Collectives Algorithms

Mar 5, 2021

Communication Cost Model (For now)

- Message transfer time is modeled as L+n/B, where L is the latency (or startup time) per message, and 1/B is the transfer time per byte, and n the message size in bytes
- All processes can send and receive one message at the same time

Optimization of Collective Communication Operations in MPICH

Rajeev Thakur Rolf Rabenseifner William Gropp

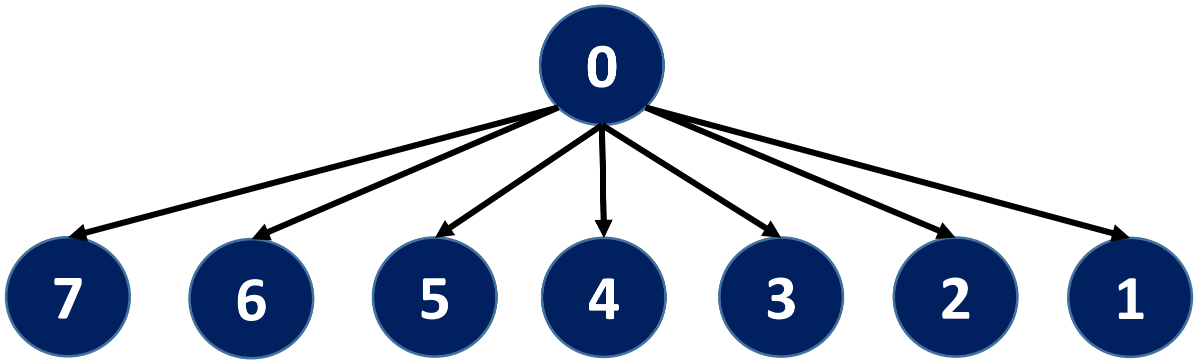
IJHPCA 2005

Broadcast – Naïve/Linear Algorithm

- Root process sends to every other process

Cons

- Root is a bottleneck
- Idling processes
- Communication links are under-utilized



- #Steps for p processes?
  - p - 1
- Transfer time for n bytes
  - T(p) = (p-1) * (L + n/B)

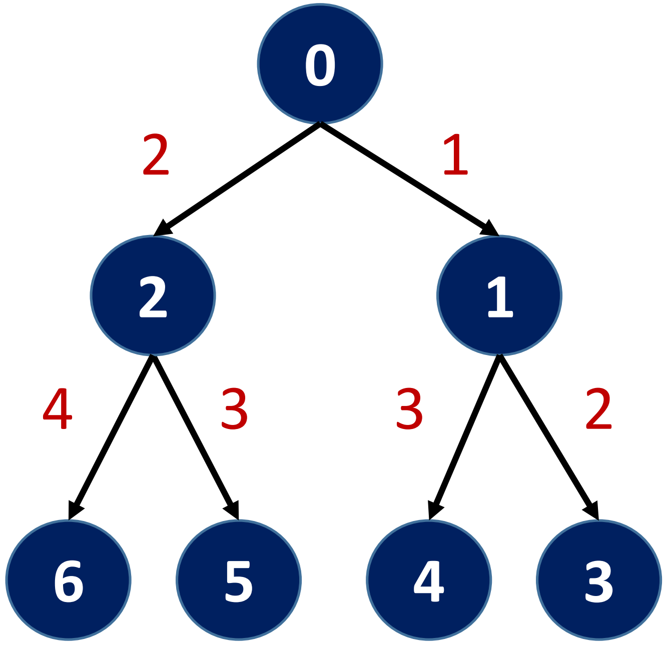
Broadcast – Binary Tree

- Root process sends to its children in 2 steps
- Every node sends to its children in 2 steps (left and right)

Cons

- Idling processes

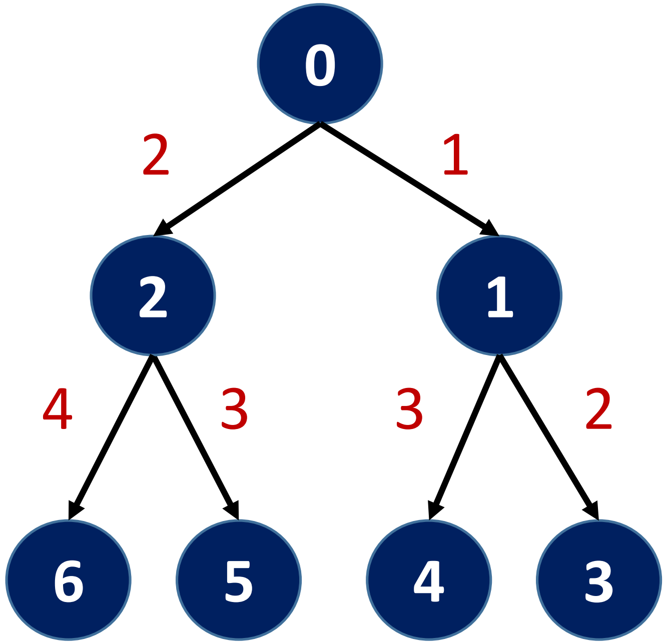
- #Steps for p processes?
  - 2 * log p
- Transfer time for n bytes
  - T(p) = 2 * log p * (L + n/B)



Broadcast – Pipelined Binary Tree

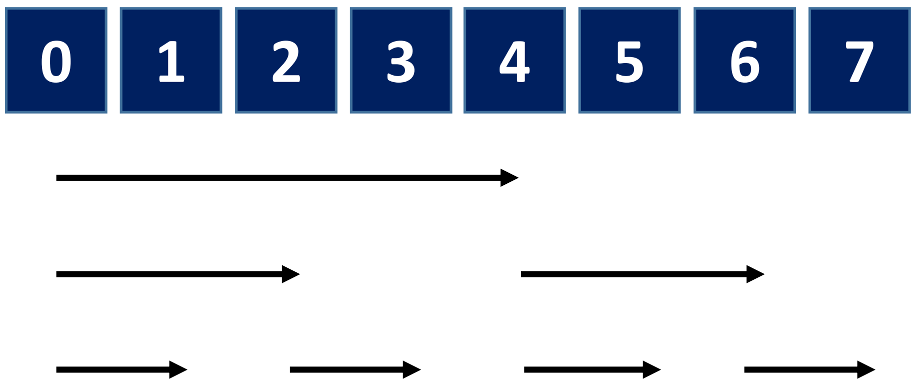
- Almost similar to binary
- Multiple rounds and smaller message size per round

- #Steps for p processes?
  - 2 * (log p + R – 1)
- Transfer time for n bytes
  - T(p) = 2 * (log p + R – 1)* (L + (n/R)*1/B)

