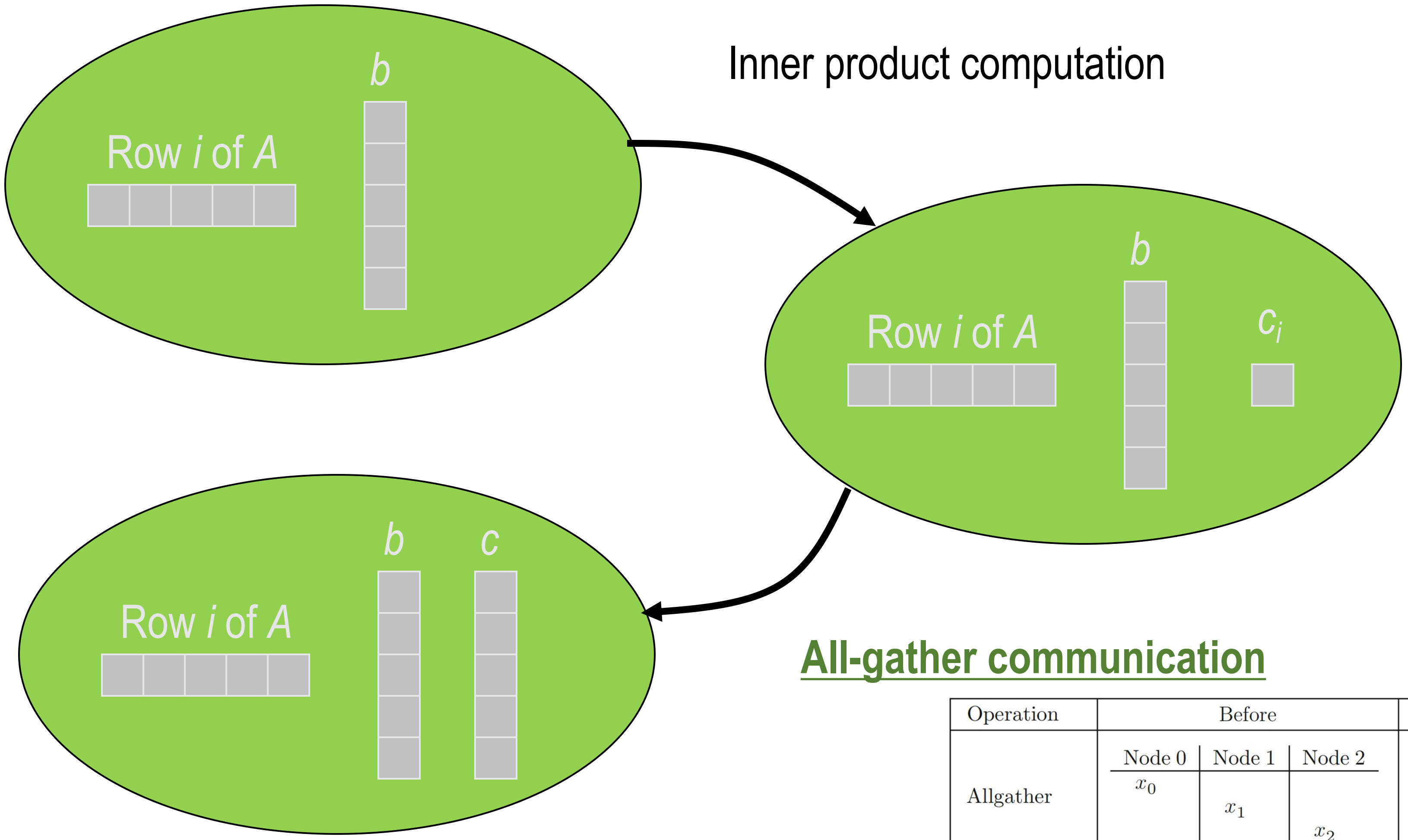
- Collective communication is <u>communication that involves a group of processing elements (nodes) and effects a data transfer between all or some of these processing elements</u>.
  - Data transfer may include the application of a reduction operator or other transformation of the data.
- Collective communication functionality is often exposed through library interfaces or language constructs.
- Collective communication is a natural extension of the message-passing paradigm.
- A widely used parallel interface with an explicit, rich set of collective communication operations is the Message Passing Interface (MPI).

