MPI – Message Passing Interface

- MPI is used for distributed memory parallelism (communication between nodes of a cluster)
- Interface specification with many implementations
- Portability was a major goal
- Widespread use in parallel scientific computing
- Six basic MPI functions
  - MPI_Init, MPI_Finalize,
  - MPI_Comm_size, MPI_Comm_rank,
  - MPI_Send, MPI_Recv
- Many other functions...

MPI on jinx

- Add to your .bash_profile
- export PATH=$PATH:/usr/lib64/openmpi/bin
- export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64:/usr/lib64/openmpi/lib

- Compile using **mpicc**
- Run using (use full path explicitly if you get “orted” error)
- **/usr/lib64/openmpi/bin/mpirun -np 2 progname**
- **/usr/lib64/openmpi/bin/mpirun -np 2 -hostfile hostfile progname**
- For now, best to do the above in an interactive session, but the “right” way is to create a “batch” job and submit the job (using qsub) from the login node.
- https://support.cc.gatech.edu/facilities/instructional-labs/how-to-run-jobs-on-the-jinx-cluster

Essential MPI Topics

- Point-to-point communication
  - Blocking and nonblocking communication
- Collective communication

Blocking Send and Recv

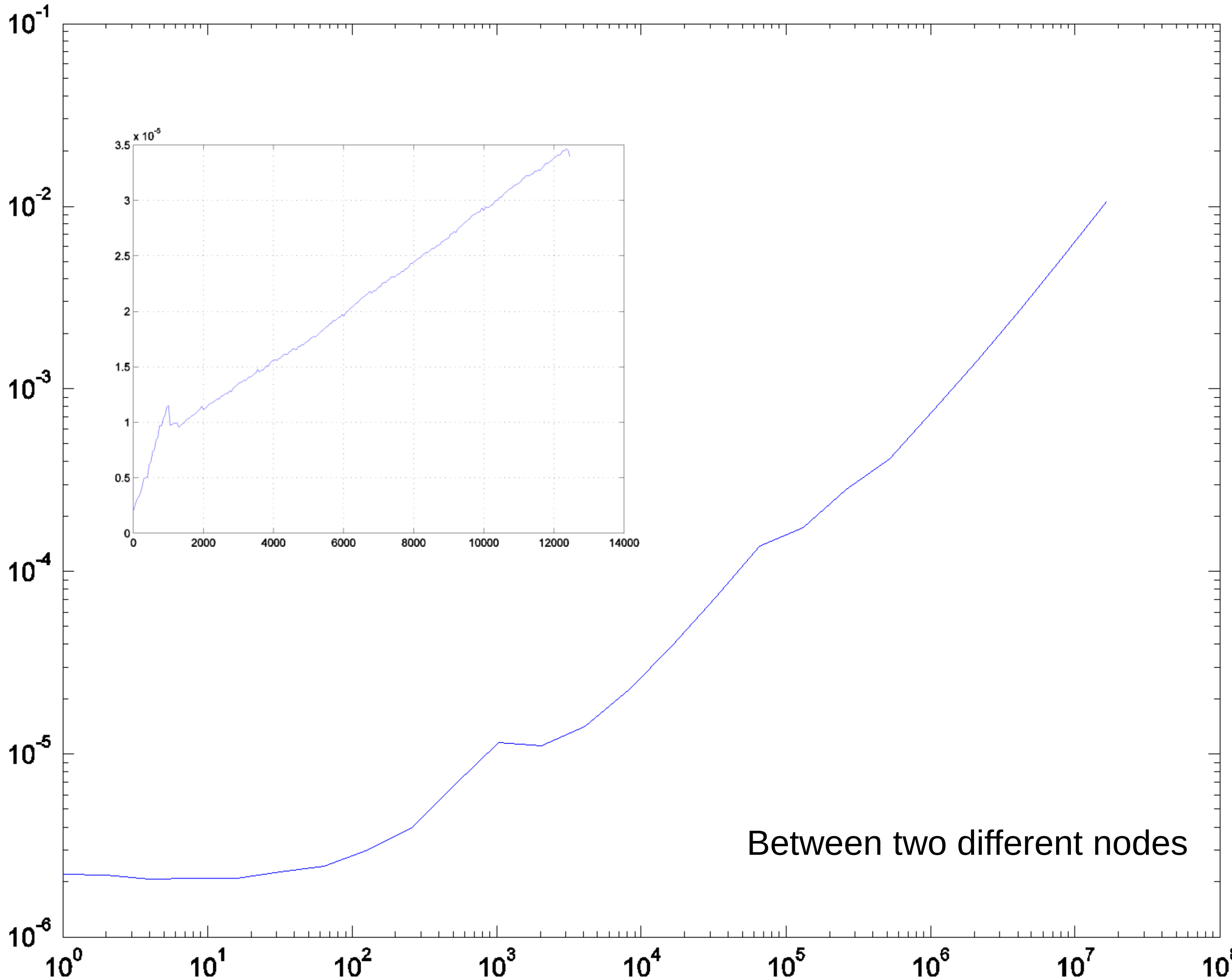
- MPI_Send
  - Function does not return until send buffer can be reused
  - Does not imply the message has been sent
  - Must be assured that the receiver posts a receive call
- MPI_Recv
  - Function does not return until recv buffer contains received message
- Deadlock example (will deadlock if no buffering)

