Collectives Algorithms

Lecture 11

February 12, 2024

Blocking Collectives

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

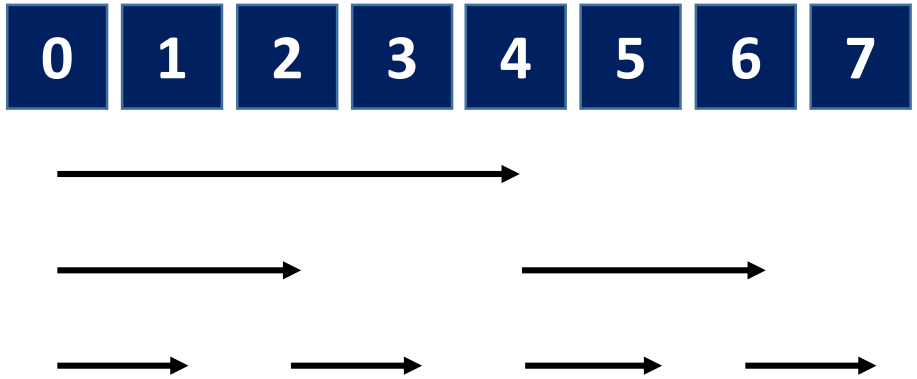
Communication Cost Model

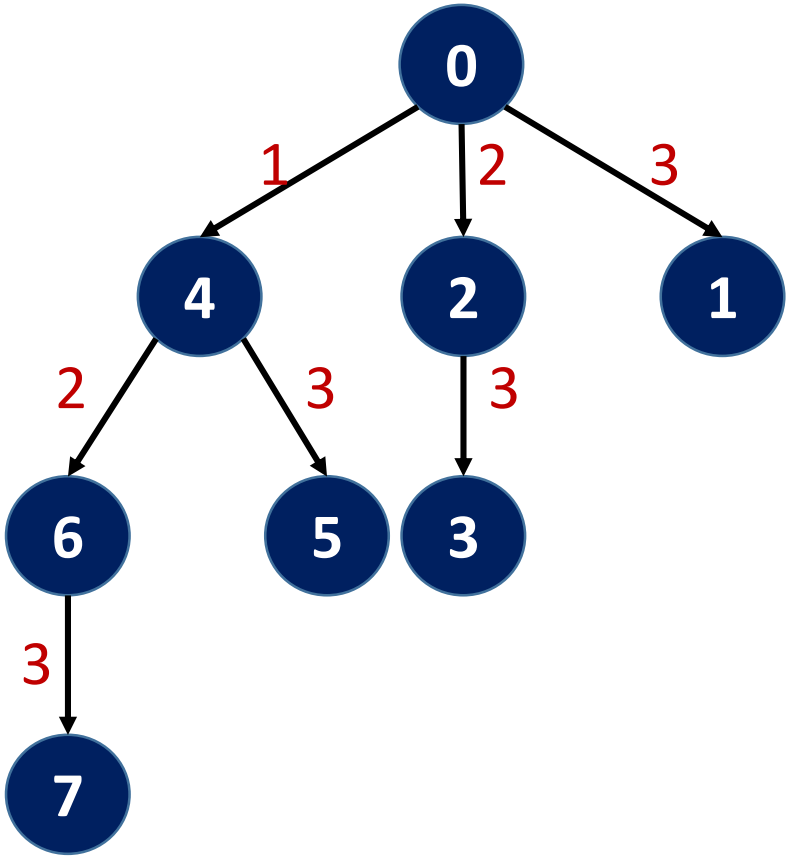
- Message transfer time is modeled as L+n/B, where L is the latency (or startup time) per message, and 1/B is the transfer time per byte, and n the message size in bytes

