2018
Practical Parallel Computing
(実践的並列コンピューティング)
No. 10

Distributed Memory Parallel Programming with MPI (4)

Toshio Endo

School of Computing & GSIC

endo@is.titech.ac.jp

Considering Performance of MPI Programs

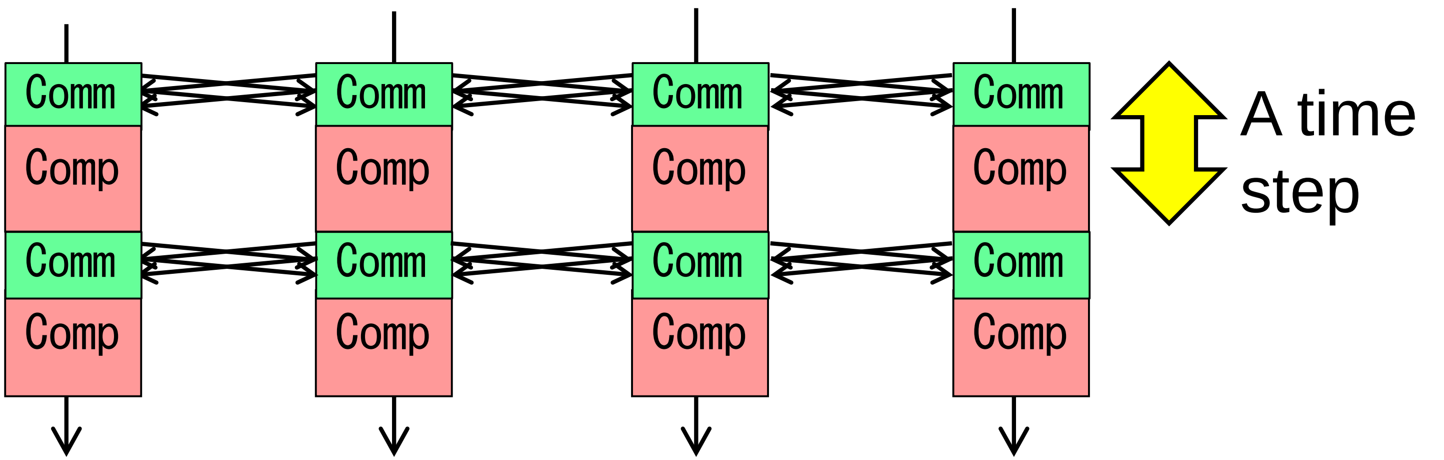
(Simplified) Execution time of an MPI program =

Computation time ← including memory access

\+ Communication time ← including congestion

\+ Others ← load imbalance, I/O…

Behavior of stencil computations on MPI



Computation Time & Communication Time (1)

How are they determined? (very simplified discussion)

1. Aspect of software

Computation time

- Longer if computation costs are larger
  - O(mnk/p) in matmul,
  - O(NX NY NT/p) in diffusion

per process

Communication time

- Longer if communication costs are larger
  - O(mk) in memory reduced matmul
  - O(NX NT) in diffusion

per process

Computation Time & Communication Time (2)