Motivating Example: Matrix-Vector Multiplication

$$
\begin{bmatrix} 2 & 1 & 0 & 4 \\ 3 & 2 & 1 & 1 \\ 4 & 3 & 1 & 2 \\ 3 & 0 & 2 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 3 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 14 \\ 19 \\ 11 \end{bmatrix}
$$

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

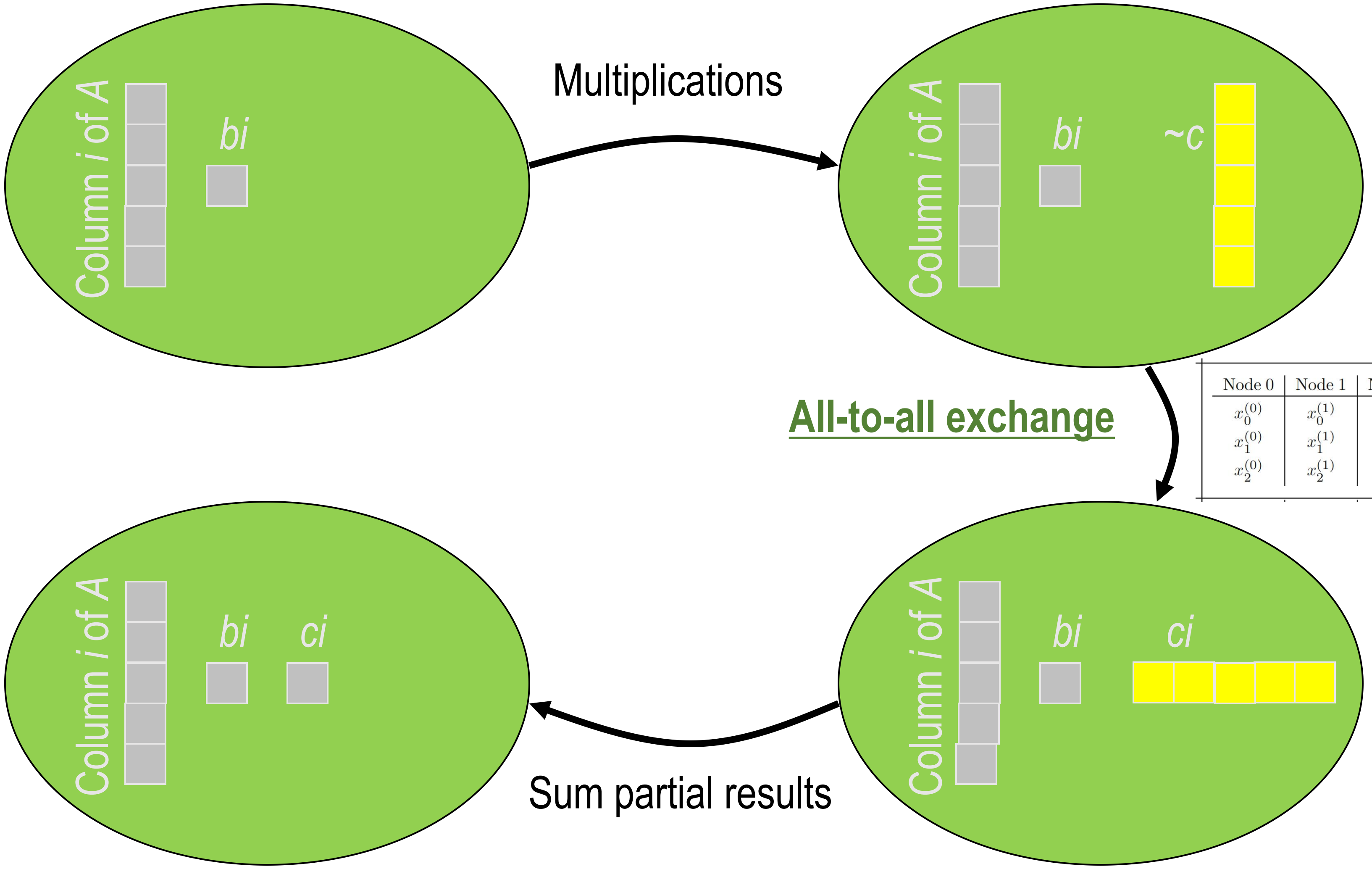
Phases of Row-Partitioned Parallel Algorithm



Operation	Before			After		
	Node 0	Node 1	Node 2	Node 0	Node 1	Node 2
Allgather	x_0	x_1	x_2	x_0 x_1 x_2	x_0 x_1 x_2	x_0 x_1 x_2