Optimization of Collective Communication Operations in MPICH

Rajeev Thakur, Rolf Rabenseifner, William Gropp
IJHPCA 2005

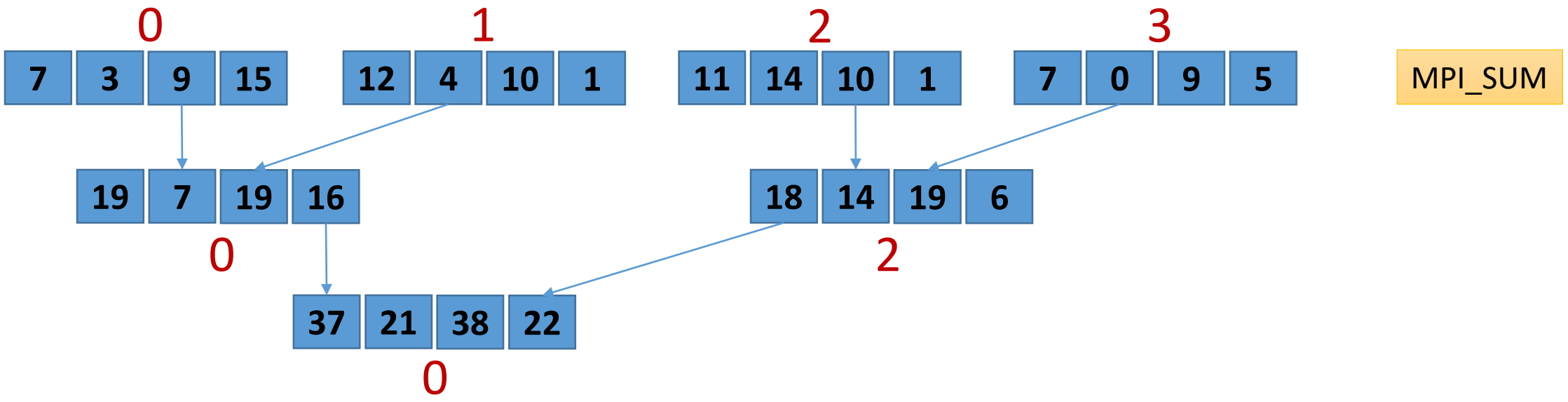
Broadcast – Binomial Tree





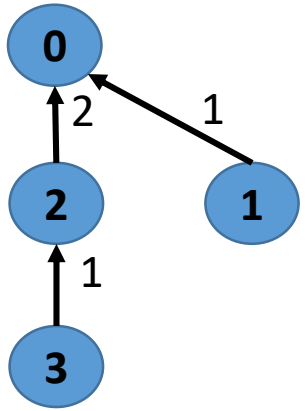
- #Steps for p (=2^d) processes?
  - log p
- Transfer time for n bytes
  - T(p) = log p * (L + n/B)

Reduce Algorithm – Recursive doubling

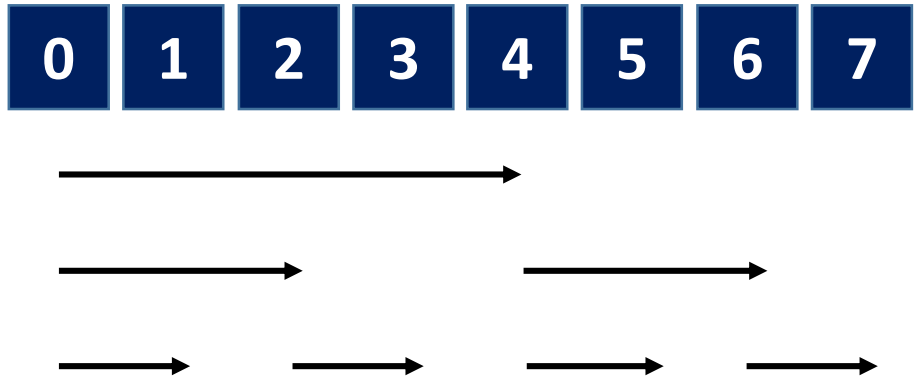


Time: log p (L + n*(1/B) + n*c)
L = latency, B = bandwidth
c = compute cost per byte

Used for short messages



Scatter



Vector halving
Distance halving

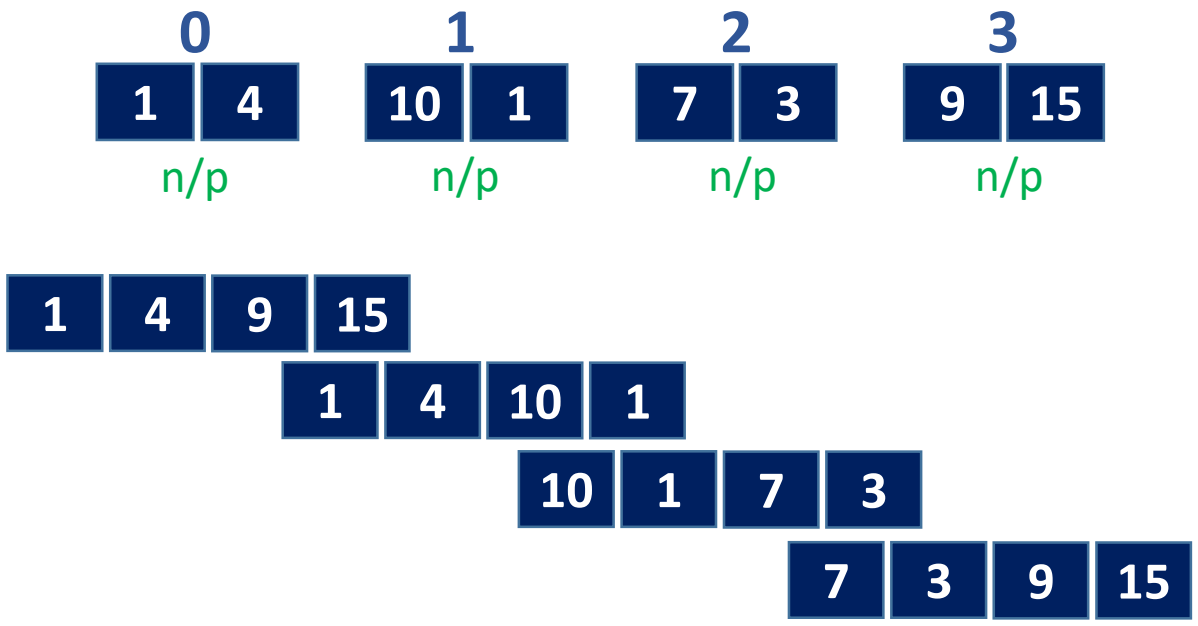
Every step the message size halves: $n/2, n/2^2, \ldots, n/2^{(\log p)}$

