Collective Communications inside MPI

Jerome Vienne

Outline

- What is a Collective ?
- Behind Gather
- Overview of algorithms
  - Binomial Tree
  - Barrier Algorithms
  - Allgather
  - …
- Current state of research

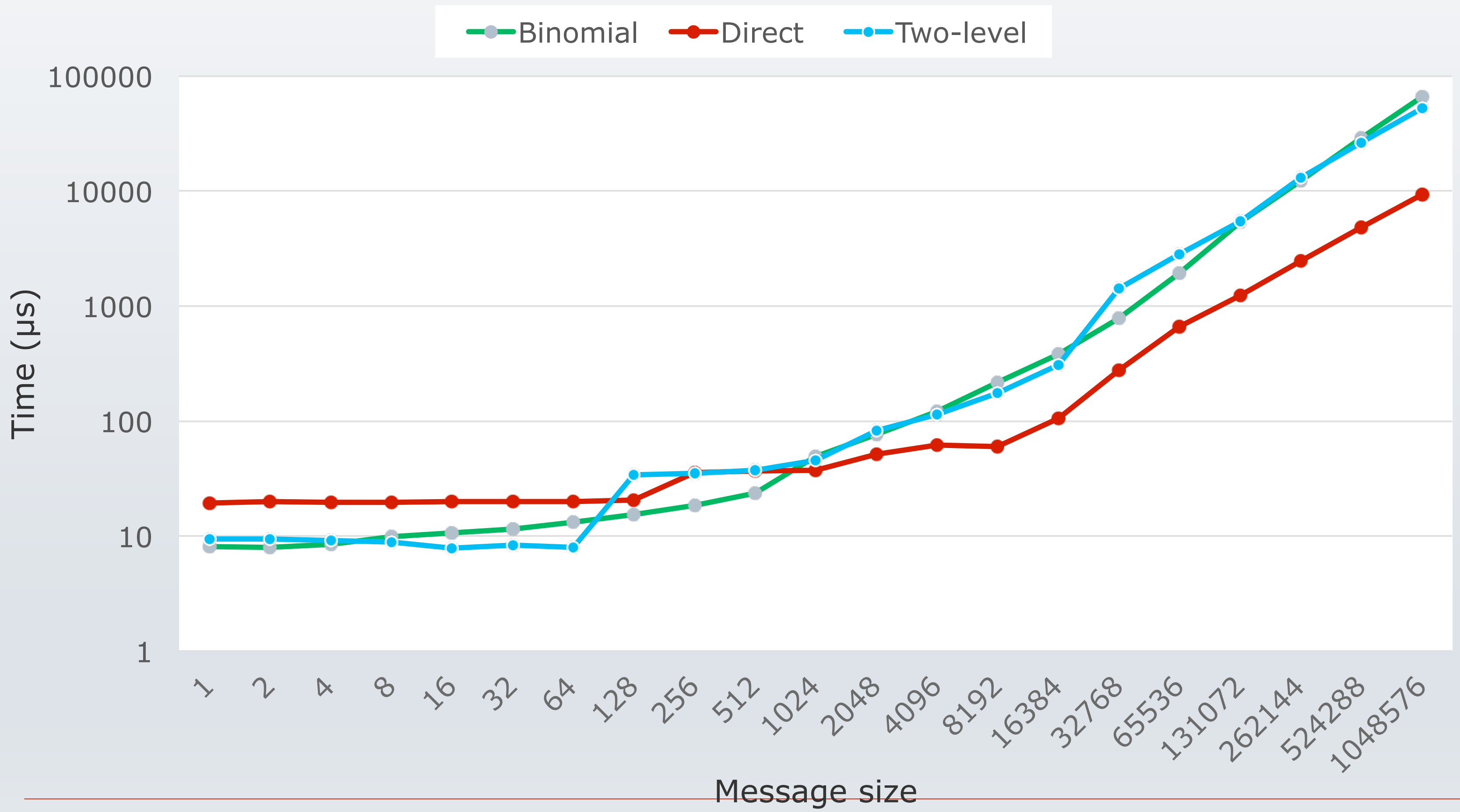
What is a Collective ?

- Collective communication is a method of communication which involves participation of **all** processes in a communicator.
- MPI-1: Blocking Collectives
- MPI-3: Non-blocking Collectives