Send(dest=1)	Send(dest=0)
Recv(dest=1)	Recv(dest=0)

Non-blocking Send and Recv

- MPI_Isend
  - Function returns immediately; the data may be buffered, and the message may not be sent yet
- MPI_Irecv
  - Function returns immediately; the message has not necessarily arrived
- MPI_Wait
  - Block until Isend/Irecv completes (buffer can only be used at this point)
- Allows overlap of communication with computation
- Easier to avoid deadlocks than using blocking calls
- Can combine blocking and non-blocking calls

Irecv/Send Communication Time

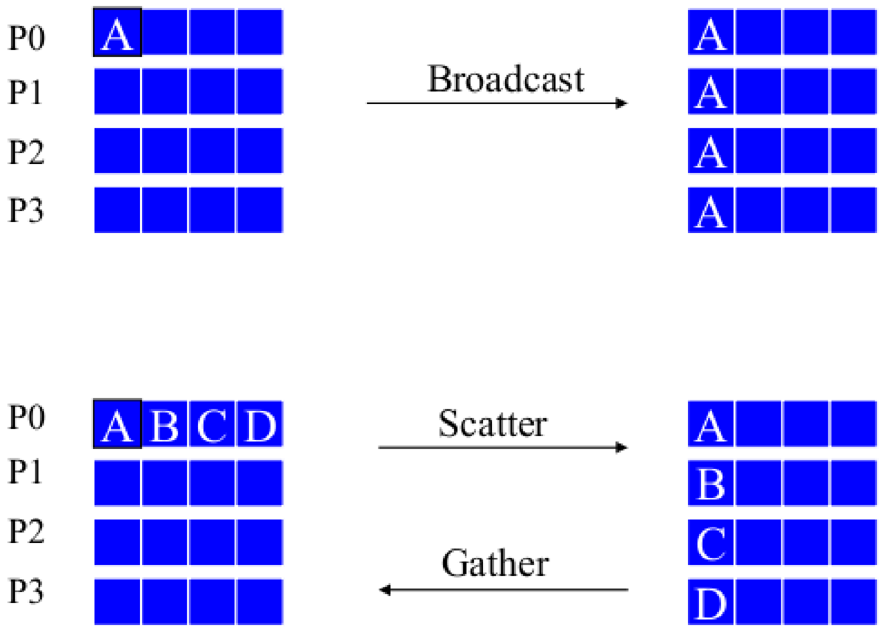


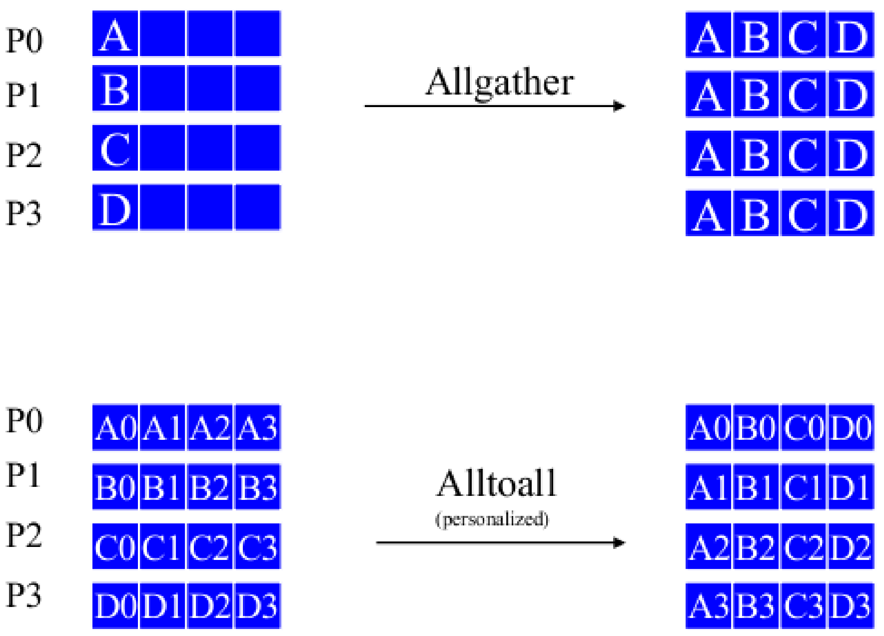
Eager and Rendezvous Protocols

- Eager protocol: if the message is short, it is sent immediately and buffered on the receiver's side. On the receiver, the message is copied to the receive buffer when the receive is posted.