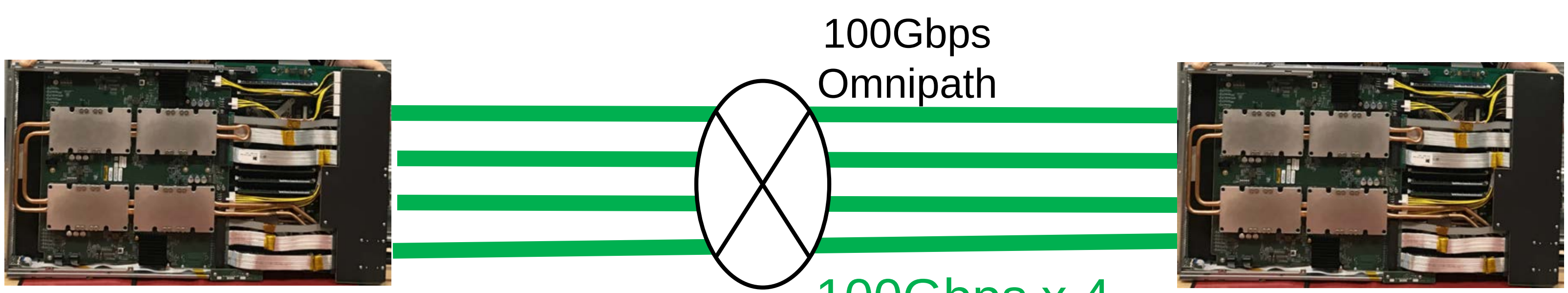
2. Aspect of hardware

Computation time

- Shorter if processor speed is faster
    - 140GFlops per node on TSUBAME2

Communication time

- Shorter if network speed is faster
    - 80Gbps per node on TSUBAME2



425GFlops x 2CPUs
= 850GFlops

100Gbps x 4
=400Gbps

425GFlops x 2CPUs
= 850GFlops

Speed of actual software is slower than the “peak” performance

Parameters for Network Speed

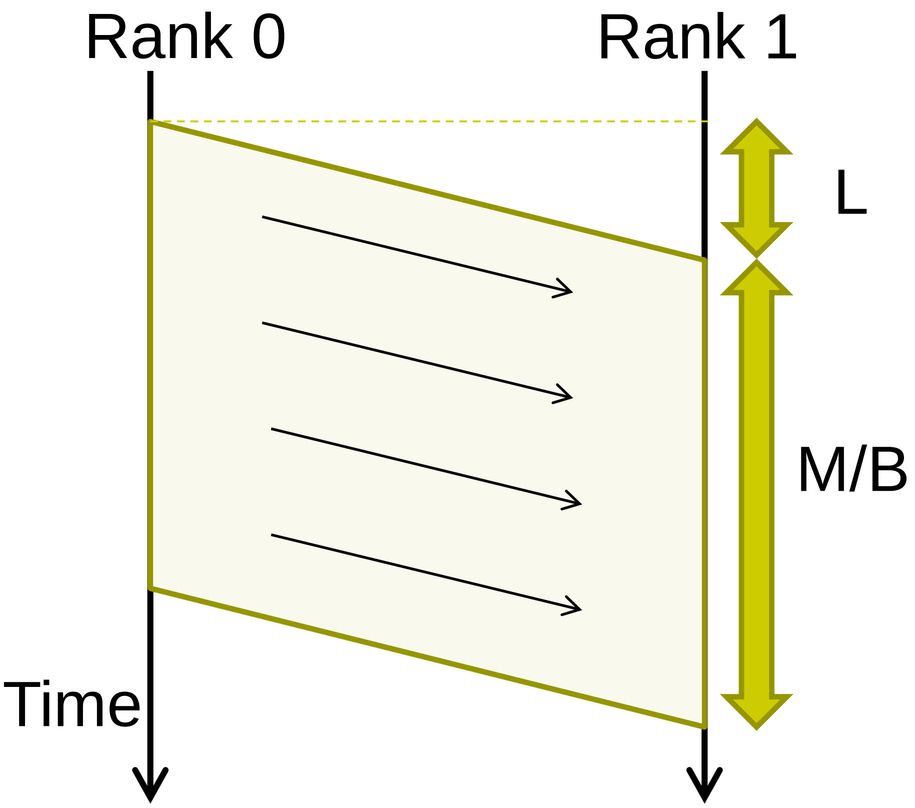
What parameters describes network speed?

- Bandwidth：Data amounts that network can transport per unit time → Larger is better
  - bps: X bits per second
  - B/s: X Bytes per second
  - On TSUBAME3, 400Gbps = 50GB/s per node
- Network latency：Time to transport minimum data (1bit, for example) → Smaller is better
  - On TSUBAME3, <10us

※ Additionally, communication time may suffer from effects of network topology: how nodes/switches are connected to each other

Bandwidth and Latency

Is “latency” reciprocal of “bandwidth”?
→ No, because data are transported in “pipe-lined” style



$$T = M / B + L$$

T: Communication time

M: Data size

B: Bandwidth

L: Network latency

※ Be aware of difference between “Byte” and “bit”: 1Byte=8bit

※ In some contexts, T, not L, may be called “latency”

Why L (Latency) > 0?