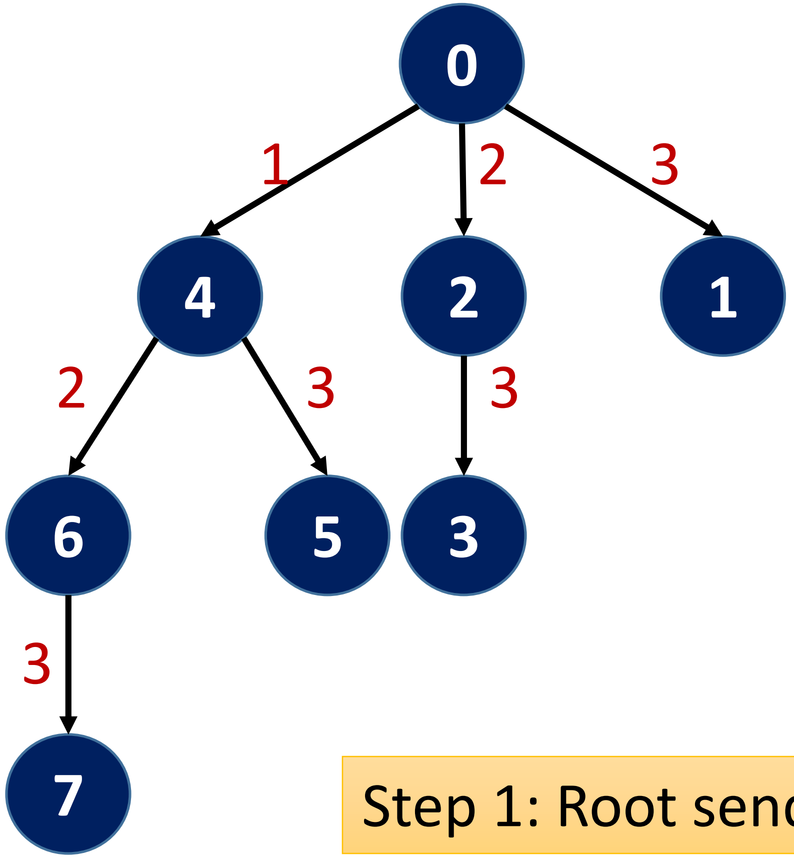


Sanders et al., “Full Bandwidth Broadcast, Reduction and Scan with only two Trees”

Broadcast – Binomial Tree



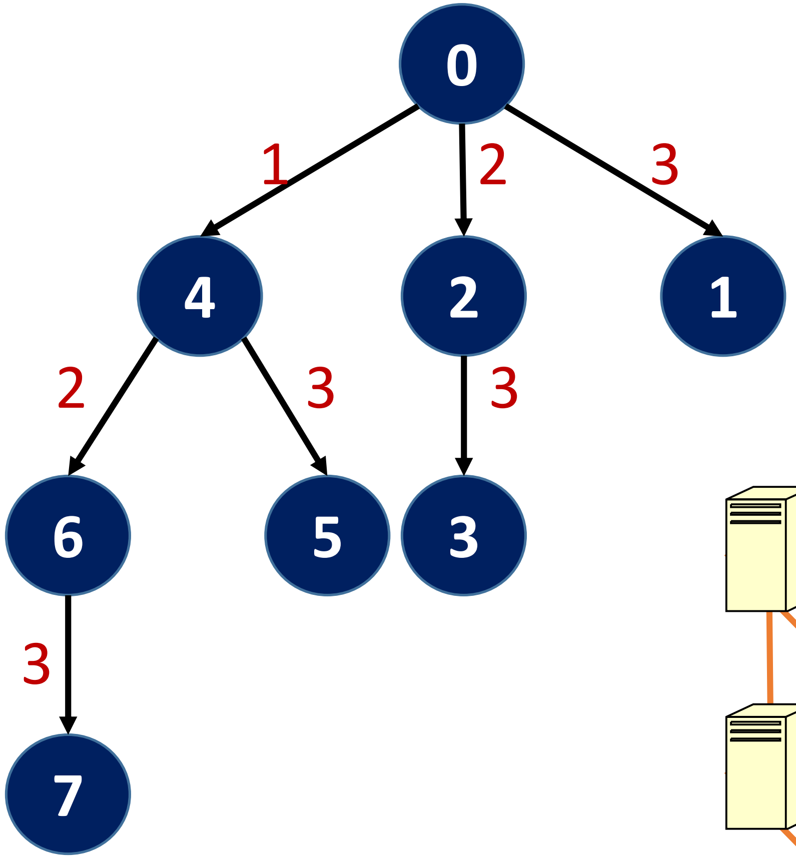
- #Steps for p (=2^d) processes?
  - log p
- Transfer time for n bytes
  - T(p) = log p * (L + n/B)
  - T(p^2) = 2 log p * (L + n/B)

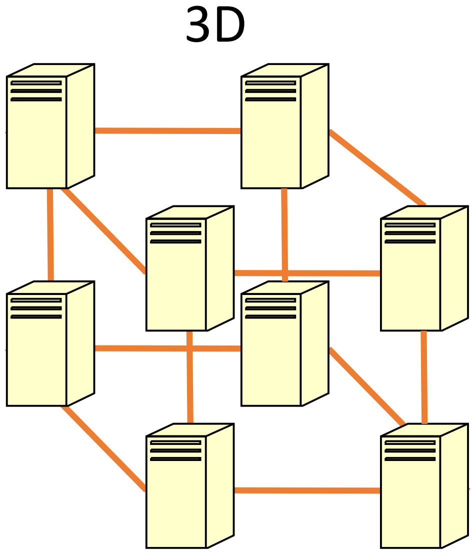


Step 1: Root sends to p/2

Broadcast

Q: Which interconnect would most likely exhibit minimum link contention for binomial tree broadcast algorithm?

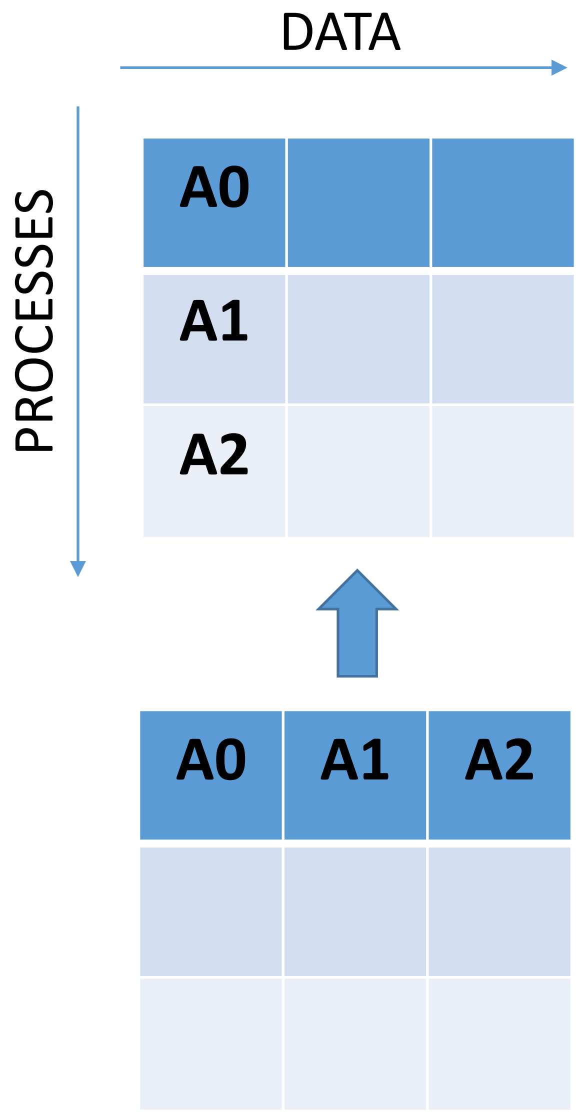




Q: Equivalent collective?