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Phases of Col.-Partitioned Parallel Algorithm



Node 0	Node 1	Node 2	Node 0	Node 1	Node 2
$x_0^{(0)}$	$x_0^{(1)}$	$x_0^{(2)}$	$x_0^{(0)}$	$x_1^{(0)}$	$x_2^{(0)}$
$x_1^{(0)}$	$x_1^{(1)}$	$x_1^{(2)}$	$x_0^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$
$x_2^{(0)}$	$x_2^{(1)}$	$x_2^{(2)}$	$x_0^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Commonly Used Collective Communications

- Data redistribution operations
  - Broadcast
  - Scatter
  - Gather
  - Allgather
  - All-to-all
  - Permuation
- Reduction operations
  - Reduce
  - Allreduce
  - Reduce-scatter
  - All prefix sums
- Barrier synchronization

Data Redistribution Operations

- Rooted redistribution operations
  - assumed that all nodes know the identity of the root node

Operation	Before: Node 0	Before: Node 1	Before: Node 2	After: Node 0	After: Node 1	After: Node 2
Broadcast	x			x	x	x
Scatter	x_0 x_1 x_2			x_0	x_1	x_2
Gather	x_0	x_1	x_2	x_0 x_1 x_2		

- Non-rooted redistribution operations

Operation	Before: Node 0	Before: Node 1	Before: Node 2	After: Node 0	After: Node 1	After: Node 2
Allgather	x_0	x_1	x_2	x_0 x_1 x_2	x_0 x_1 x_2	x_0 x_1 x_2
All-to-all	$x_0^{(0)}$ $x_1^{(0)}$ $x_2^{(0)}$	$x_0^{(1)}$ $x_1^{(1)}$ $x_2^{(1)}$	$x_0^{(2)}$ $x_1^{(2)}$ $x_2^{(2)}$	$x_0^{(0)}$ $x_0^{(1)}$ $x_0^{(2)}$	$x_1^{(0)}$ $x_1^{(1)}$ $x_1^{(2)}$	$x_2^{(0)}$ $x_2^{(1)}$ $x_2^{(2)}$
Permutation	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(\pi^{-1}(0))}$	$x^{(\pi^{-1}(1))}$	$x^{(\pi^{-1}(2))}$

Reduction Operations

Operation	Before			After		
	Node 0	Node 1	Node 2	Node 0	Node 1	Node 2
Reduce	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$y = \oplus_{j=0}^{p-1} x(j)$		
Reduce-scatter	$x_0^{(0)}$ $x_1^{(0)}$ $x_2^{(0)}$	$x_0^{(1)}$ $x_1^{(1)}$ $x_2^{(1)}$	$x_0^{(2)}$ $x_1^{(2)}$ $x_2^{(2)}$	$y_0 = \oplus_{j=0}^{p-1} x_0^{(j)}$	$y_1 = \oplus_{j=0}^{p-1} x_1^{(j)}$	$y_2 = \oplus_{j=0}^{p-1} x_2^{(j)}$
Allreduce	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$y = \oplus_{j=0}^{p-1} x^{(j)}$	$y = \oplus_{j=0}^{p-1} x^{(j)}$	$y = \oplus_{j=0}^{p-1} x^{(j)}$
Prefix	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$y^{(0)} = \oplus_{j=0}^{0} x^{(j)}$	$y^{(1)} = \oplus_{j=0}^{1} x^{(j)}$	$y^{(2)} = \oplus_{j=0}^{2} x^{(j)}$

Broadcast (a.k.a., One-to-All Boardcast)

- Among a group of processing elements (nodes), a designated root node has a data item to be communicated (copied) to all other nodes.

Before			After		
Node *r*	Node 1	Node 2	Node *r*	Node 1	Node 2
x			*x*	x	*x*

- All nodes are assumed to explicitly take part in the broadcast operation.
- It is generally assumed that before its execution all nodes know the index of the designated root node as well as the the amount n of data to be broadcast.
- The data item x may be either a single, atomic unit or divisible into smaller, disjoint pieces.
  - The latter can be exploited algorithmically when n is large.