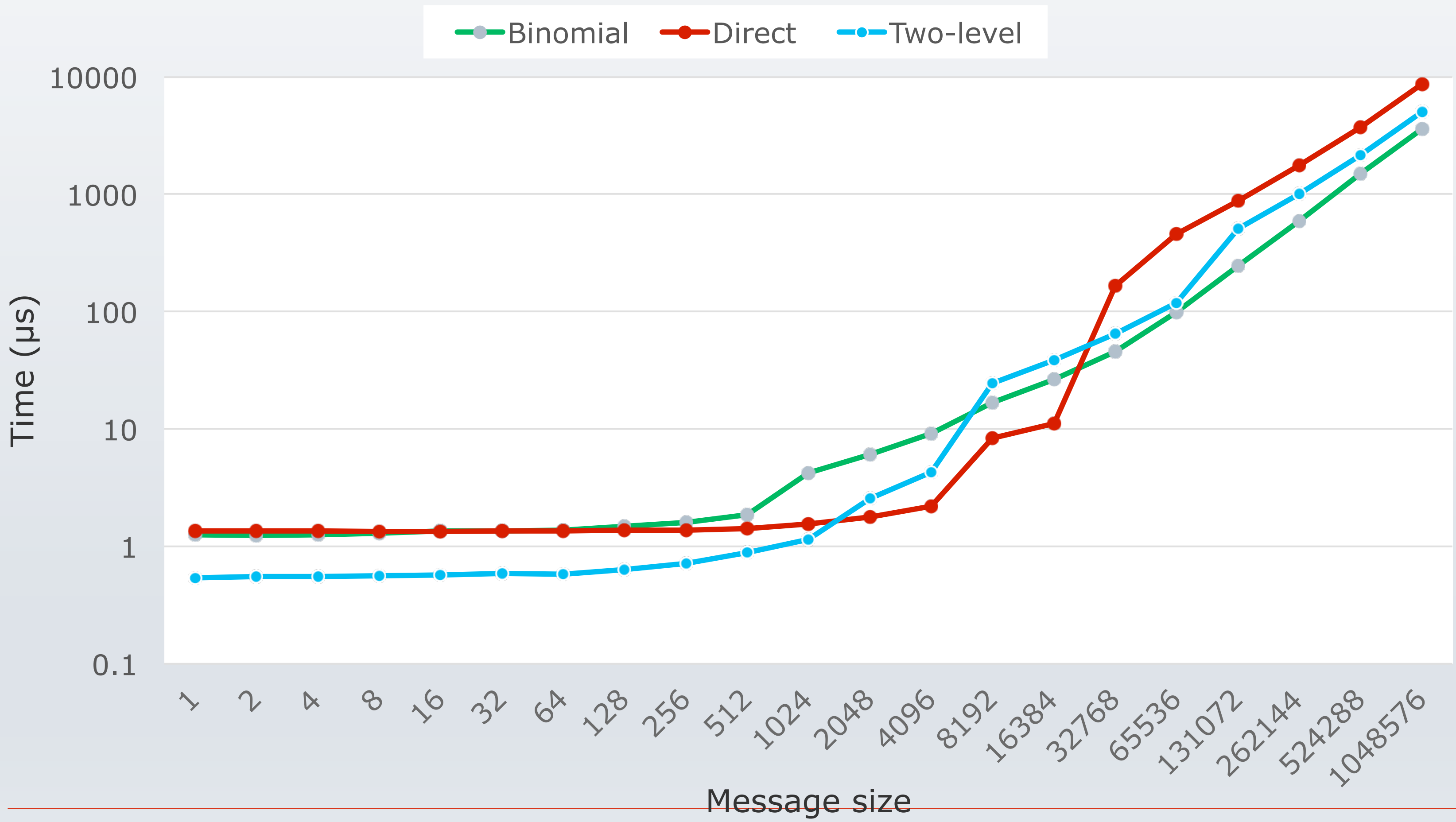
Goal of this presentation

- Why do we need different algorithms ?
- Which one is the best ? How to do the tuning ?
- Example of algorithms.
- What is the current state of research in this area ?