Scatter

- Scatters values to all processes from a root process
- int MPI_Scatter (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
- Arguments send* not relevant on non-root processes
- Output parameter – recvbuf



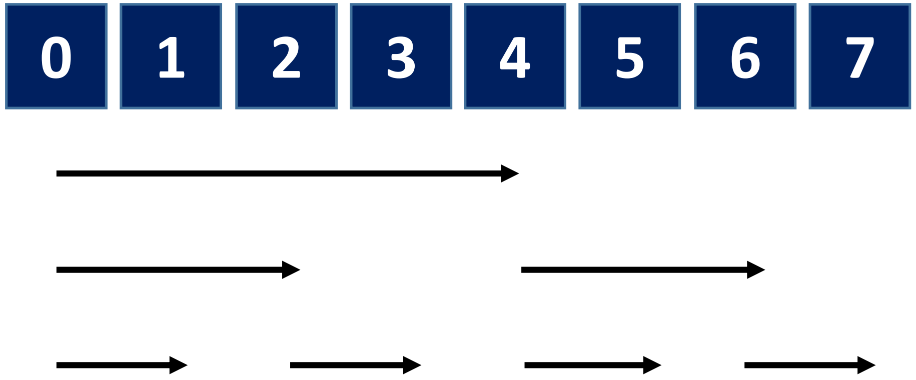
DATA

PROCESSES

A0		
A1		
A2		

A0	A1	A2

Scatter – Recursive Halving



Every step the message size halves: $n/2$, $n/2^2$, …, $n/2^{(\log p)}$

Time for scatter of n bytes from root

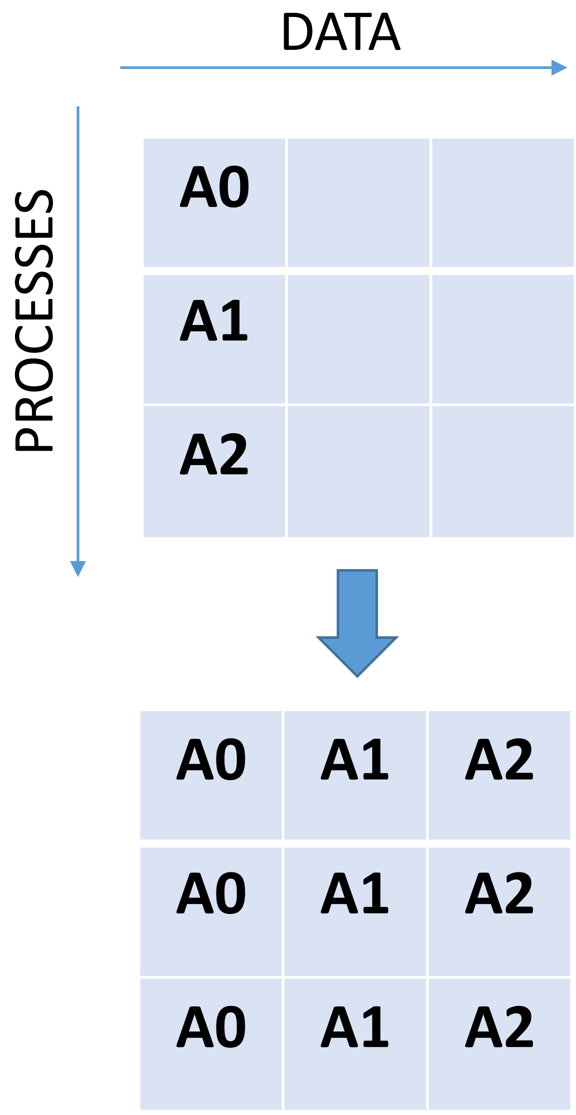
$\log p * L + (p-1)*(n/p)*(1/B)$

Allgather

- All processes gather values from all processes
- int MPI_Allgather (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
- No root process

```
recvbuf = (int *) malloc
(size*100*sizeof(int));

MPI_Allgather (sendbuf, 100, MPI_INT,
recvbuf, 100, MPI_INT, comm);
```