1. Overhead when data passes network switches

2. Software overhead
   - Cf) Socket library, MPI library performs data copy

3. Transfer speed of data cannot exceed speed of light (3×10^8 m/s)

Considering T = M / B + L,
batching communication may improve communication time

cf) Sending 1Gbytes at once is much faster than sending 1Kbytes for 1,000,000 times

How to Improve Performance of MPI Programs?

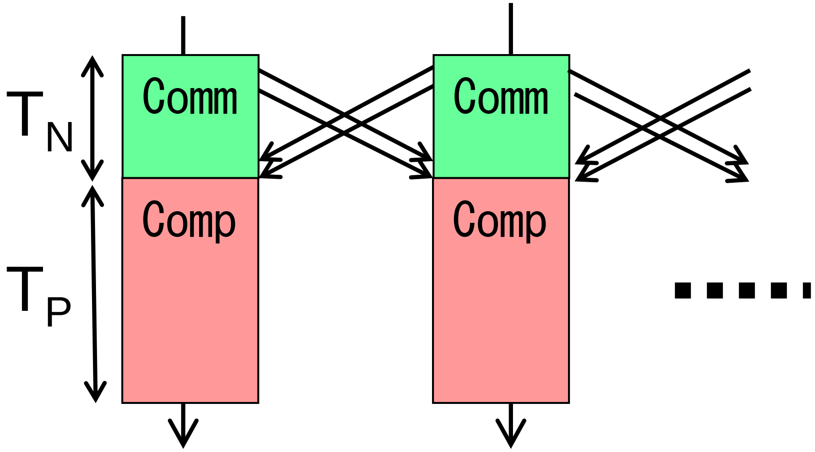
- Reduce computation time
  - Reduce computation amount
  - Using cache memory efficiently
- Reduce communication time
  - Reduce communication amount
  - Batch communication
  - Using collective communication is also good
- Reduce other time
  - Improve load balancing
  - Reconsider I/O

… And overlap computation and communication

Idea of Overlapping

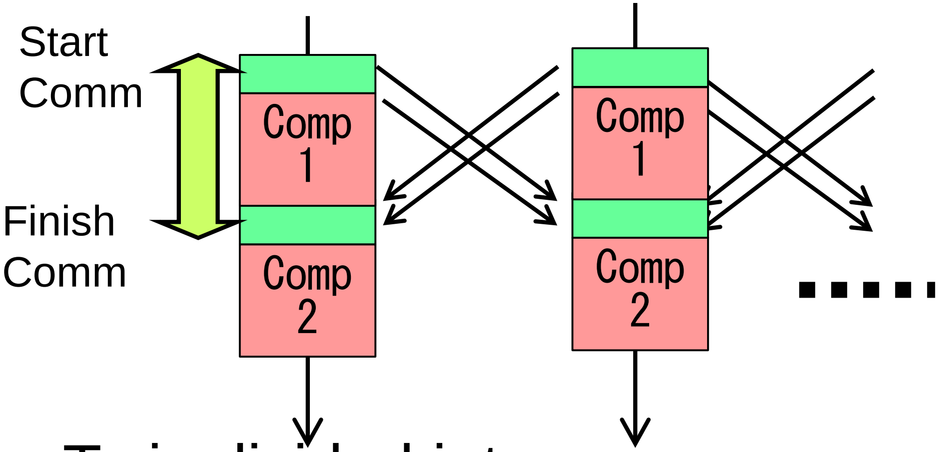
If "some computations" do not require contents of message, we may start them beforehand