- Rendezvous protocol: if the message is long, a short message is first sent to the receiver to indicate that a send has been posted. The receiver sends the address of the receive buffer. The sender then sends the actual message.
- "kink" in previous graph is due to the switchover from eager to rendezvous protocols

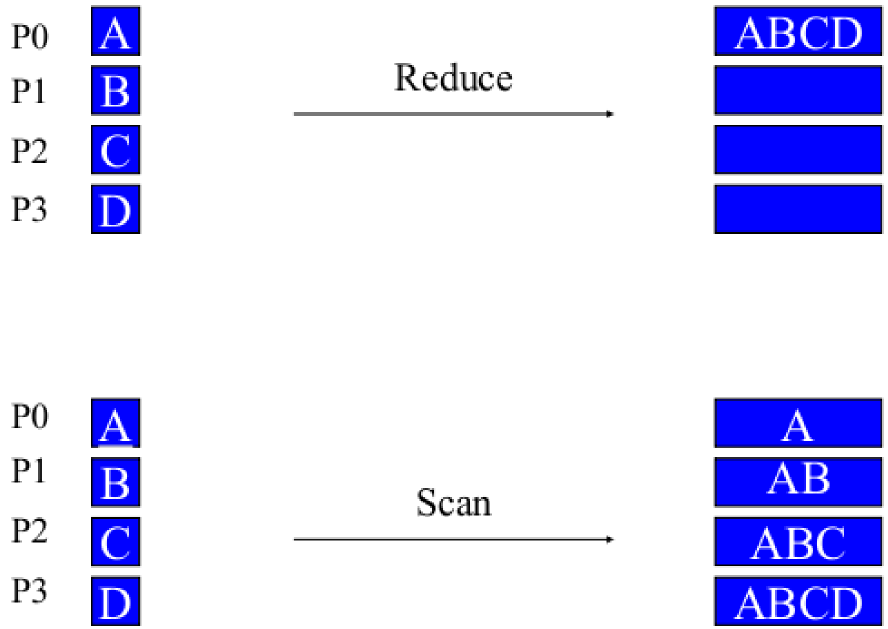
Collective Communication

- MPI_Barrier
- MPI_Bcast
- MPI_Scatter
- MPI_Gather
- MPI_Allgather
- MPI_Alltoall
- MPI_Reduce
- MPI_Allreduce


P0
P1
P2
P3
A
Broadcast
A
A
A
A
P0
P1
P2
P3
A B C D
Scatter
Gather
A
B
C
D



P0
P1
P2
P3
A
B
C
D
Allgather
A B C D
A B C D
A B C D
A B C D
P0
P1
P2
P3
A0 A1 A2 A3
B0 B1 B2 B3
C0 C1 C2 C3
D0 D1 D2 D3
Alltoall
(personalized)
A0 B0 C0 D0
A1 B1 C1 D1
A2 B2 C2 D2
A3 B3 C3 D3