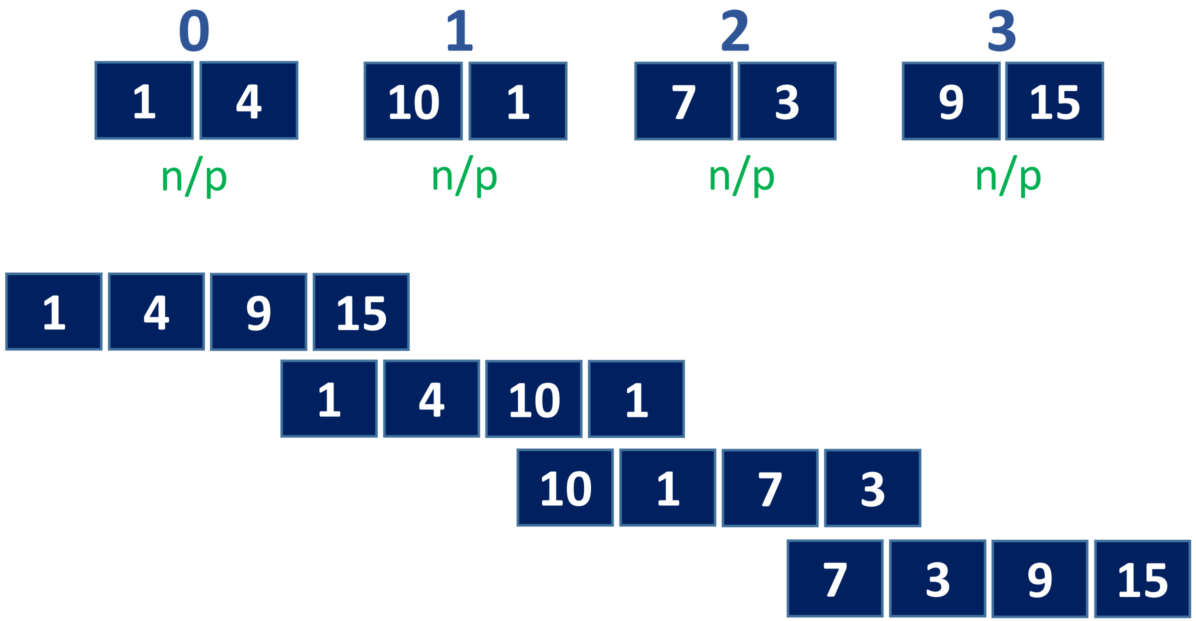


A0		
A1		
A2		

A0	A1	A2
A0	A1	A2
A0	A1	A2

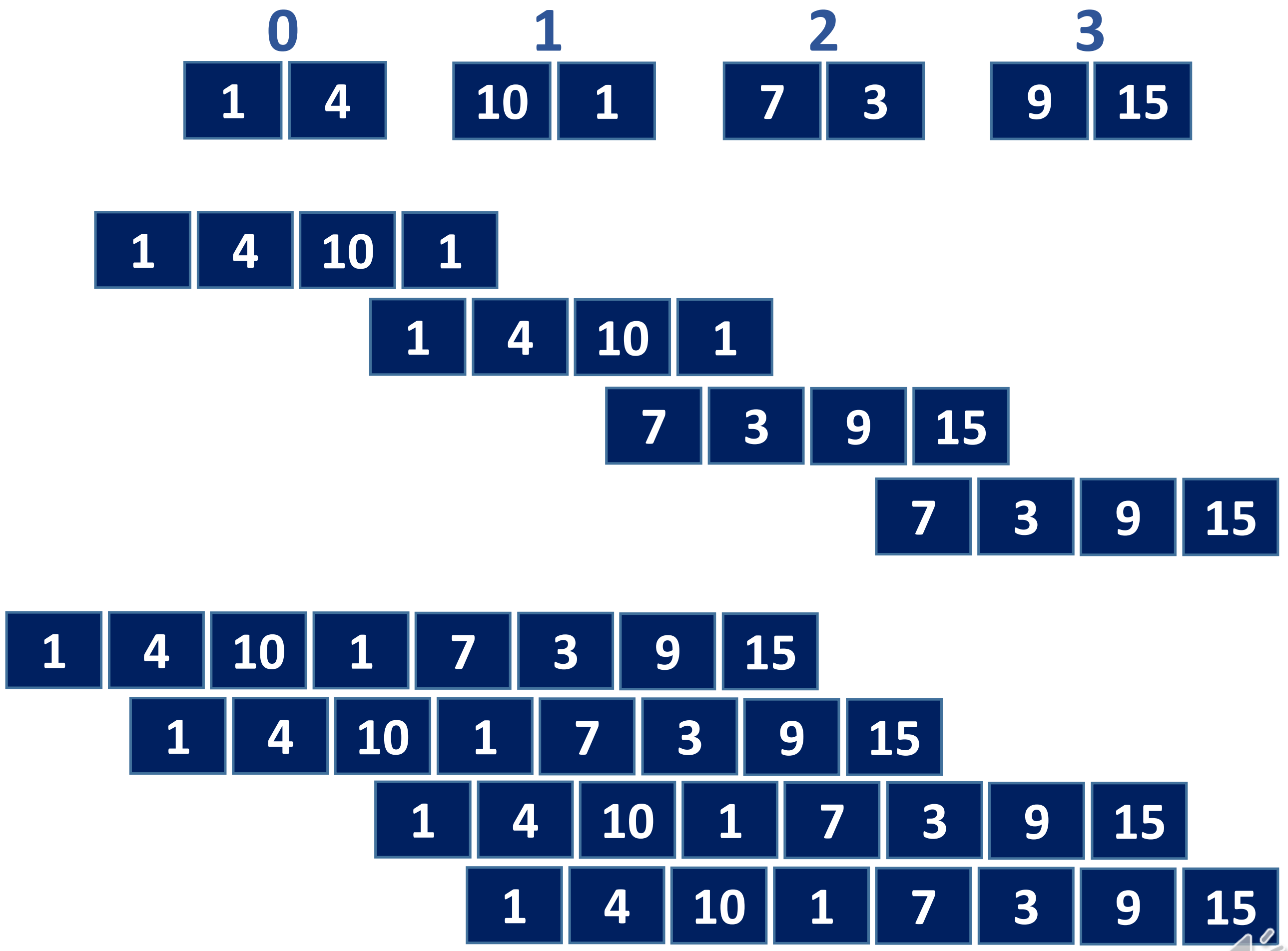
Allgather – Ring Algorithm

- Every process sends to and receives from everyone else
- Assume p processes and total n bytes
- Every process sends and receives n/p bytes
- Time
    - (p – 1) * (L + n/p*(1/B))
- How can we improve?



Allgather – Recursive Doubling

- Every process sends and receives $(2^{k-1})*$ n/p bytes at kth step
- (log p) * L + (p-1)*n/p*(1/B)



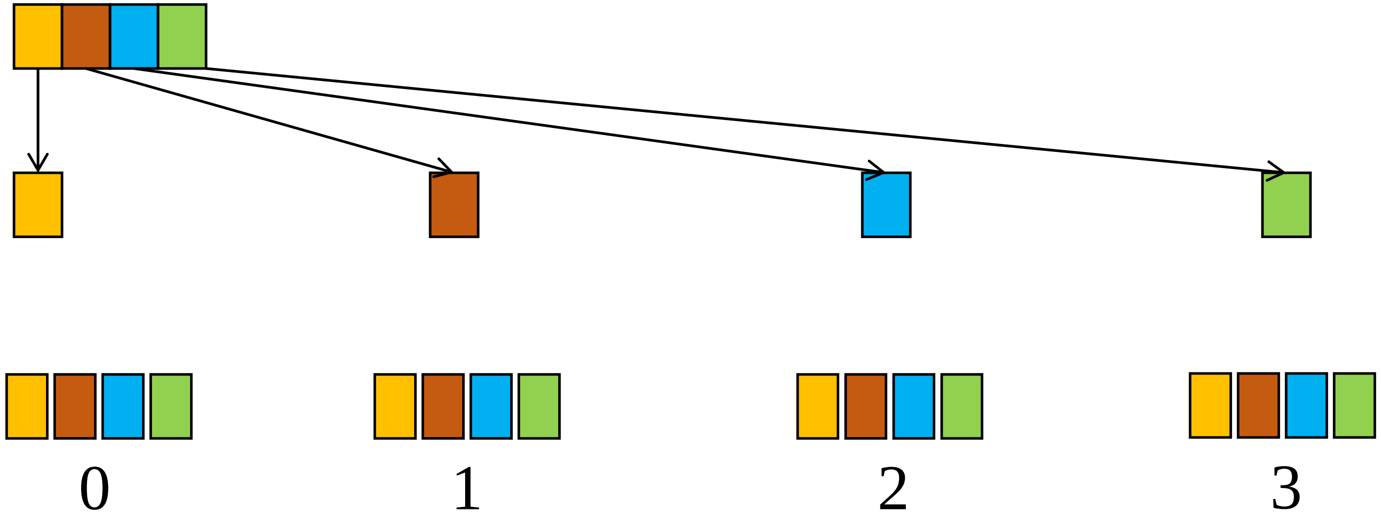
1 4 | 10 1 | 7 3 | 9 15

1 4 10 1
1 4 10 1
7 3 9 15
7 3 9 15

1 4 10 1 7 3 9 15
1 4 10 1 7 3 9 15
1 4 10 1 7 3 9 15
1 4 10 1 7 3 9 15

Bcast – Van de Geijn et al.

- Message is first scattered from the root
- Scattered data is collected at all processes