Without overlap



$T = T_N + T_P$

With overlap



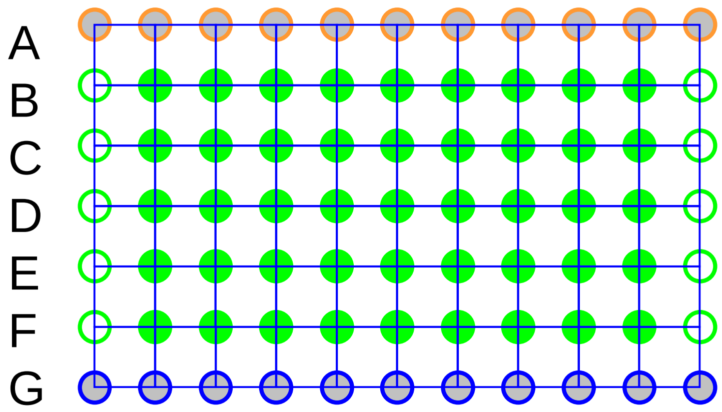
T_P is divided into

- T_{P1}: can be overlapped
- T_{P2}: cannot be overlapped

$T = \max(T_N, T_{P1}) + T_{P2}$

Overlapping in Stencil Computation
(related to [M1], but not requied)

When we consider data dependency in detail, we can find computations that do not need data from other processes



Rows C, D, E do not need data from other processes
→ They can be computed without waiting for finishing communication

On the other hand, rows B, F need received data

For such purposes, non-blocking communications (MPI_Isend, MPI_Irecv…) are helpful

Implementation without Overlapping (Not Fast!)

for (t = 0; t < nt; t++) {

Start Send B to rank-1, Start Send F to rank+1 (MPI_Isend)

Start Recv A from rank-1, Start Recv G from rank-1 (MPI_Irecv)

Waits for finishing all communications (MPI_Wait)

Compute rows B--F

Switch old and new arrays

}

T_N

T_P

$$T = T_N + T_P$$

Implementation <u>with</u> Overlapping

for (t = 0; t < nt; t++) {

- Start Send B to rank-1, Start Send F to rank+1 (MPI_Isend)
- Start Recv A from rank-1, Start Recv G from rank-1 (MPI_Irecv)
- Compute rows C--E
- Waits for finishing all communications (MPI_Wait)
- Compute rows B, F
- Switch old and new arrays

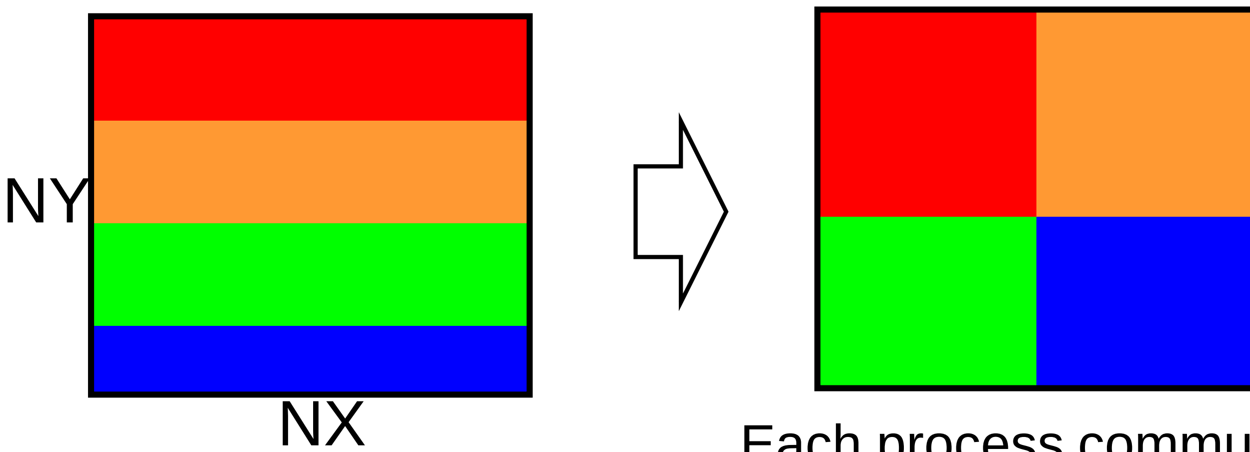
}

computations are divided

$$T=\max(T_N,T_{P1})+T_{P2}$$

Another Improvement:
Reducing Communication Amounts

Multi-dimensional division may reduce communication



Each process communicate with
upper/lower/right/left processes

- Comp: O(NY NX/p)
- Comm: O(NX)