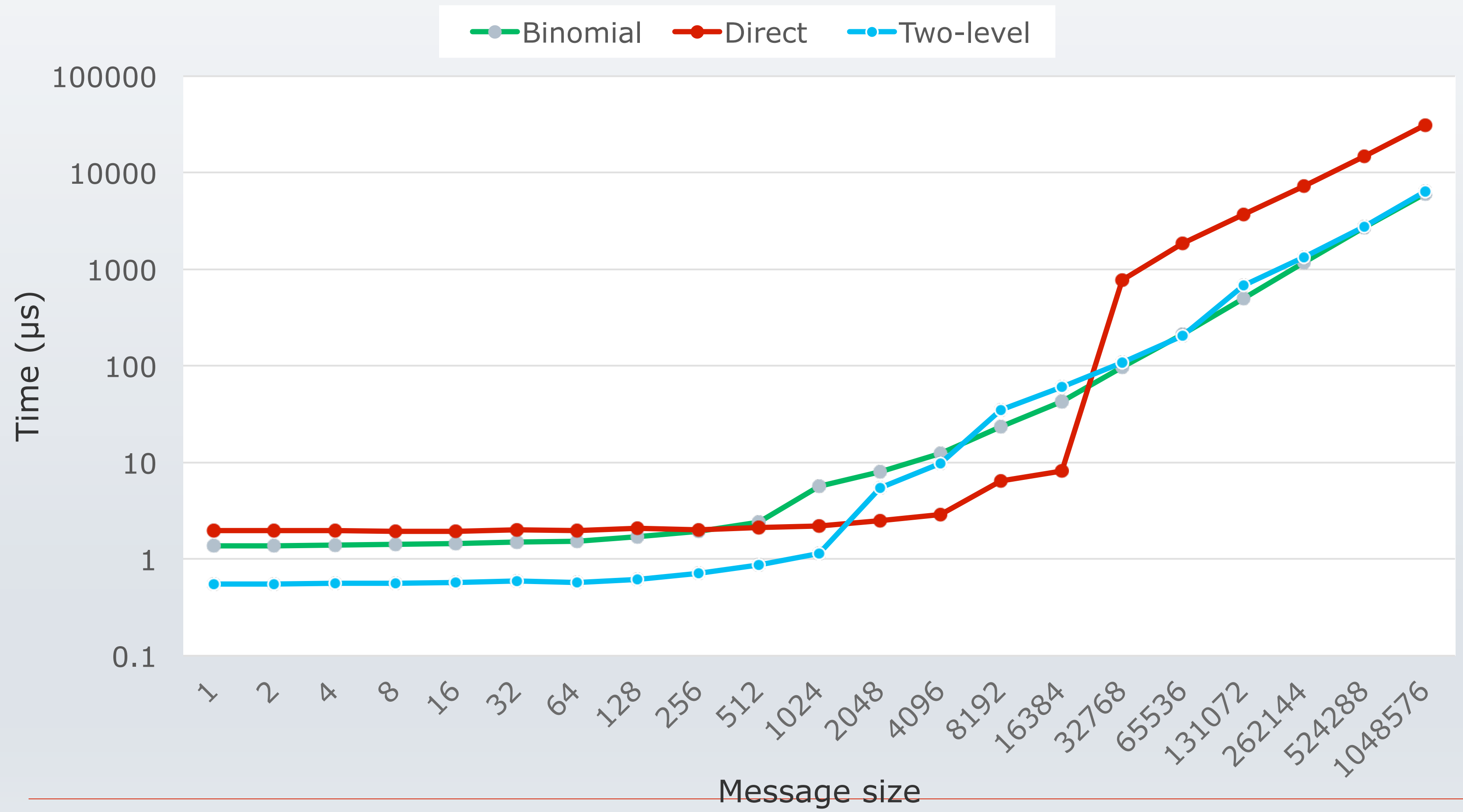
Behind Gather (64 cores/IMB)



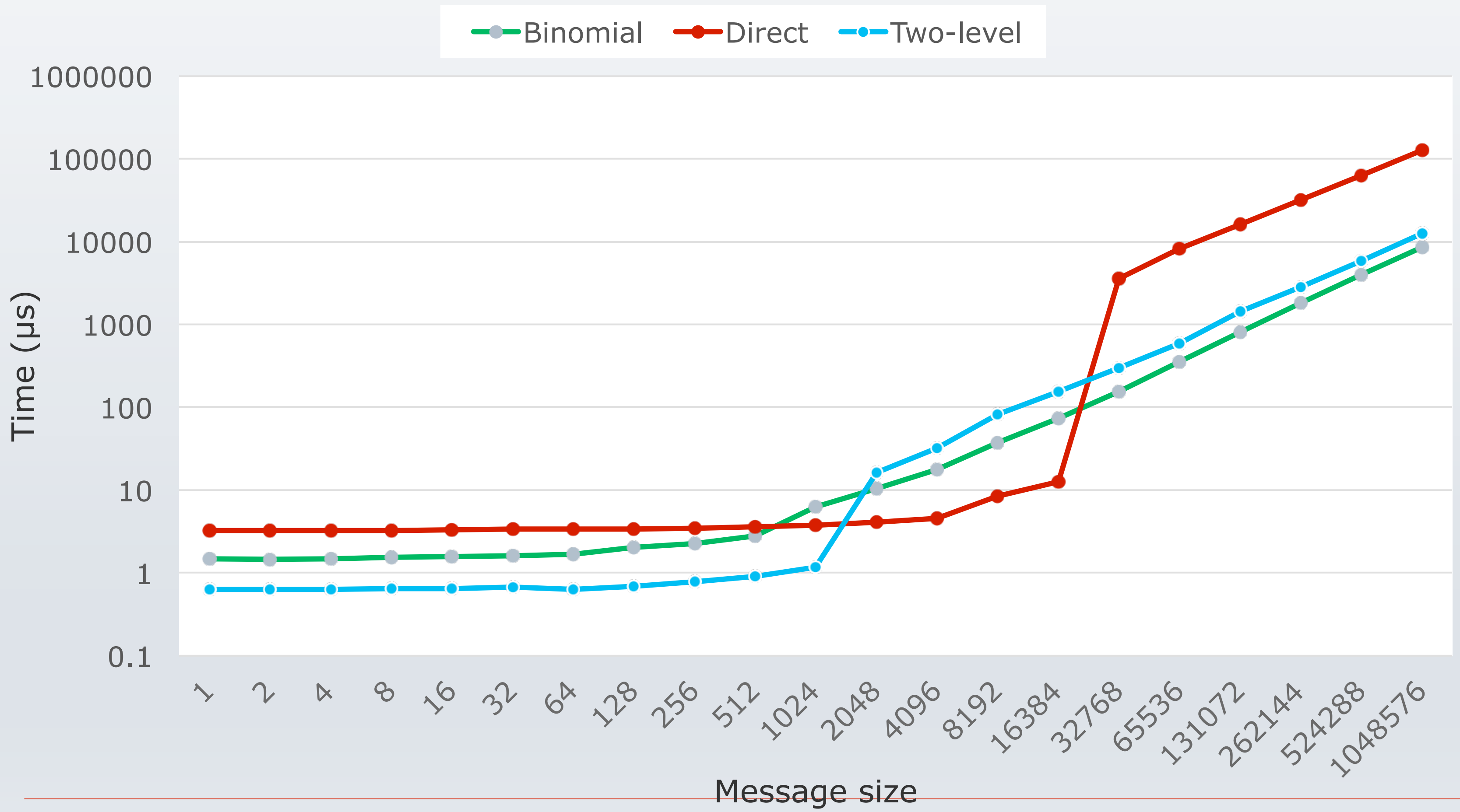
Behind Gather (64 cores/OMB)