Broadcast: Assumptions

- Assumptions in the following analysis (for this lecture ONLY)
  - All nodes can communicate through a communication network.
  - Individual nodes perform communication operations that send and/or receive individual messages.
  - Communication is through a single port

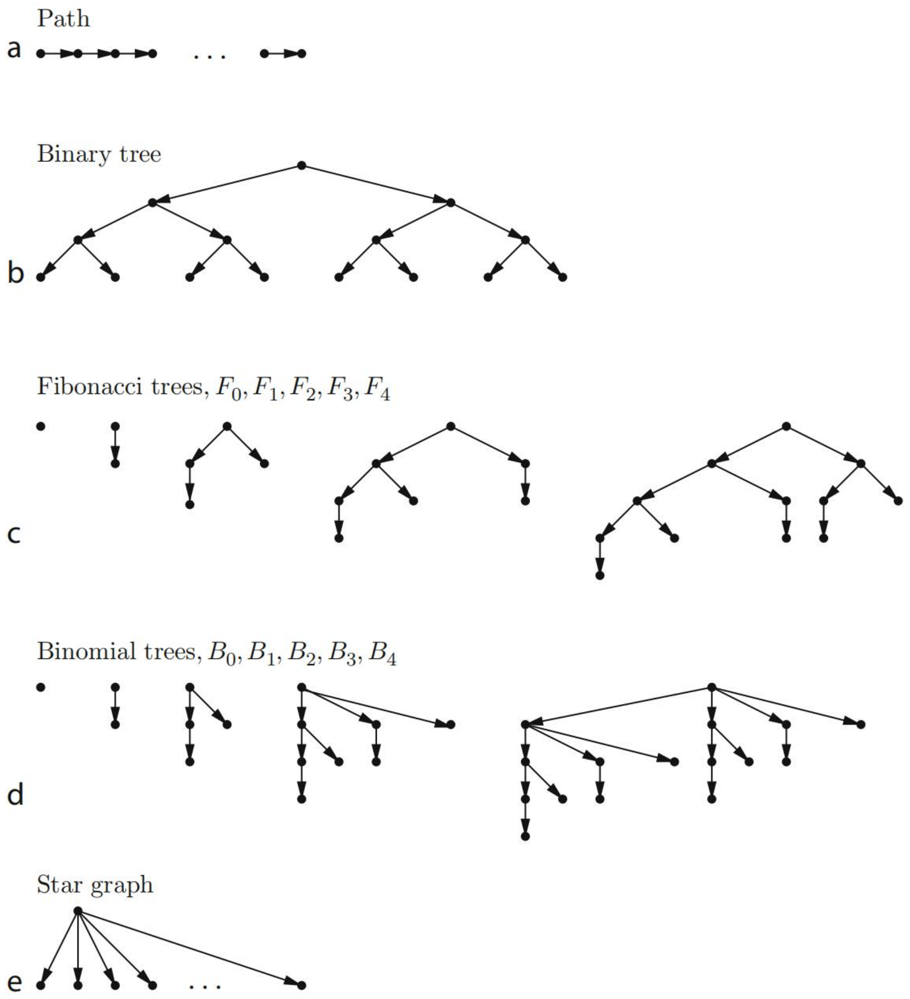
    (such that a node can be involved in at most one communication operation at a time)
    - Such an operation can be either a send to or a receive from another node (unidirectional communication),
    - a combined send to and receive from another node (bidirectional, telephone like communication), or
    - a send to and receive from two possibly different nodes (simultaneous sendreceive, fully bidirectional communication).
  - The communication medium is homogeneous and fully connected
  - A first approximation for the transferring time between two nodes: $\alpha + n\beta$
    - n is the message size
    - α is the start-up cost (latency)
    - β is the cost per item transfered (inverse of the bandwidth)

Broadcast: Two Lower Bounds

- Two lower bounds
  - for the α term: $\lceil \log_2 p \rceil \alpha$
    - a round: during which each node can send at most one message and receive at most one message.
    - In each round, the number of nodes that know message x can at most double.
    - Thus, a minimum of $\lceil \log_2 p \rceil$ rounds are needed to broadcast the message.
    - Each round costs at least α.
  - for the β term: $n\beta$
    - If $p > 1$ then the message must leave the root node, requiring a time of at least $n\beta$.
- Notes
  - When assumptions about the communication system change, these lower bounds change as well.
  - Lower bounds for mesh and torus networks, hypercubes, and many other topologies are known[WGe95].
  - Determining the minimum broadcast time in an arbitrary, given graph is NP-hard [JaM95]

[WGe95] Watts J, van de Geijn RA (1995) A pipelined broadcast for multi-dimensional meshes. Parallel Process Lett 5: 281–292
[JaM95] Jansen K, Müller H (1995) The minimum broadcast time problem for several processor networks. Theor Comput Sci 147(1&2): 69–85