Time for scatter of n bytes from root

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

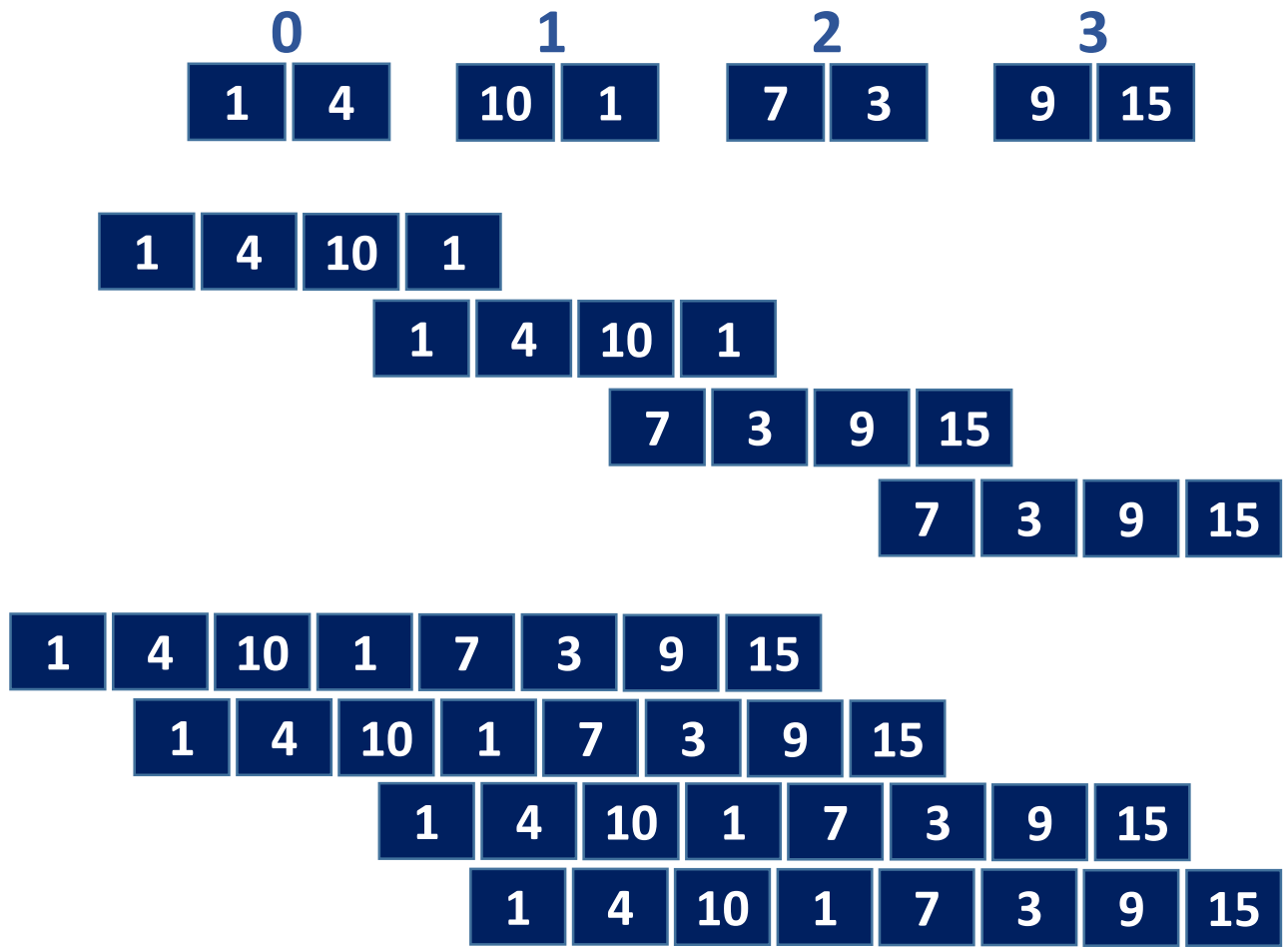
Allgather – Ring Algorithm

- 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

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



MPI_Allgather in MPICH

- MPIR_CVAR_ALLGATHER_SHORT_MSG_SIZE=81920
- MPIR_CVAR_ALLGATHER_LONG_MSG_SIZE=524288

- Bruck algorithm
  - 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