Behind Gather (256 cores/OMB)



Behind Gather (1024 cores/OMB)



General Notes on Optimizing Collectives

- 2 components for collective communications – latency and bandwidth
- Latency(α) – time when the collective completes with the first byte
- Bandwidth(β) – rate at which collective proceeds after the first byte transmission
- Cost for communication – $\alpha+\beta m$ (Hockney)
- Latency is critical for small message sizes and bandwidth for large message sizes

MVAPICH Algorithm choices

- Depend on Platform
- Depend on system size
- Based on OSU Micro Benchmarks (OMB) results
- For each Platform:
  Step 1: Point-to-Point Tuning
  Step 2: Collective Tuning

Overview of algorithms

- Binomial Tree
- Barrier (Butterfly, Dissemination, Tournament…)
- Allgather
- Alltoall
- Reduce-Scatter for commutative operations: Recursive halving algorithm

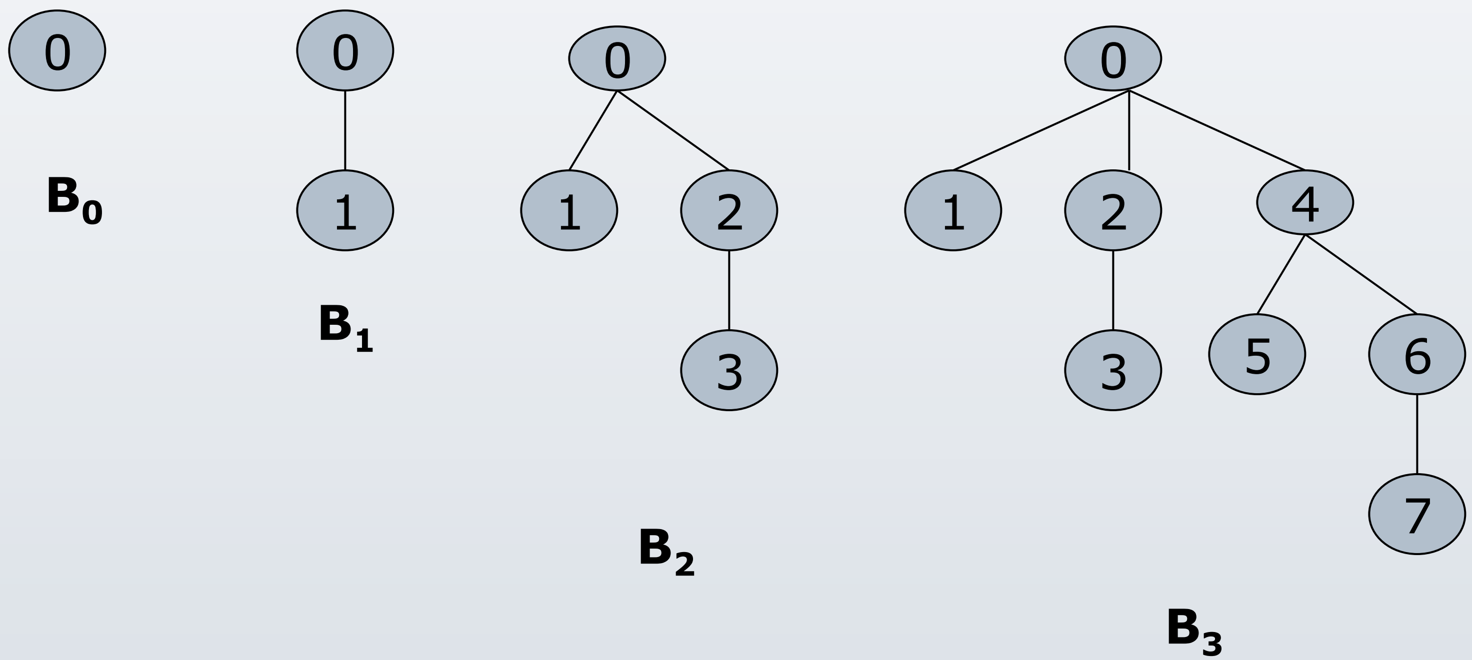
Binomial Tree

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

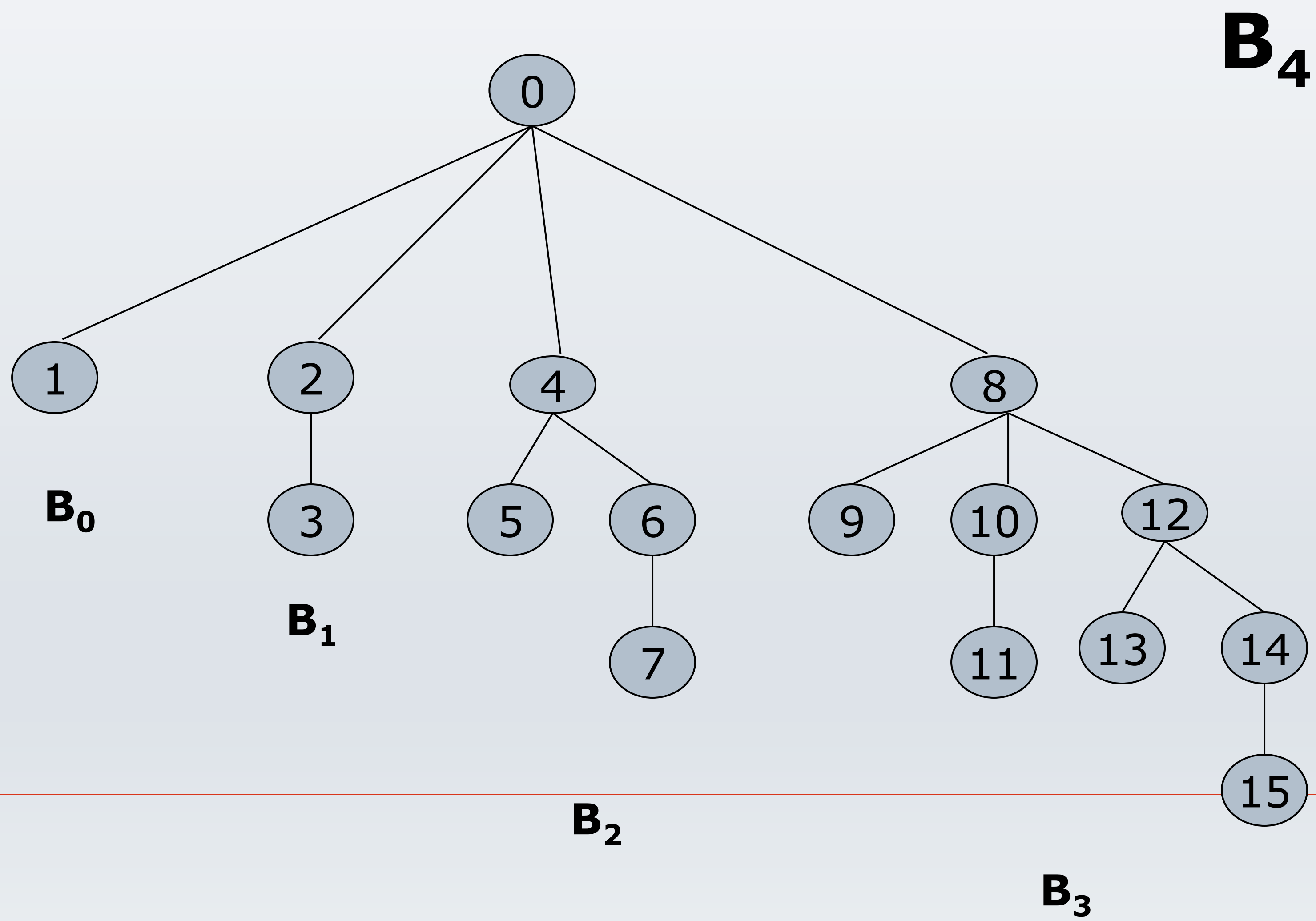
  1. If $k=0$, $B_k = \{R\}$. i.e., the binomial tree of order zero consists of a single node, *R*.