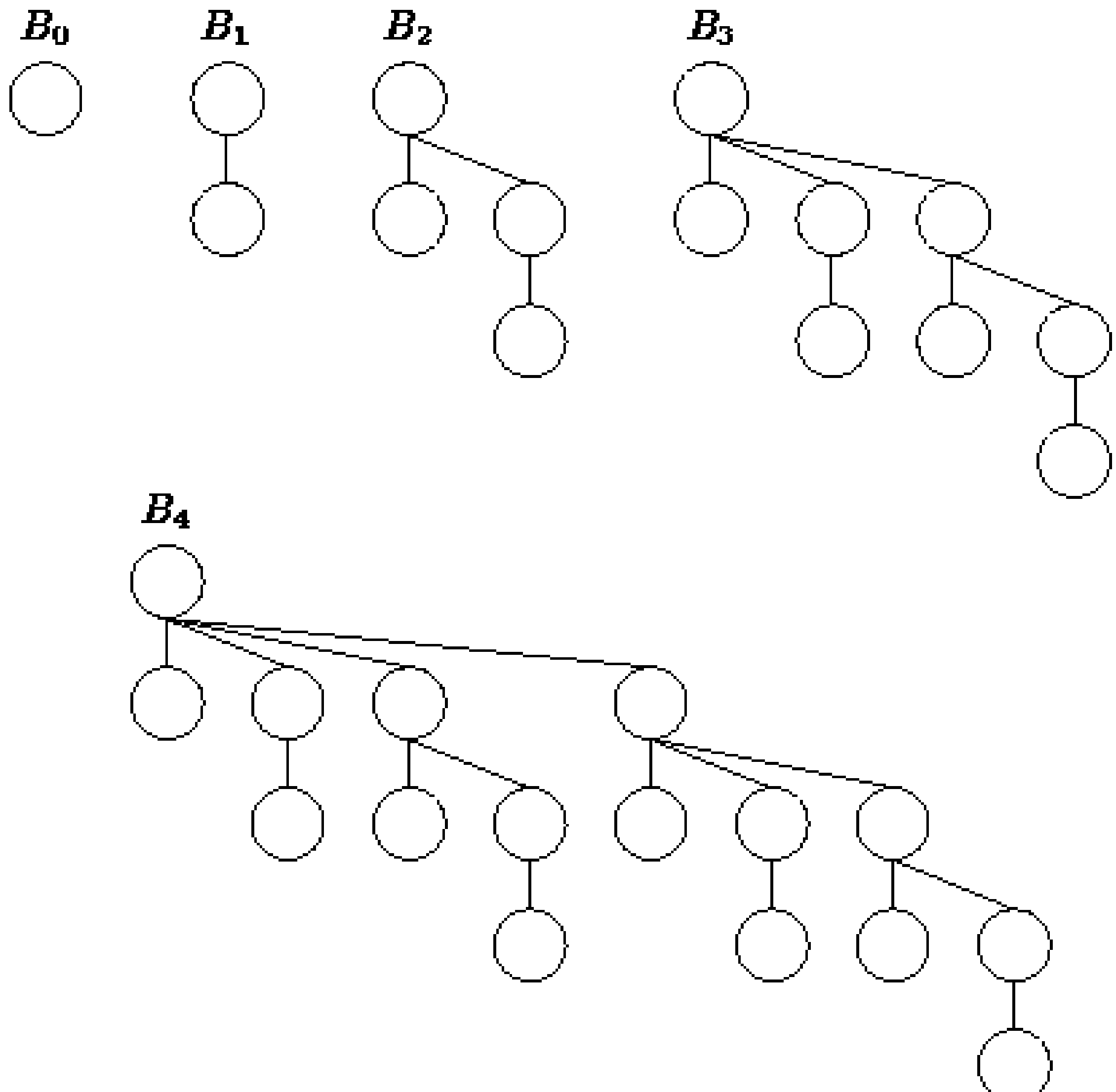

P0
A
P1
B
P2
C
P3
D
Reduce
ABCD
P0
A
P1
B
P2
C
P3
D
Scan
A
AB
ABC
ABCD

Implementation of collective operations

- All-gather
- Broadcast
- Reduce-scatter
- Reduce
- All-reduce (sum on all nodes)

- Reference: Thakur, Rabenseifner, and Gropp, “Optimization of Collective Communication Operations in MPICH,” 2005