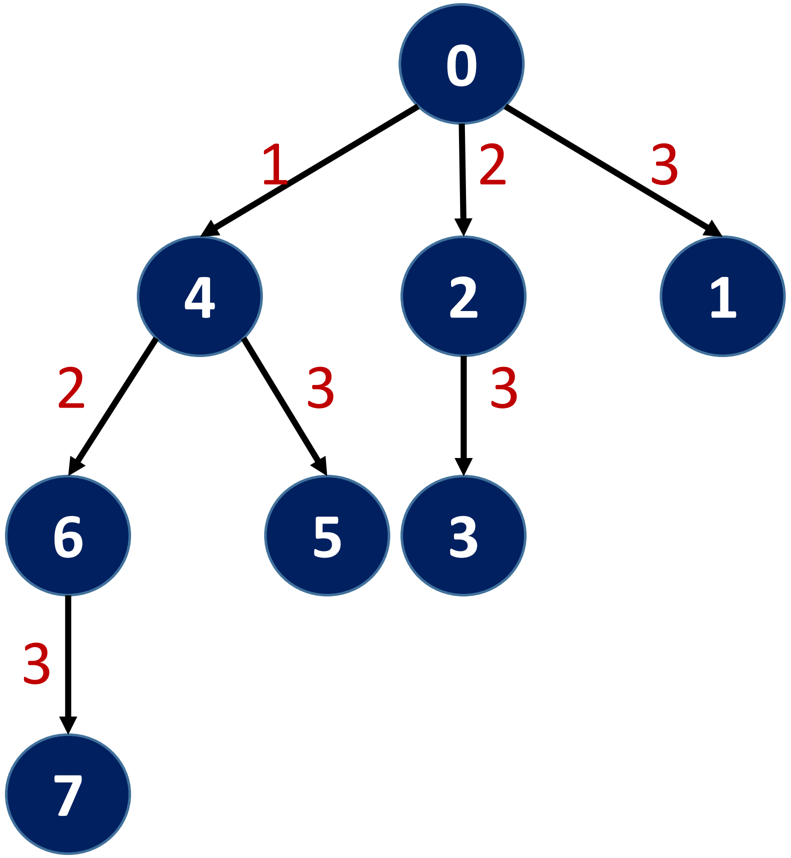


MPI_Scatter

MPI_Allgather

Bcast – Time Analysis

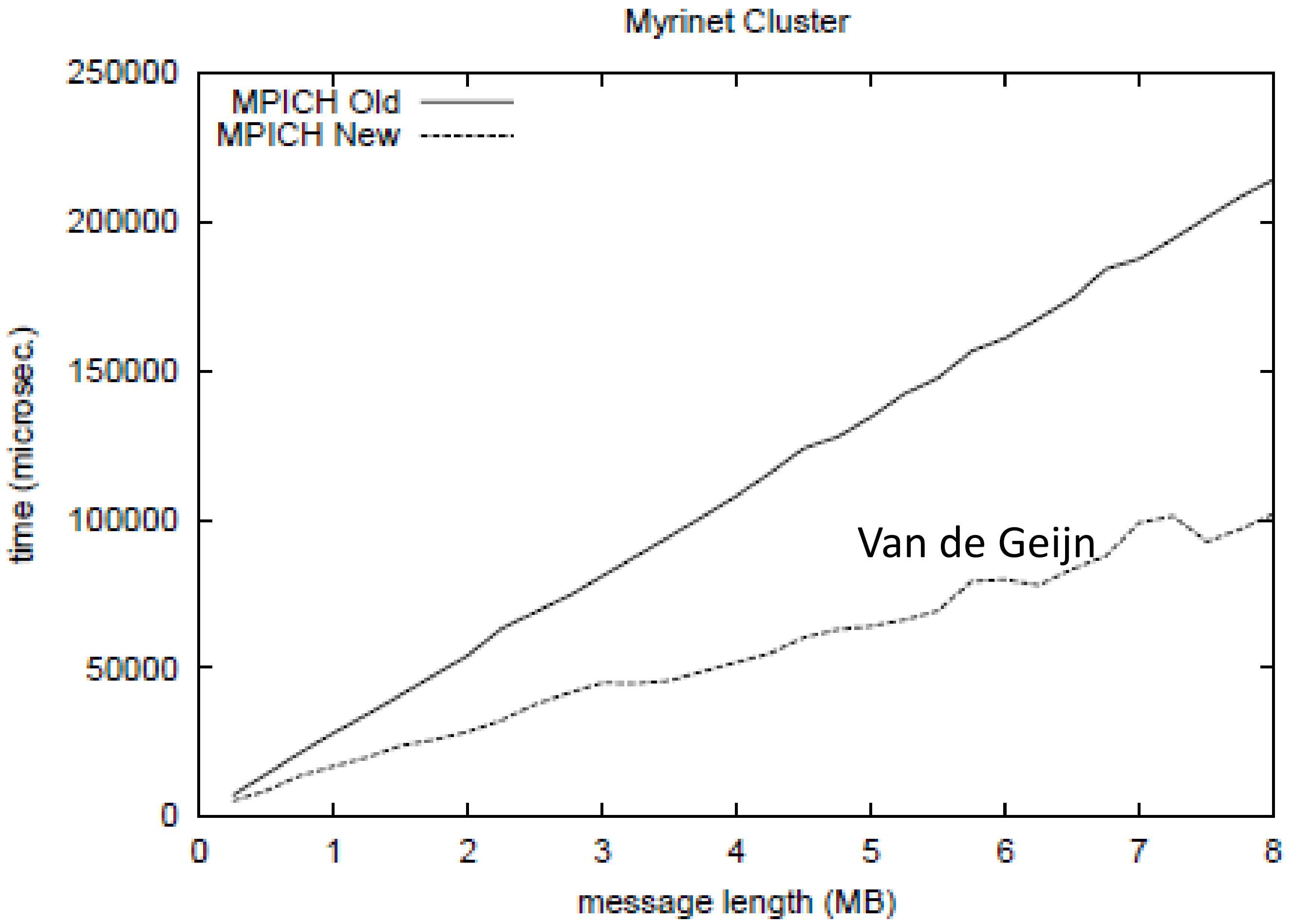
- Time for broadcasting n bytes from root (binomial tree)
  - log p * (L + n/B)
  - Latency term: log p
  - Bandwidth term: log p
- Time for scatter of n bytes from root
  - log p * L + (p-1)*(n/p)*(1/B)
- Time for allgather (ring) of n/p bytes
  - (p-1) * L + (p-1)*(n/p)*(1/B)
- Time for broadcast of n bytes using scatter and allgather
  - (log p + p-1) * L + 2((p-1)/p)*(n/B)



Broadcast Algorithms in MPICH

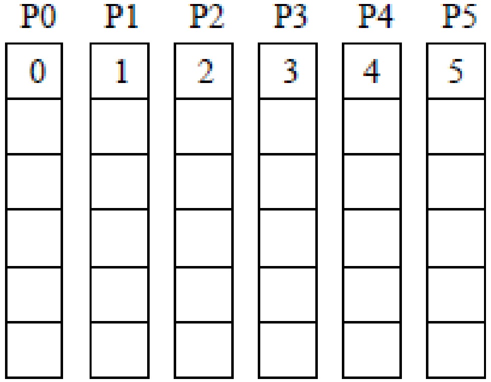
- Short messages
  - < MPIR_CVAR_BCAST_SHORT_MSG_SIZE
  - Binomial
- Medium messages
  - Scatter + Allgather (Recursive doubling)
- Large messages
  - > MPIR_CVAR_BCAST_LONG_MSG_SIZE
  - Scatter + Allgather (Ring)