  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

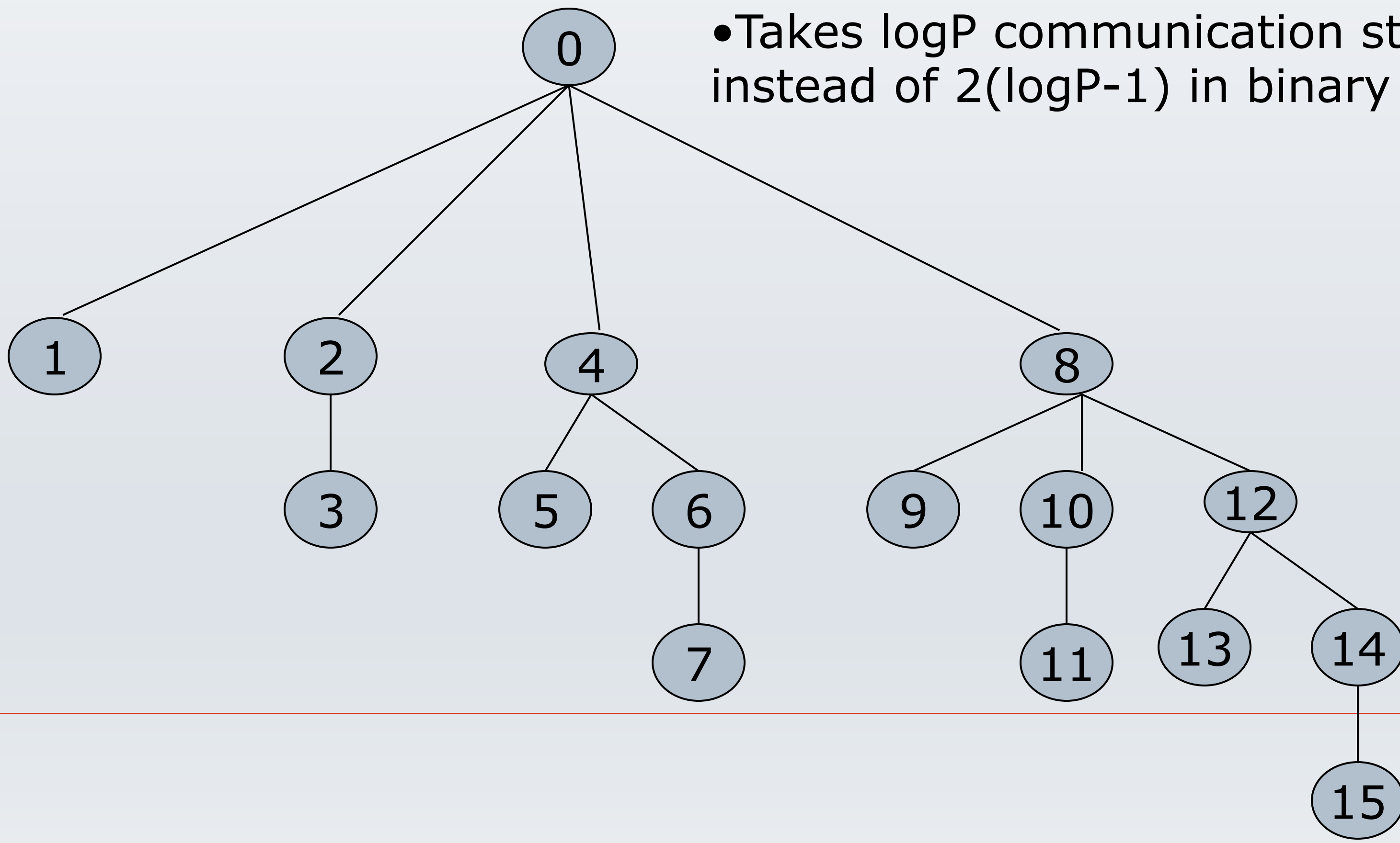


Binomial Trees



Binomial Trees

- Broadcast, Scatter and Gather usually implemented by binomial
- Takes logP communication steps instead of 2(logP-1) in binary



Barrier Algorithms

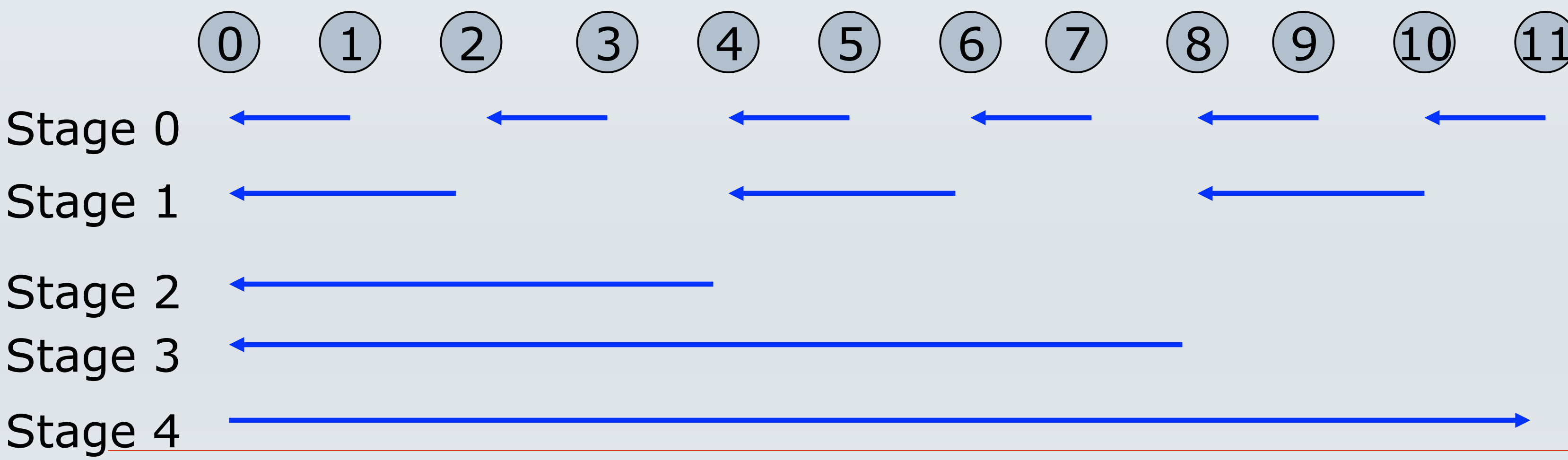
- **Butterfly barrier** by Eugene Brooks II
- In round k, i synchronizes with $i \oplus 2^k$ pairwise.
- If p not power of 2, existing procs. stand for missing ones.
- Worstcase – 2logP pairwise synchronizations by a processor