allgather.c

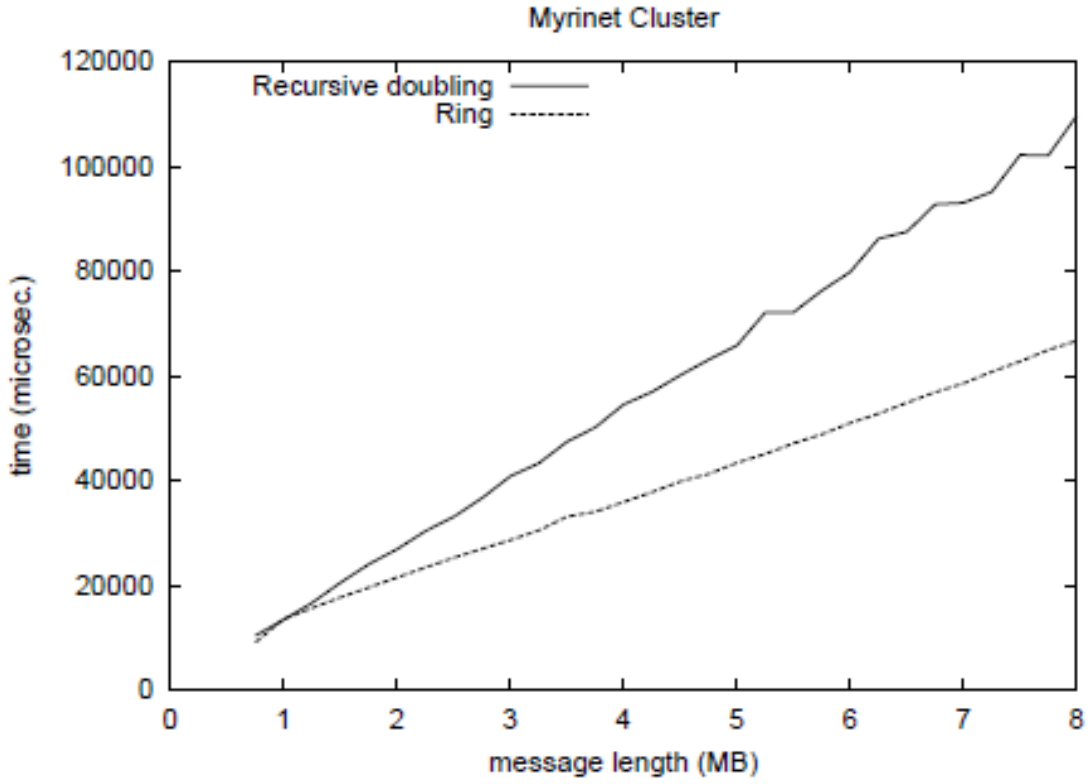
```
tot_bytes = (MPI_Aint) recvcount *comm_size * type_size;
if ((tot_bytes < MPIR_CVAR_ALLGATHER_LONG_MSG_SIZE) && !(comm_size & (comm_size - 1))) {
    mpi_errno =
        MPIR_Allgather_intra_recursive_doubling(sendbuf, sendcount, sendtype, recvbuf,
                                                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);
}
```