Old vs. New MPI_Bcast



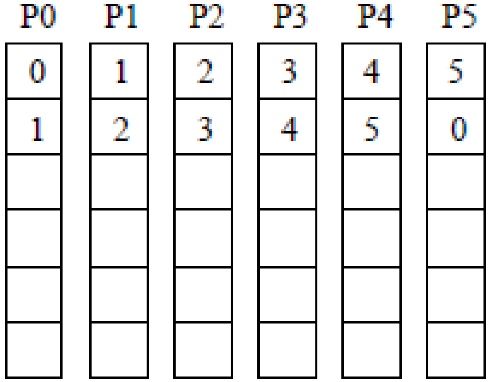
Van de Geijn

Allgather – Bruck Algorithm



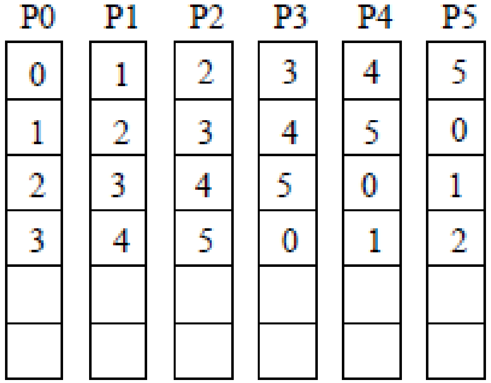
P0	P1	P2	P3	P4	P5
0	1	2	3	4	5

Initial data



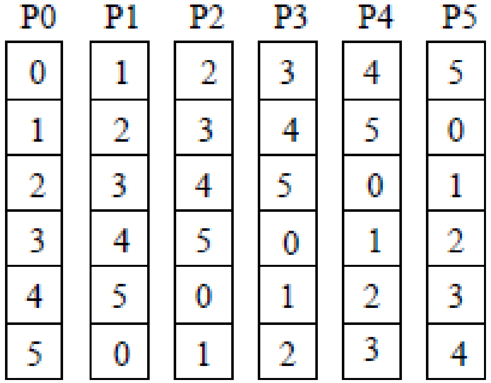
P0	P1	P2	P3	P4	P5
0	1	2	3	4	5
1	2	3	4	5	0

After step 0



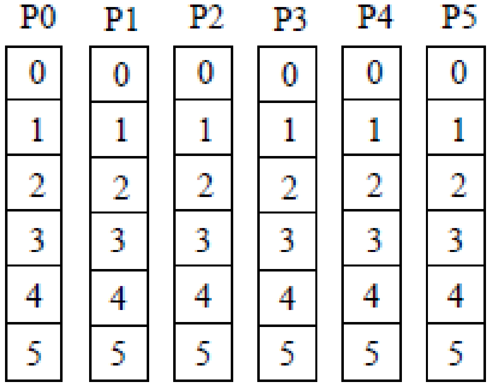
P0	P1	P2	P3	P4	P5
0	1	2	3	4	5
1	2	3	4	5	0
2	3	4	5	0	1
3	4	5	0	1	2