Barrier Algorithms

- **Dissemination barrier** by Hensgen, Finkel and Manser
- In round k, i signals $(i+2^k) \bmod P$
- No pairwise synchronization
- Same as butterfly but with different partners
- Almost log(next power of 2 > P) on critical path irrespective of P

Barrier Algorithms

- **Tournament barrier** by Hensgen, Finkel and Manser
- In the 1st round, each pair of nodes (players) synchronize (play a game)
- The receiver will be considered as the winner of the game
- In the 2nd round, the winners of the 1st round will synchronize (play games)
- The receiver in the 2nd round will advance to the 3rd 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 $i = j - 2^k$

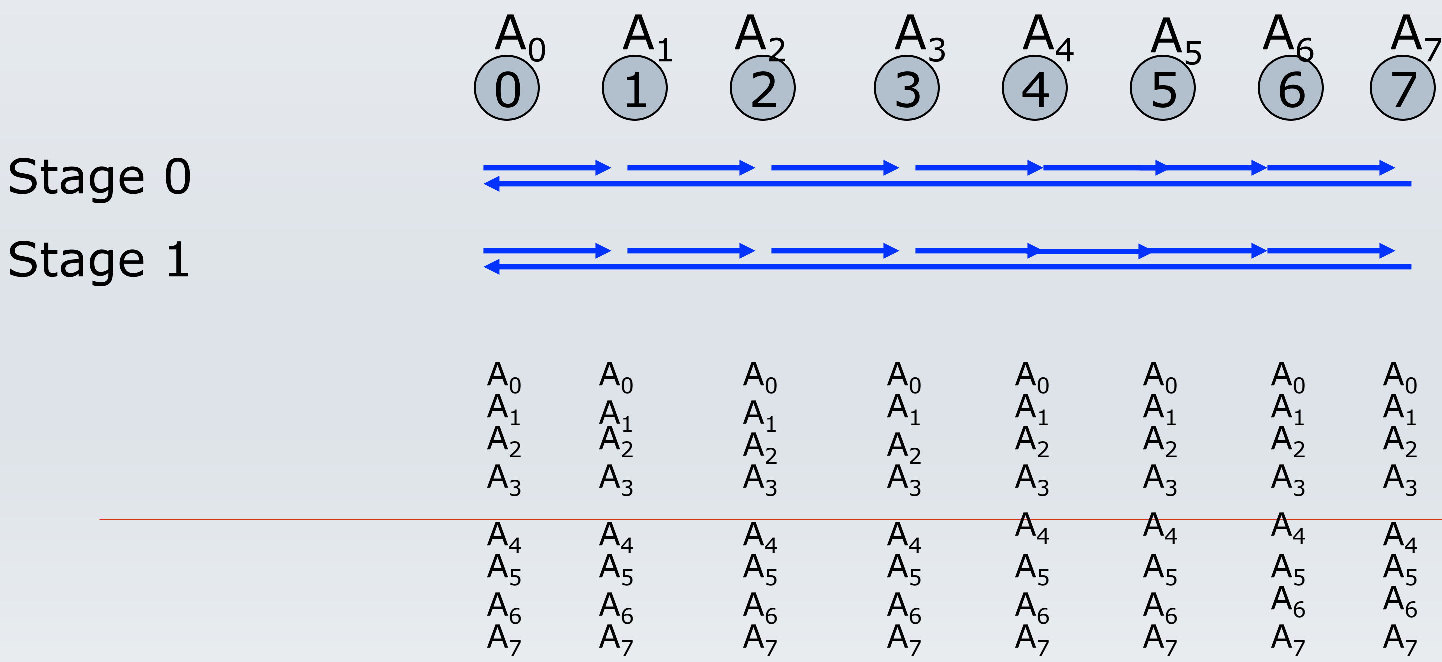


Barrier Algorithms

- **MVAPICH 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

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



Bruck’s Allgather

- Similar to dissemination barrier
- logP steps

0	1	2	3	4	5
A_0	A_1	A_2	A_3	A_4	A_5
A_0	A_1	A_2	A_3	A_4	A_5
A_1	A_2	A_3	A_4	A_5	A_0
A_2	A_3	A_4	A_5	A_0	A_1
A_3	A_4	A_5	A_0	A_1	A_2
A_4	A_5	A_0	A_1	A_2	A_3
A_5	A_0	A_1	A_2	A_3	A_4