Broadcast: Tree-based Algorithms



Broadcast: The "MST" Algorithm

- The so-called* Minimum Spanning Tree (MST) algorithm
  - Partition the set of nodes into two roughly equalsized subsets.
  - Send x from the root to a node in the subset that does not include the root.
    - The receiving node will become a local root in that subset.
  - Recursively broadcast x from the (local) root nodes in the two subsets.
- The total cost of the MST algorithm: $\lceil \log_2 p \rceil (\alpha + n\beta)$
  - It achieves the lower bound for the α term
  - but not for the β term for which it is a logarithmic factor off from the lower bound.
- The MST algorithm constructs a binomial spanning tree over the set of nodes
  - Figure (c) in the previous slide

* which is a misnomer, since all broadcast trees are spanning trees and minimum for homogeneous communication media

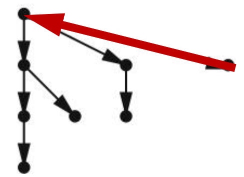
Broadcast: Pipelining

- For large n, pipelining improves the broadcast cost
  - the message is split into k blocks of size n/k each
- In the "path" boradcast tree:
  - The first piece from node 0 takes p-1 rounds to reach node p-1
  - an additional k-1 rounds for the remaining k-1 pieces
  - total time cost = $(k+p-2)(\alpha+\beta n/k) \geq (p-2)\alpha + 2\sqrt{(p-2)n\alpha\beta} + \beta n$
    - with optimal number of blocks $k = \sqrt{(p-2)\beta n/\alpha}$
- Pipelining can be applied on fixed degree trees
  - Binary tree: best-effort time cost = $2(\log_2 p - 1)\alpha + 4\sqrt{(\log_2 p - 1)n\alpha\beta} + 2\beta n$
  - Fibonacci tree: best-effort time cost = $(\log_\Theta p - 1)\alpha + 2\sqrt{2(\log_\Theta p - 1)n\alpha\beta} + 2\beta n$ where $\Theta = \frac{\sqrt{5}+1}{2}$
- Pipelining is not helpful on nonconstant degrees trees (e.g., binomial tree and star graph)

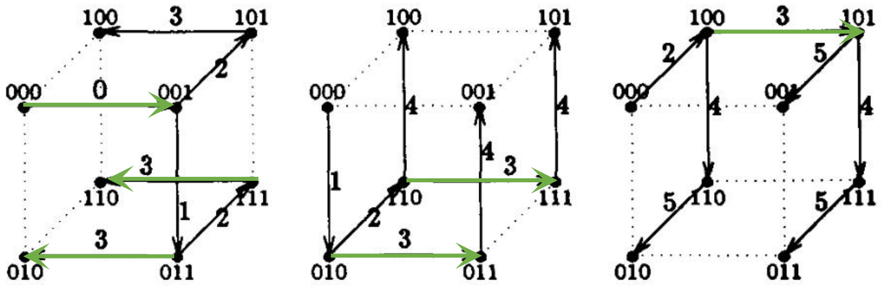
Broadcast: the Third Lower Bound

- The third lower bound on the number of rounds: $k - 1 + \lceil \log_2 p \rceil$
  - first piece $\lceil \log_2 p \rceil$ rounds plus k-1 additional rounds
  - total time cost = $(k - 1 + \lceil \log_2 p \rceil)(\alpha + \beta n/k) \ \geq \ (\log_2 p - 1)\alpha + 2\sqrt{(\log_2 p - 1)\alpha}\sqrt{\beta n} + \beta n$
    - with optimal number of blocks $k = \sqrt{(\log_2 p - 1)\beta n/\alpha}$

Broadcast: Simultaneous Trees



- optimal $k - 1 + \log_2 p$ rounds of pipelining
  - when the message is split into k blocks
- $p=2^d$ nodes in a d-dimensional hypercube
  - node id: $i \in \{ 0, 1, ..., 2^d-1 \}$ with i_j as the j-th bit
- d edge-disjoint spanning binomial trees (ESBT)
  - all trees are rooted at node 0
  - the j-th tree has $2^{d-1}-1$ leaves with $i_j=0$
  - the j-th tree broadcasts blocks j, j+d, j+2d, …
- Leaves in tree j = t%d receive block t-d in round d
  - block -d, 1-d, …, -1, 0, 1, …, k-1
- BitDistance$_i$[j]: distance to the next 1 to the left of i_j
  - BitDistance$_0$[k] ≡ k for $0 < k \le d$