After step 1



P0	P1	P2	P3	P4	P5
0	1	2	3	4	5
1	2	3	4	5	0
2	3	4	5	0	1
3	4	5	0	1	2
4	5	0	1	2	3
5	0	1	2	3	4

After step 2



P0	P1	P2	P3	P4	P5
0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5

After local shift

Shift up by i blocks

Process i

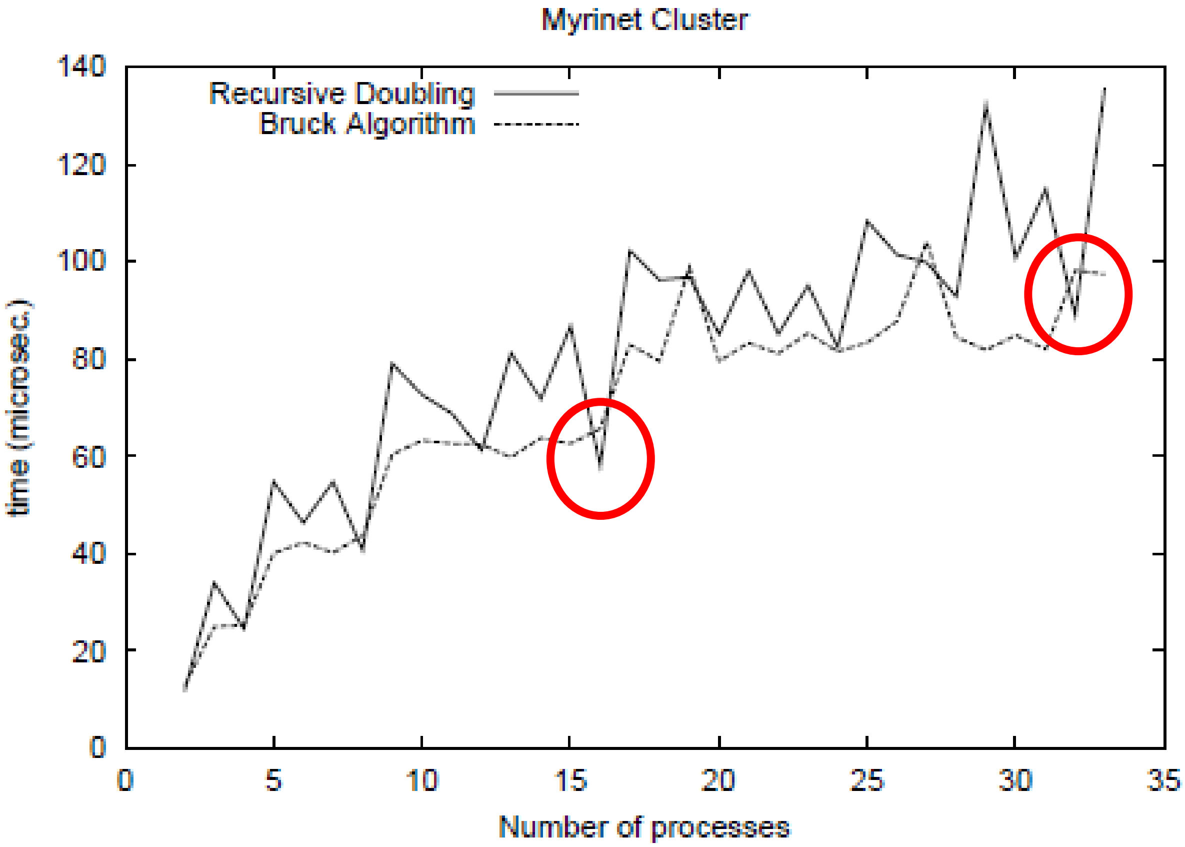
Send i – 2^k

Recv i + 2^k

Non-power-of-2

Send top p - $2^{\text{floor(lg p)}}$

Recursive Doubling vs. Bruck

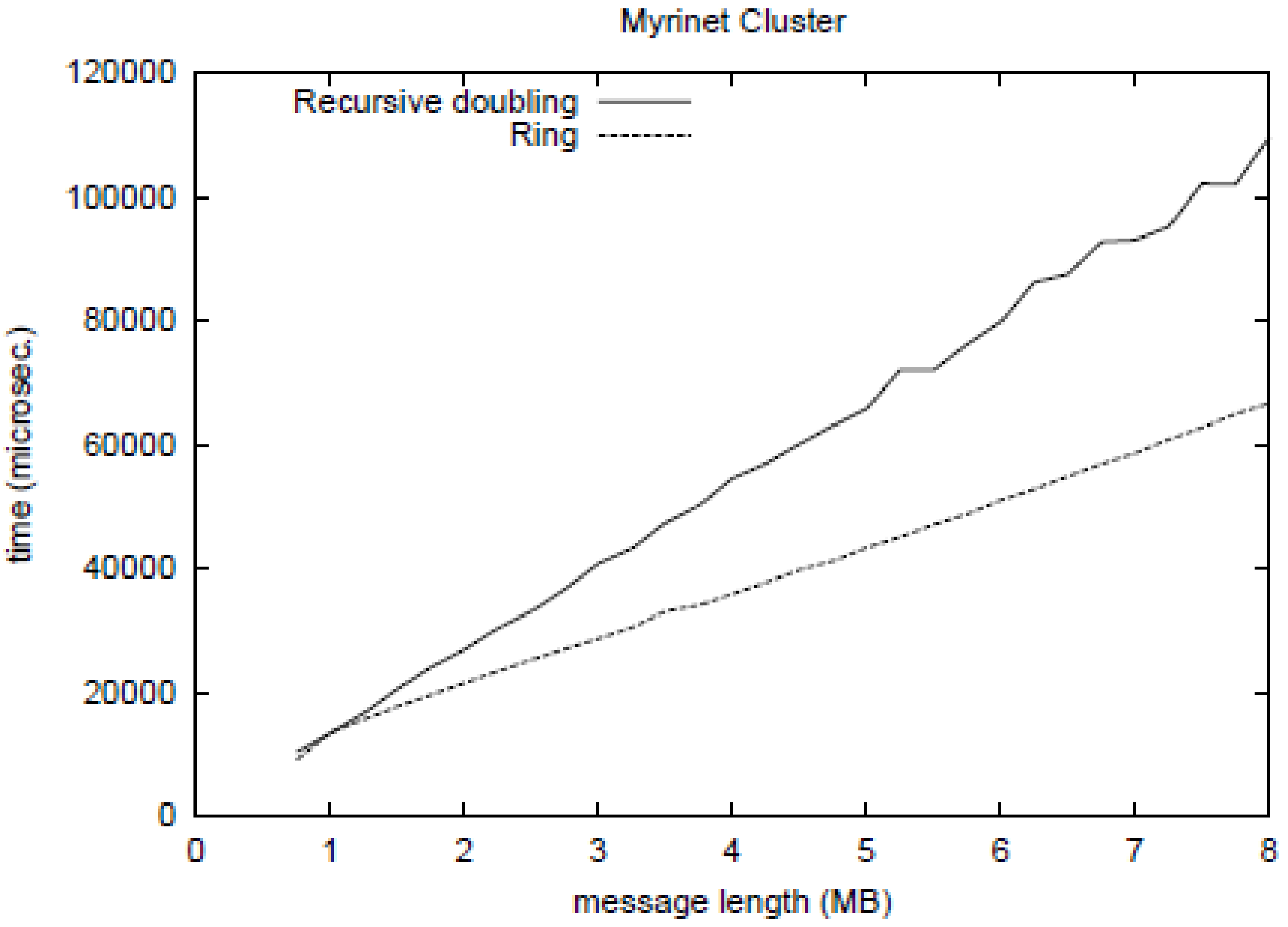


MPI_Allgather in MPICH

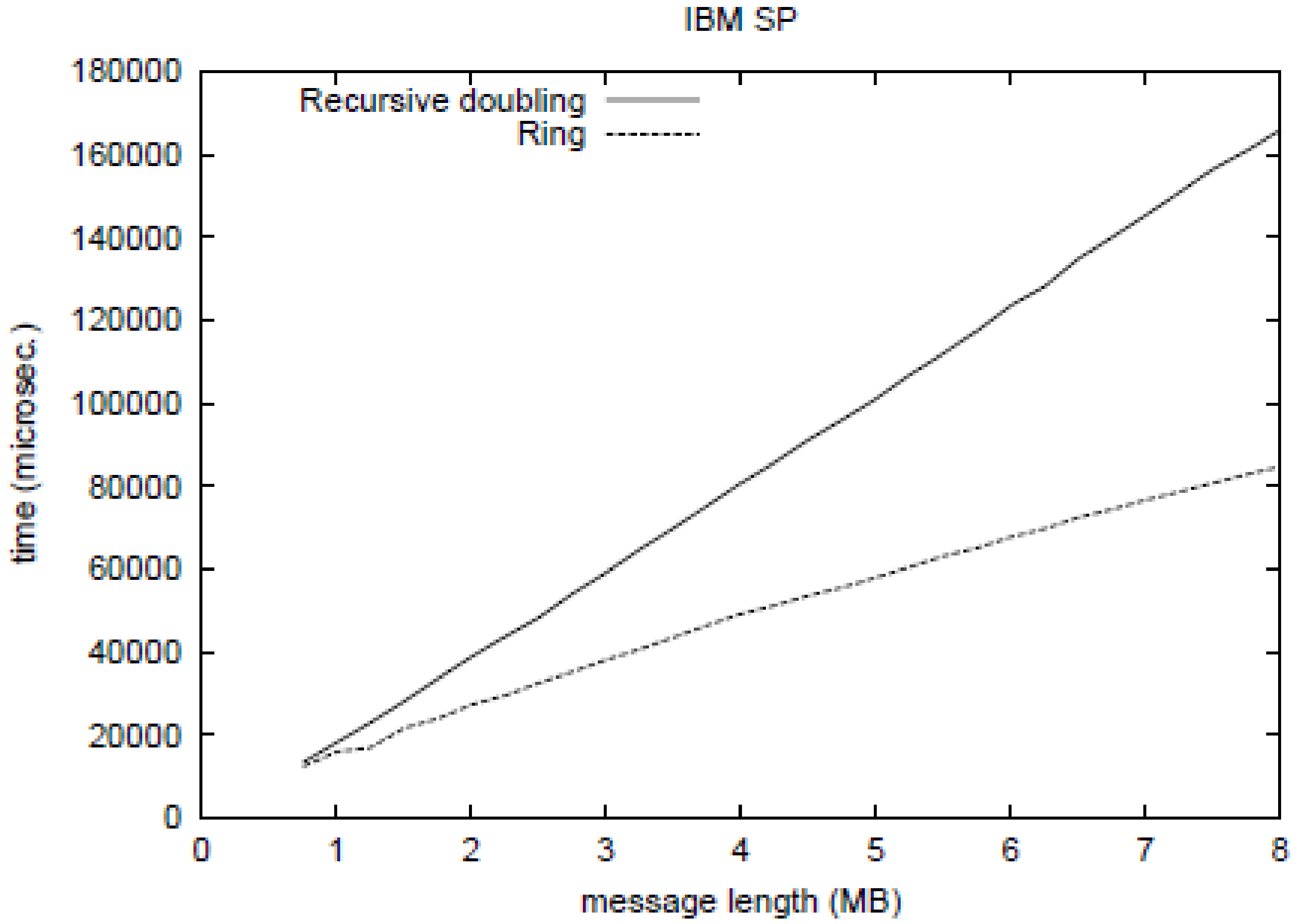
- Bruck algorithm
  - ceil(log p) * L + (p-1)*n/p*(1/B)
  - short messages (< 80 KB) and non-power-of-two numbers of processes
- Recursive doubling
  - (log p) * L + (p-1)*n/p*(1/B)
  - power-of-two numbers of processes and short or medium-sized messages (< 512 KB)
- Ring algorithm
  - (p-1) * L + (p-1)*n/p*(1/B)
  - long messages and any number of processes
  - medium-sized messages and non-power-of-two numbers of processes

Why does ring perform better?