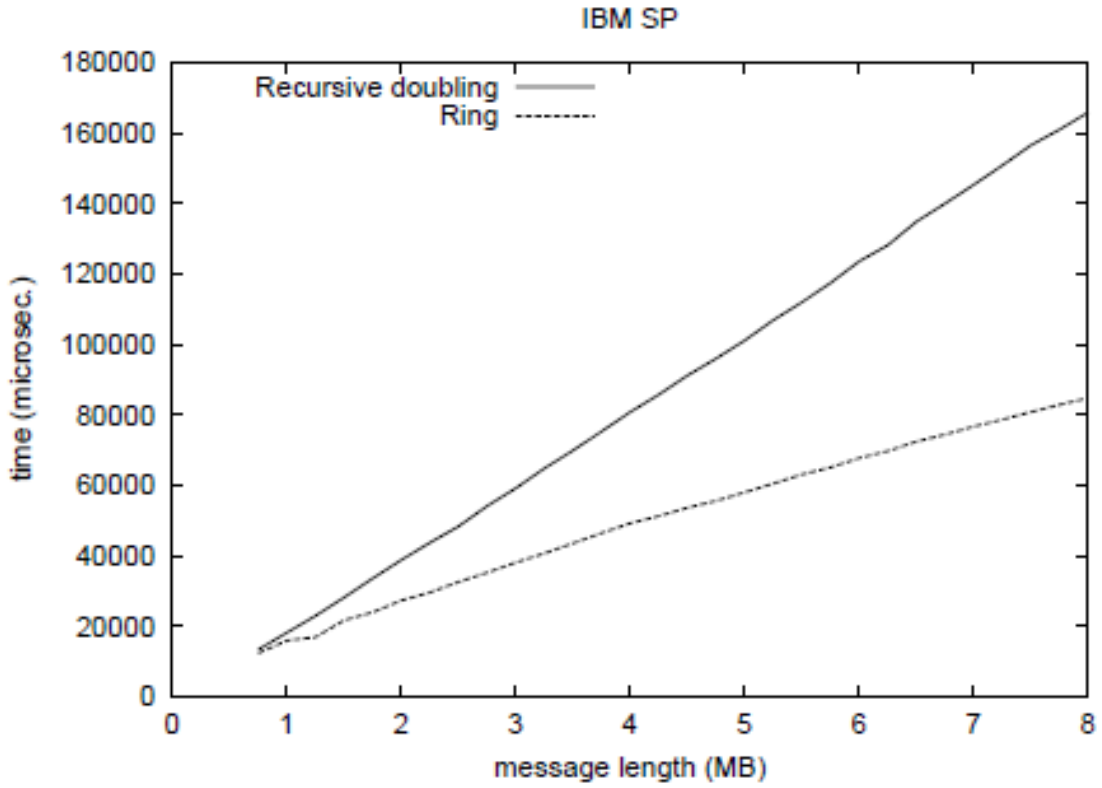
Performance Comparison on 64 nodes

Why does ring perform better?

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

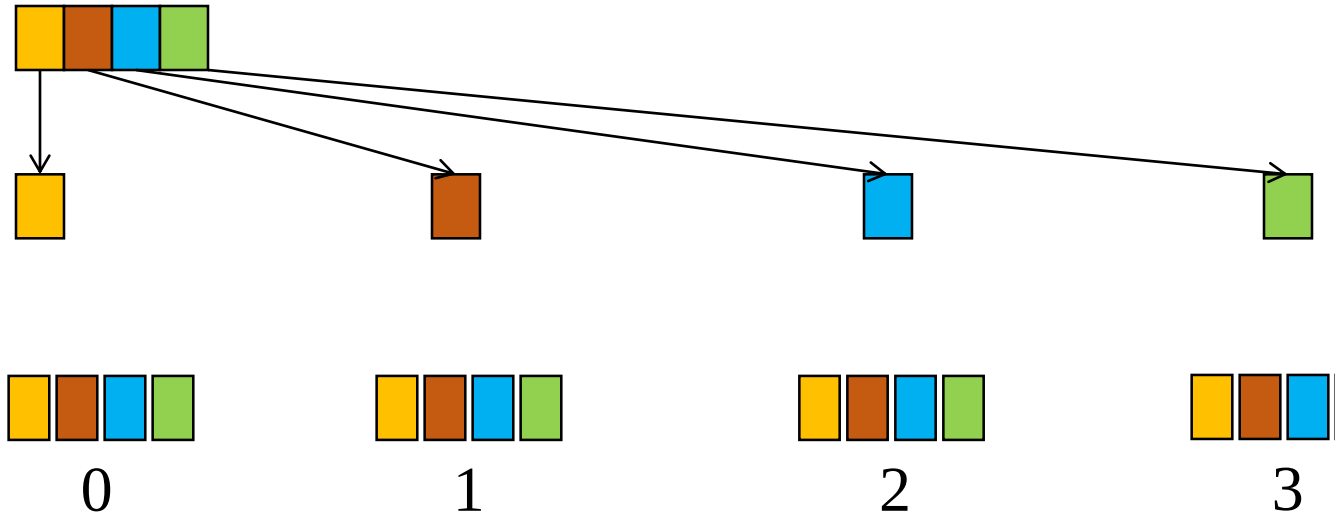
(p-1) * L + (p-1)*n/p*(1/B)