SBT_0 SBT_1 SBT_2

```
f ← ((k mod d) + d − 1) mod d /* Start round for first phase */
t ← 0
while t < k + d − 1 do
    /* New phase consisting of (up to) d rounds */
    for j ← f to d − 1
        s ← t − d + (1 − i_j) * BitDistance_i[j] /* block to send */
        r ← t − d + i_j * BitDistance_i[j]       /* block to receive */
        if s ≥ k then s ← k − 1
        if r ≥ k then r ← k − 1
        par /* simultaneous send-receive with neighbor */
            if s ≥ 0 then Send block s to node (i xor 2^j)
            if r ≥ 0 then Receive block r from node (i xor 2^j)
        end par
        t ← t + 1 /* next round */
    end for
    f ← 0 /* next phases start from 0 */
endwhile
```

S. Lennart Johnsson and C. T. Ho, “Optimum Broadcasting and Personalized Communication in Hypercubes,” IEEE TC, 1989.

Broadcast in a Real Library

- gloo
  - https://github.com/facebookincubator/gloo/
  - Collective communications library with various primitives for multi-machine training

Broadcast

Broadcast contents of buffer on one process to other P-1 processes.

broadcast_one_to_all

- Communication steps: 1
- Bytes on the wire: P*S

Non-root processes: receive buffer from root.

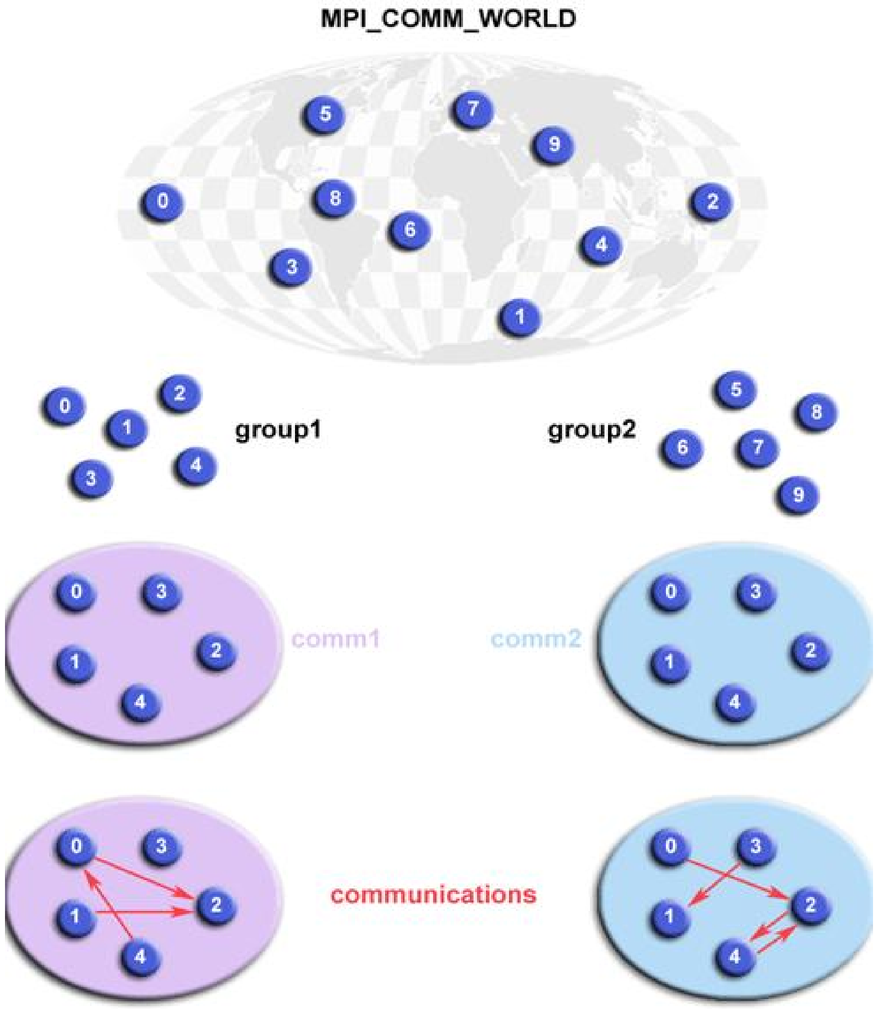
Root process: send buffer to P-1 processes.