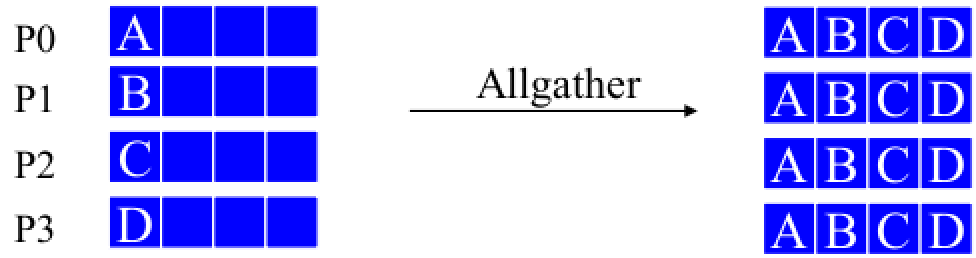
Binomial trees. Log2(p) messages



Model of execution time

- For one message of length n,
  Time = $\alpha + \beta n$
- Assume a node cannot send multiple messages or receive multiple messages at once
- But a node can send a message and receive a message at the same time
- We will see that different algorithms are best, depending on whether messages are short (latency term dominates) or long (bandwidth term dominates)

Allgather

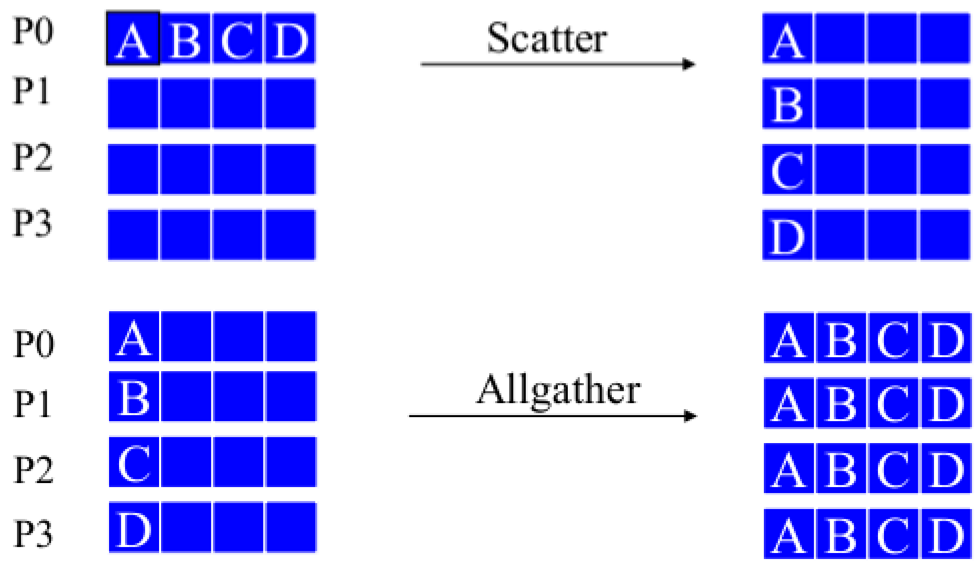


- Data contributed by each process is gathered onto all processes.
- Each process contributes n/p words
- Recursive doubling tree algorithm (for short messages). P0&P1 exch so both have AB; P2&P3 exch so both have CD. P0&P2 exch so both have ABCD; P1&P3 exch both have ABCD. Time = $\alpha \log p + \beta(p-1)n/p$
- Ring algorithm (for long messages). Time = $\alpha(p-1) + \beta(p-1)n/p$
- It seems that recursive doubling should always be faster, but the ring algorithm can effectively have lower α and β on a torus network

Broadcast

- Broadcast n words among p processes
- Two algorithms, depending on message length:
- 1. Binomial tree algorithm (short messages)
  time = $(\log p)(\alpha + n\beta)$
- 2. Scatter then allgather (long messages)
  time = $\alpha(\log p + p - 1) + 2\beta(p-1)n/p$
  (See next slide)
- For large n, scatter then allgather can be faster

Broadcast: scatter then allgather



- Scatter time: $\alpha \log p + \beta(p-1)n/p$
  (recursive halving algorithm)
- Allgather time: $(p-1)\alpha + \beta(p-1)n/p$
  (ring algorithm)
- Total time: $\alpha(\log p + p - 1) + 2\beta(p-1)n/p$

Reduce-Scatter (variant of reduce)

- Reduction of n items. Then each n/p part is scattered to the p processes.
- Basic algorithm: binomial reduce and then linear scatter
- What is the communication time?

Reduce

- Binomial tree (short messages)
  time = $(\log p)(\alpha + n\beta)$
- Reduce-scatter then gather (long messages)
  time = $2\alpha \log p + 2\beta(p-1)n/p$

Allreduce

- Recursive doubling (short messages)
  time = $(\log p)(\alpha + n\beta)$
- Reduce-scatter then allgather (long messages)
  time = $2\alpha \log p + 2\beta(p-1)n/p$