Bcast

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

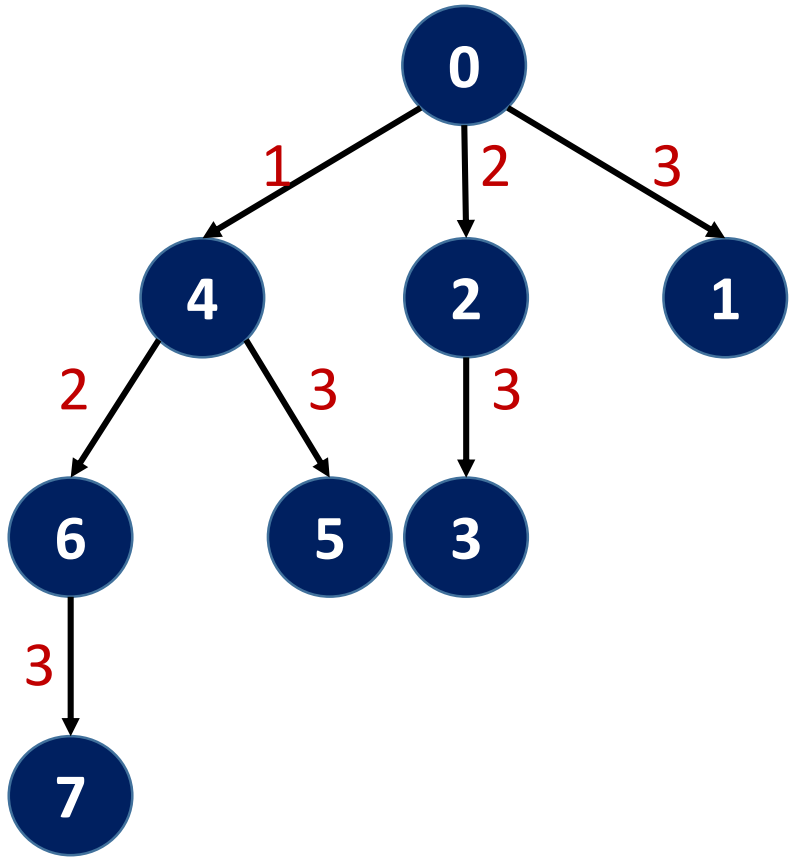


MPI_Scatter

MPI_Allgather

Bcast – Time Analysis

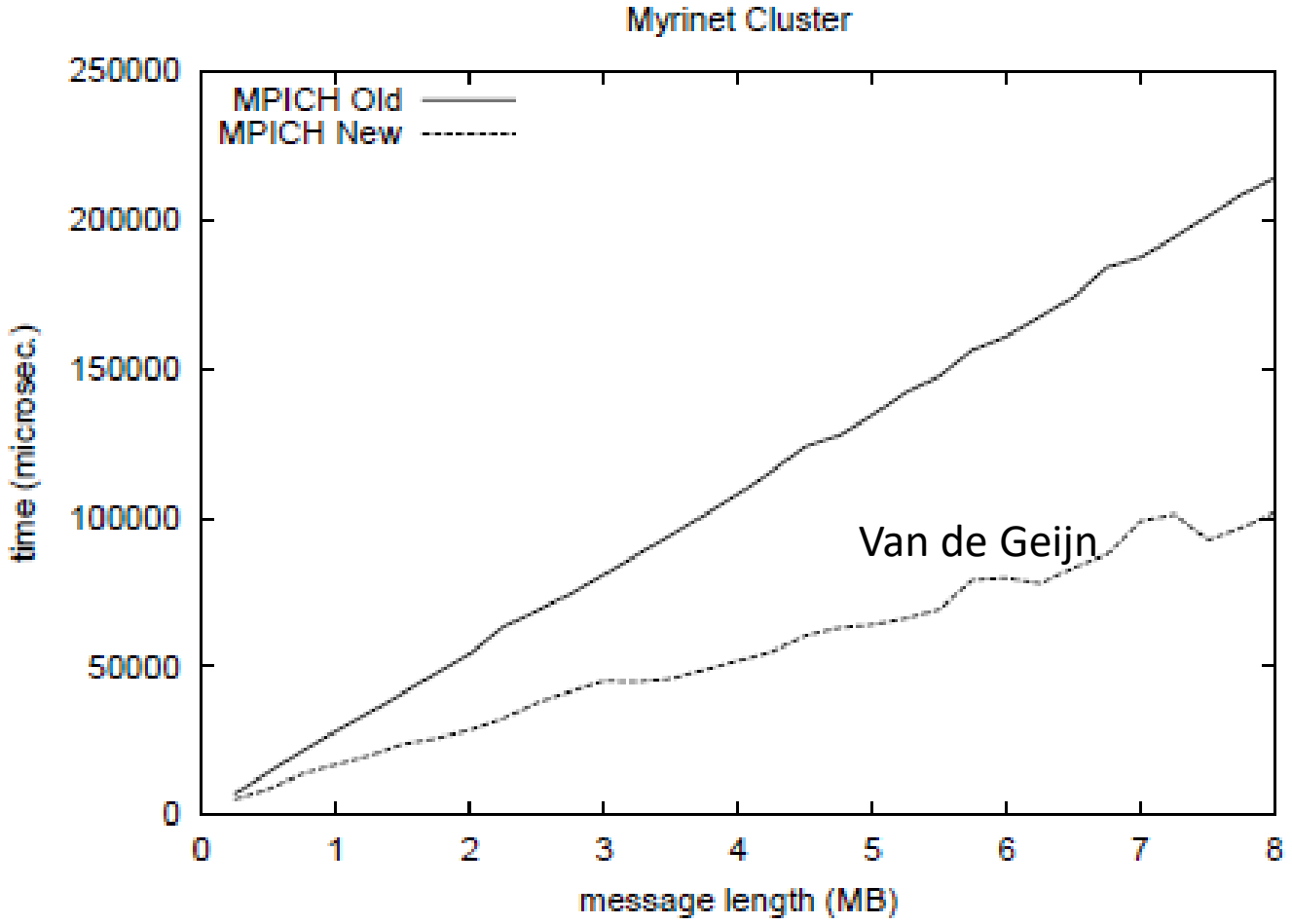
- Time for broadcasting n bytes from root (binomial tree)
  - log p * (L + n/B)
  - Latency term: log p
  - Bandwidth term: log p
- Time for scatter of n bytes from root
  - log p * L + (p-1)*(n/p)*(1/B)
- Time for allgather (ring) of n/p bytes
  - (p-1) * L + (p-1)*(n/p)*(1/B)
- Time for broadcast of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