main gloo / gloo / broadcast.cc Top

Code Blame Raw

```cpp
void broadcast(BroadcastOptions& opts) {
  size_t numSends = 0;

  // Create mask with all 1's where we progressively set bits to 0
  // starting with the LSB. When the mask applied to the virtual rank
  // equals 0 we know the process must participate. This results in
  // exponential participation starting with virtual ranks 0 and 1.
  size_t mask = (1 << dim) - 1;

  for (size_t i = 0; i < dim; i++) {
    // Clear bit `i`. In the first iteration, virtual ranks 0 and 1 participate.
    // In the second iteration 0, 1, 2, and 3 participate, and so on.
    mask ^= (1 << i);
    if ((vrank & mask) != 0) {
      continue;
    }

    // The virtual rank of the peer in this iteration has opposite bit `i`.
    auto vpeer = vrank ^ (1 << i);
    if (vpeer >= vsize) {
      continue;
    }

    // Map virtual rank of peer to actual rank of peer.
    auto peer = (vpeer + opts.root) % vsize;
    if ((vrank & (1 << i)) == 0) {
      in->send(peer, slot);
      numSends++;
    } else {
      out->recv(peer, slot);
      out->waitRecv(opts.timeout);
    }
  }

  // Copy local input to output if applicable.
  if (context->rank == opts.root && in != out) {
    memcpy(out->ptr, in->ptr, out->size);
  }

  // Wait on pending sends.
  for (auto i = 0; i < numSends; i++) {
    in->waitSend(opts.timeout);
```

Groups and Communicators in MPI

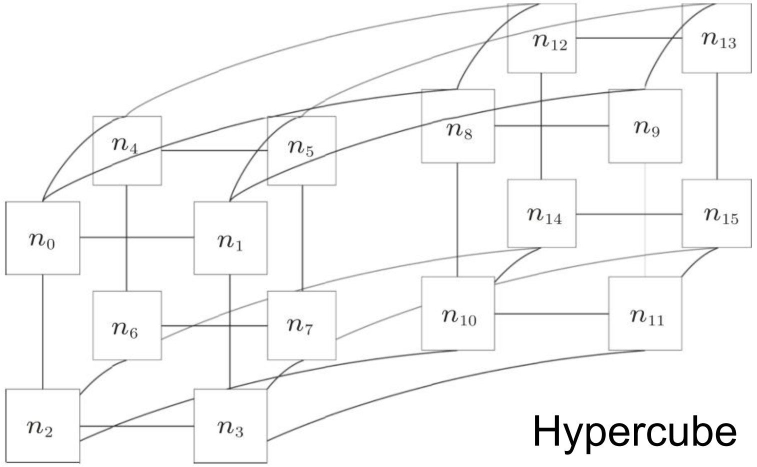


- Related MPI functions
  - Form new group as a subset of global group using *MPI_Group_incl*
  - Create new communicator for new group using *MPI_Comm_create*
  - Determine new rank in new communicator using *MPI_Comm_rank*
  - Conduct communications using any MPI message passing routine
  - When finished, free up new communicator and group (optional) using *MPI_Comm_free* and *MPI_Group_free*

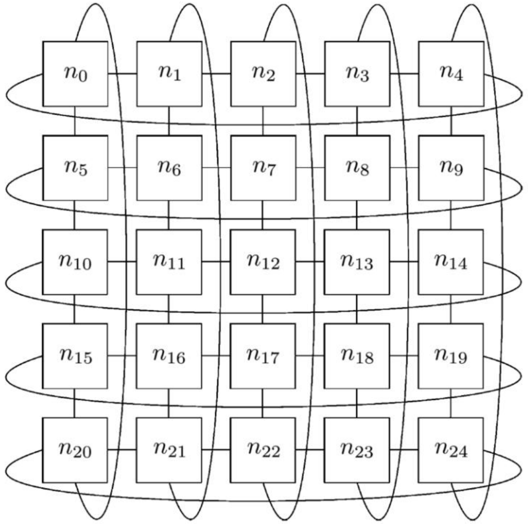
B. Barney, "Message Passing Interface (MPI)", LLNL.