AlltoAll

- The naive implementation

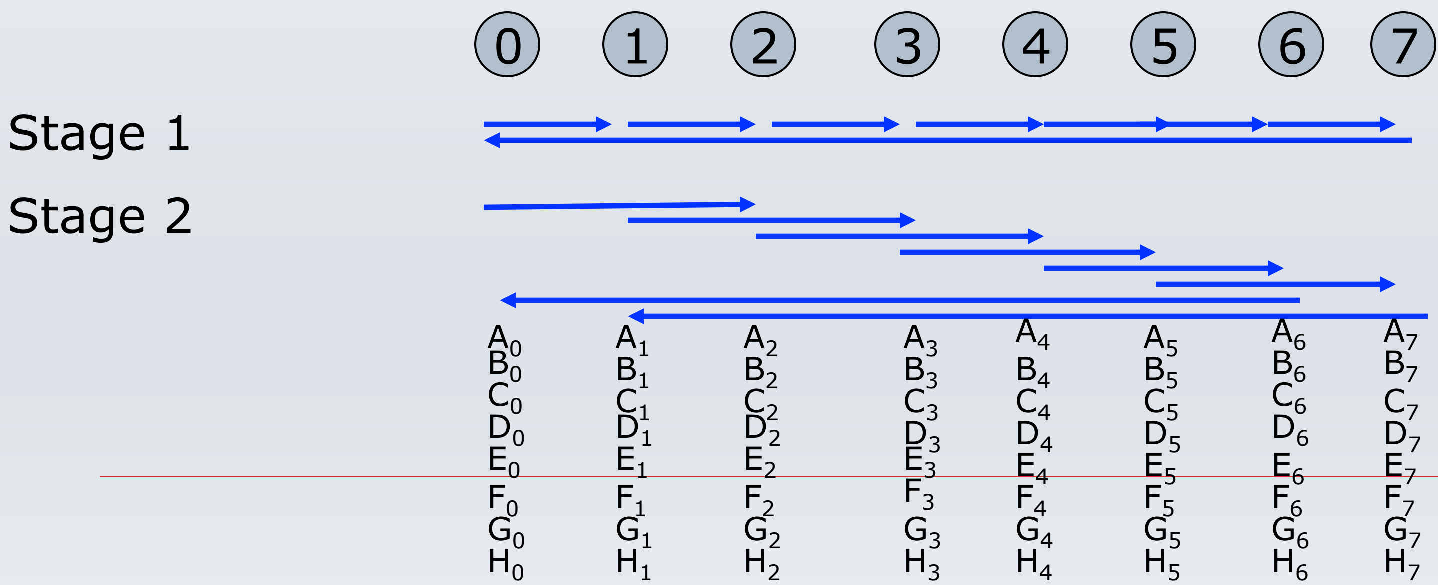
```
for all procs. i in order{
  if i # my proc., then send to i and recv from i
}
```

- 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
}
```

AlltoAll implementation

- Circular alltoall
- For step k in {1..P}, proc. i sends to (i+k)modP and receives from (i-k+P)modP



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

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…
- lgP 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…
- lgP steps

Example – Broadcast (MPICH)

- Binomial Broadcast
  - log p steps
  - Amount of data communicated at each step - n
  - cost = log p (α+nβ)
- scatter and allgather
  - Divide message into p segments
  - Scatter the p segments to p processes using binomial scatter – log p α + (n/p)(p-1) β
  - Scattered data collected at all processes using ring allgather – (p-1) α + (n/p)(p-1) β
  - cost = (log p + p-1) α + 2(n/p)(p-1) β
- Hence binomial broadcast for small messages and (scatter+allgather) for long messages

Two-level Algorithms of MVAPICH

- Take into account the node topology
- Notion of “node leader”
- Inter-node communications are between node leaders
- Intra-node communications are between the node leader and other processes
- Used for Bcast, Reduce, Gather...

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 allgather (> 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
  - Binomial algorithm (< 2 KB), Rabenseifner (> 2 KB)
- AllReduce
  - Recursive doubling (short) , Rabenseifner (long messages)

Current state of research

- Socket level leaders
- Auto - Selection of best algorithms (Tuning)
- Usage of CMA (Cross Memory Attach)
- Support for GPUs and MICs
- Topology aware collectives
- Energy Aware collectives
- Etc….

References

- Thakur et. al. – Optimization of Collective Communication Operations in MPICH. IJHPCA 2005.
- Thakur et. al. - Improving the Performance of Collective Operations in MPICH. EuroPVM/MPI 2003.
- http://mvapich.cse.ohio-state.edu/publications/