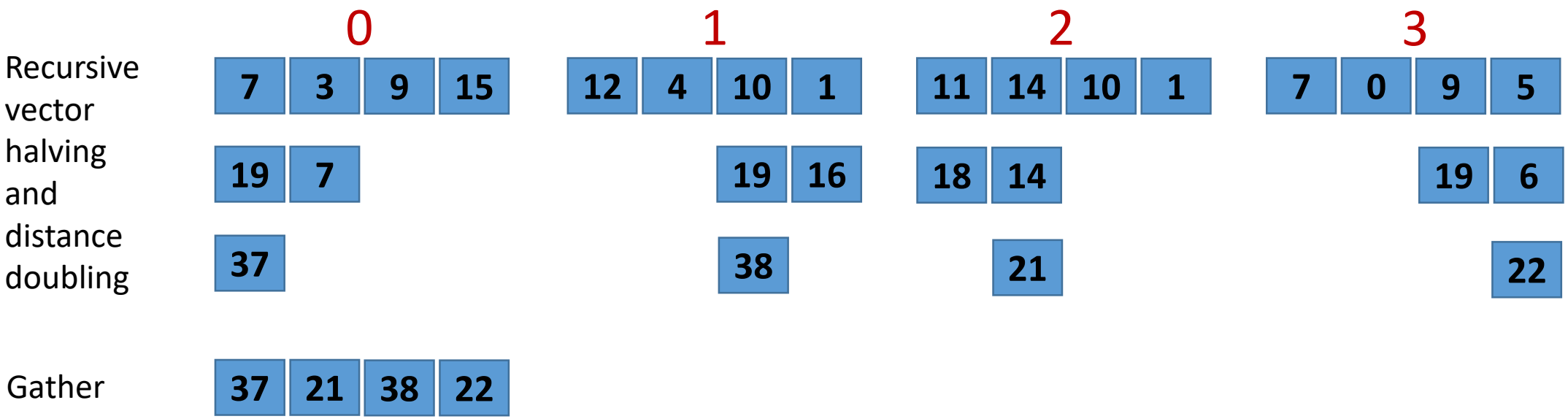
Broadcast Algorithms in MPICH

- 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 (64 nodes)



Reduce



Time:

log p * L + (p-1)/p*(n/B) + (p-1)/p*n*c (reduce-scatter) +

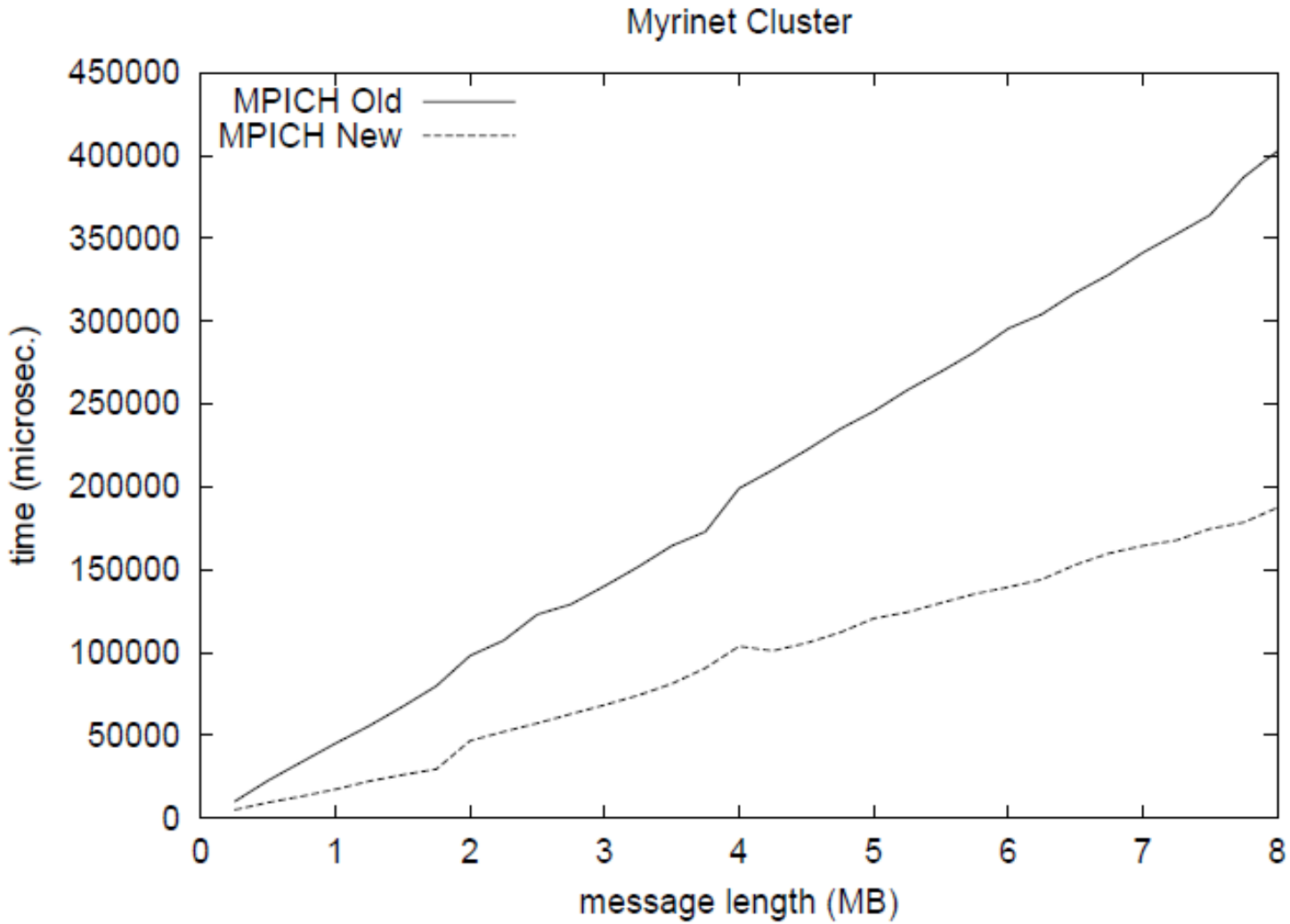
log p * L + (p-1)/p*(n/B) (gather using binomial)