Physical Network Topologies

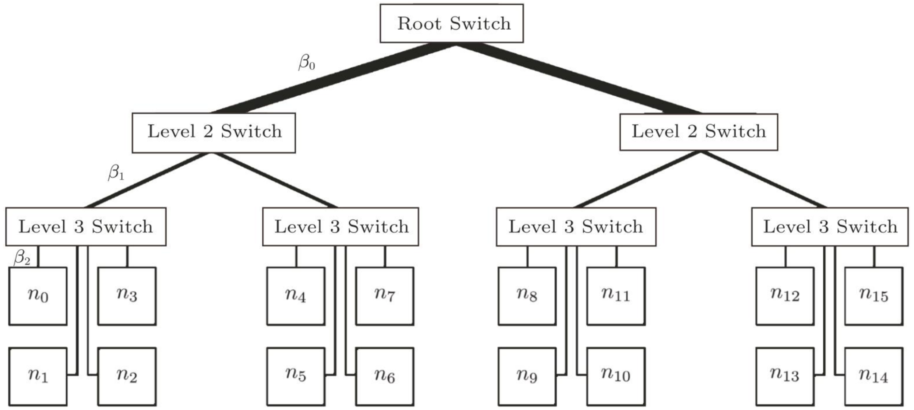
- Ring
- Hypercube
- Torus
- Fat Trees
- Dragonfly(+)
- ...



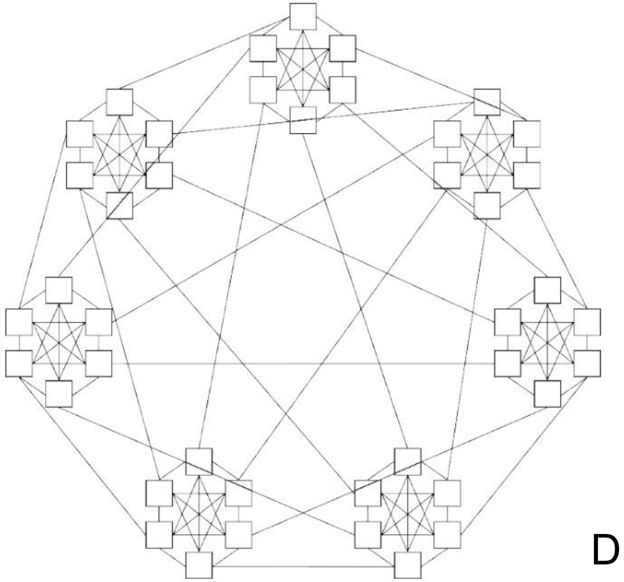
Hypercube



2D Torus



Fat Tree



Dragonfly

image source: A. Weingram et al., "xCCL: A Survey of Industry-Led Collective Communication Libraries for Deep Learning," JCST 2023.

Universal Routers

- Fat-tree U(A) emulates arbitary router A
  - within area $A_U \in O(A \log^2 A)$ and routing time in $O(\lambda \log^2 A)$, where λ is the load factor

Universal routers				
Topology	**Mode**	**Blowup**	**Routing time**	**Ref**
Concentrator Fat-Tree (CFT)	off/det	$O(\log^2 A)$	$O(\lambda \log^2 A)$	[8]
CFT	on/rand	$O(\log^2 A)$	$O(\lambda \log A +$ $+ \log^2 A \log\log A)$	[9]
CFT (⊖:word model)	on/rand	$O(\log^2 A)$	$O(\lambda + \log A)$	[10]
Pruned Butterfly	on/det	$O(\log^2 A)$	$O(\lambda \log^2 A)$	[11]
Sorting FT (⊖: constant-degree message sets)	on/det	$O(\log^2 A)$	$O(\lambda \log A + \log^2 A)$	[11]
Fat Pyramid (⊖:$O(A/\log A)$ terminals)	off/det	$O(1)$	$O(\lambda + \log A)$ (⊕:general delay model)	[12]

C. E. Leiserson, “Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing.,” Proc. Int. Conf. Parallel Process., pp. 393–402, 1985.

Summary

- Distributed memory architecture
- Fundamentals of Message passing
- Broadcast, a collective communication operation

Further Readings

- MPI standard, history and tutorials
  - MPI Forum http://www.mpi-forum.org
  - Implementations: MPICH2, Open MPI, etc.
  - Gropp, W. (2011). MPI (Message Passing Interface). In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_222
  - Tutorials: https://computing.llnl.gov/tutorials/mpi/ and http://www.mcs.anl.gov/research/projects/mpi/
- Boardcast, one of the collective communication operations
  - van de Geijn, R., Träff, J. (2011). Collective Communication. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_28
  - Träff, J.L., van de Geijn, R.A. (2011). Broadcast. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_29