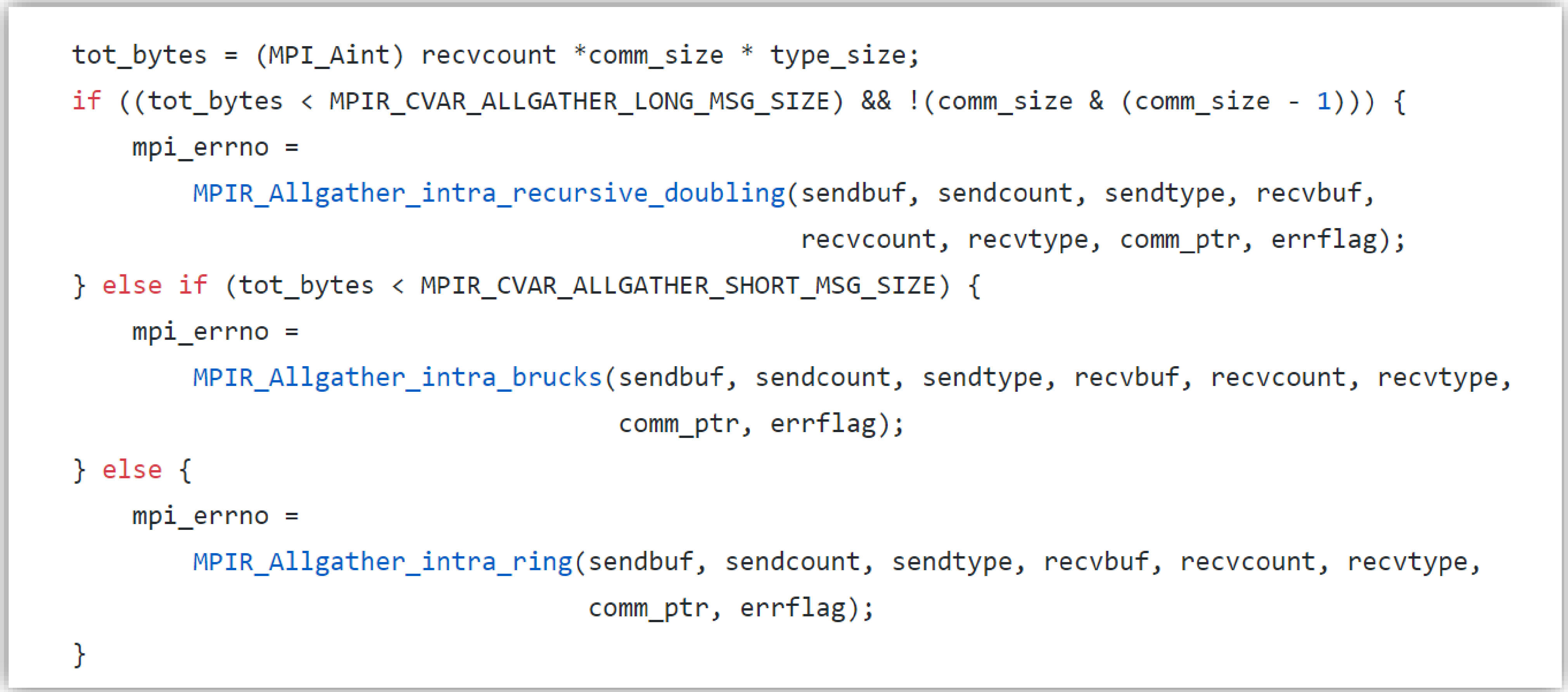
Performance Comparison



Performance Comparison

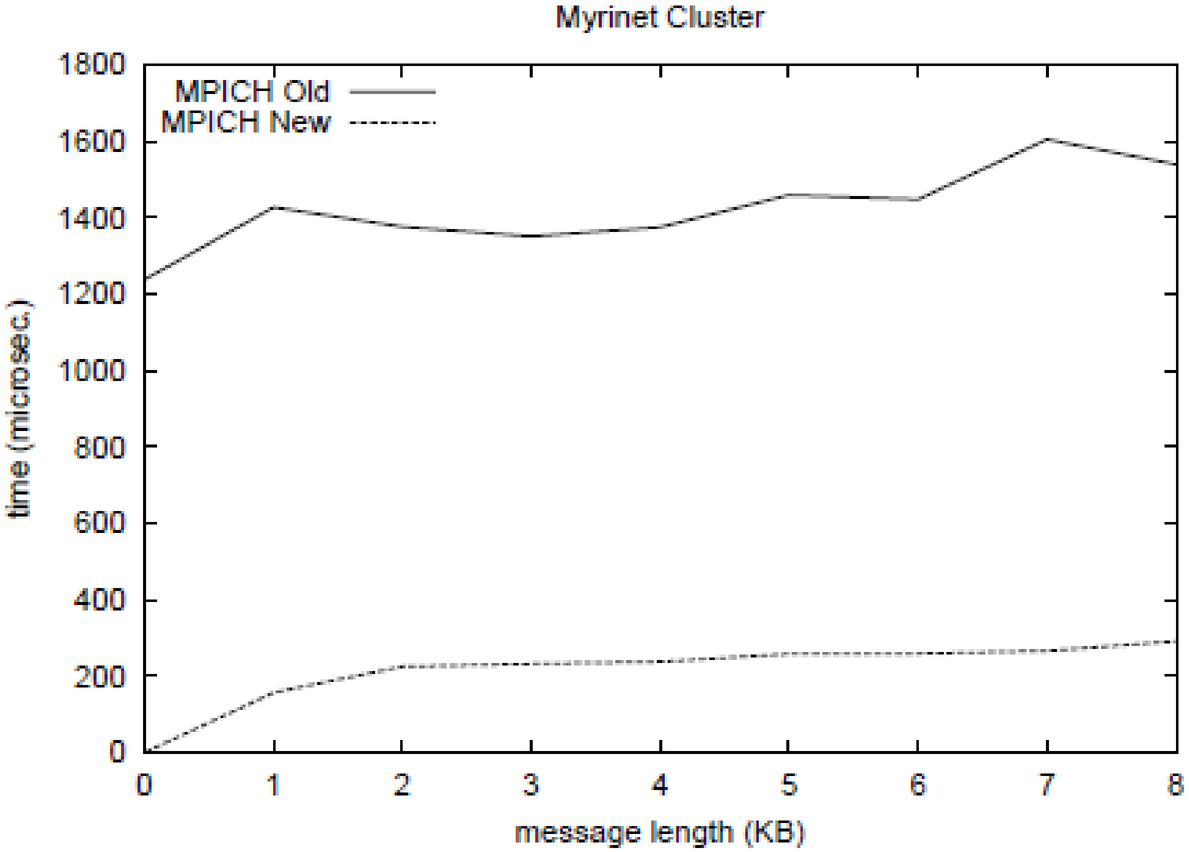


allgather.c



```
tot_bytes = (MPI_Aint) recvcount *comm_size * type_size;
if ((tot_bytes < MPIR_CVAR_ALLGATHER_LONG_MSG_SIZE) && !(comm_size & (comm_size - 1))) {
    mpi_errno =
        MPIR_Allgather_intra_recursive_doubling(sendbuf, sendcount, sendtype, recvbuf,
                                                recvcount, recvtype, comm_ptr, errflag);
} else if (tot_bytes < MPIR_CVAR_ALLGATHER_SHORT_MSG_SIZE) {
    mpi_errno =
        MPIR_Allgather_intra_brucks(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype,
                                    comm_ptr, errflag);
} else {
    mpi_errno =
        MPIR_Allgather_intra_ring(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype,
                                  comm_ptr, errflag);
}
```

Adaptive/Dynamic Algorithm Selection



Old vs. new MPI_Allgather times