n = data size L = latency

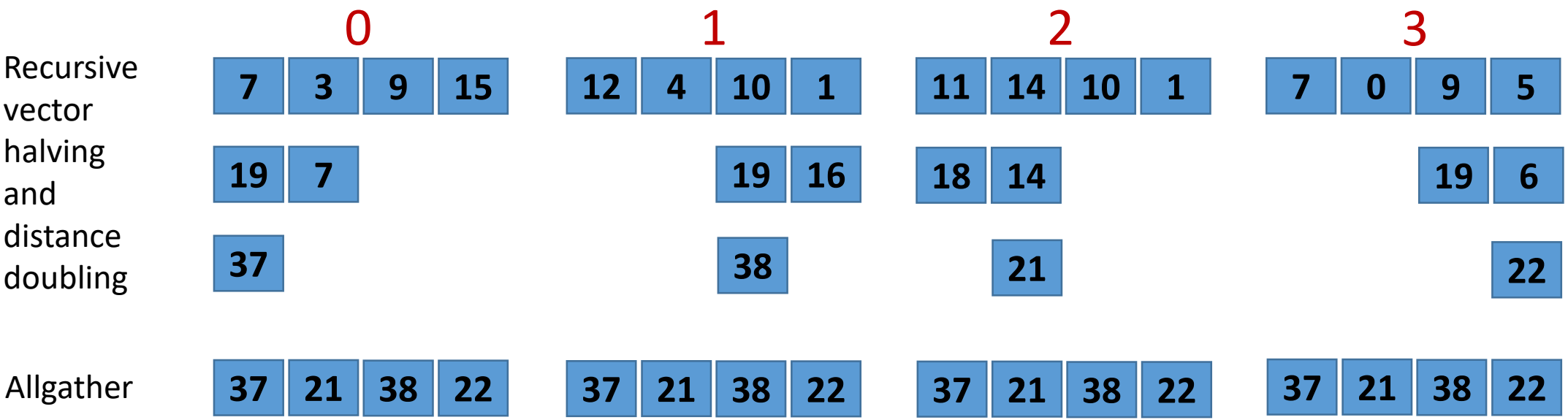
p = #processes B = bandwidth

c = compute cost per byte

Reduce on 64 nodes



Allreduce – Rabenseifner's Algorithm



Time:

log p * L + (p-1)/p*(n/B) + (p-1)/p*n*c (reduce-scatter) +

log p * L + (p-1)/p*(n/B) (allgather using recursive vector doubling and distance halving)

n = data size L = latency

p = #processes B = bandwidth

c = compute cost per byte

Allreduce Algorithms – Summary

Short - Medium messages

- Reduce (recursive doubling) followed by broadcast (binomial)
  - Time: [log p (L + n*(1/B) + n*c)] + [log p (L + n*(1/B)]

Long messages

- Reduce-scatter followed by allgather (recursive doubling)
  - Time: 2 * log p * L + 2(p-1)/p * (n/B) + (p-1)/p * n * c

Effect of Process Placement for Bcast