per 1 process, 1 iteration

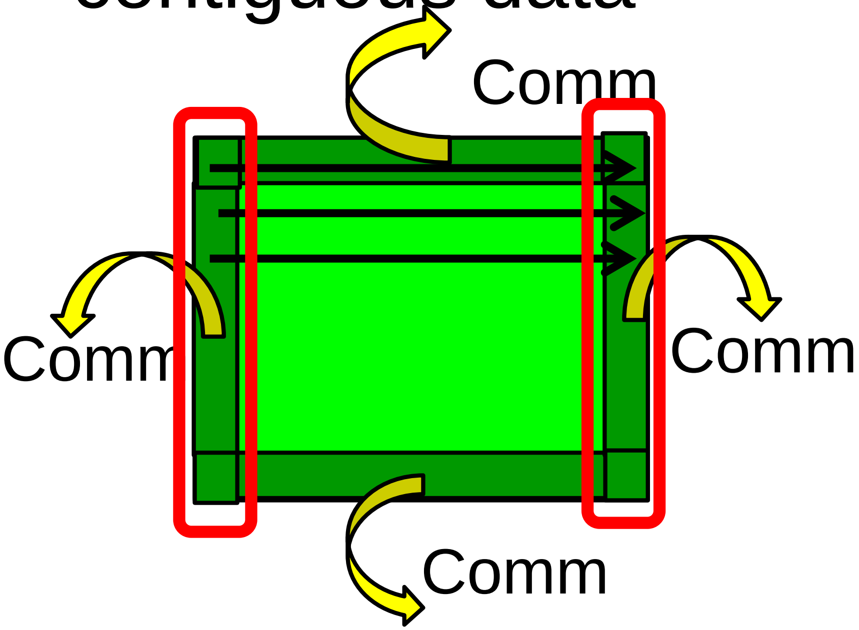
- Comp: O(NY NX/p)
- Comm: $O((NY+NX)/p^{1/2})$

per 1 process, 1 iteration

→ Comm is reduced

Multi-dimensional division and Non-contiguous data (1)

- MD division may need communication of non-contiguous data



In Row-major format,
we need send/recv of non-contiguous data for left/right borders

But “fragmented communication” degrades performance! (since Latency > 0)
How do we do?

Multi-dimensional division and Non-contiguous data (2)

Solution (1):

- Before sending, copy non-contiguous data into another contiguous buffer
- After receiving, copy contiguous buffer to non-contiguous area

Solution (2):

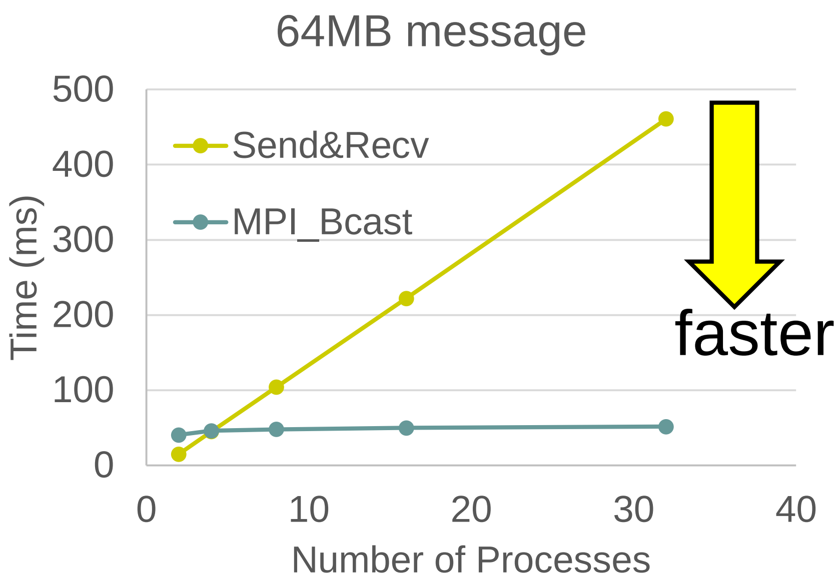
- Use MPI_Datatype
  - Skipped in the class; you may use Google :-p

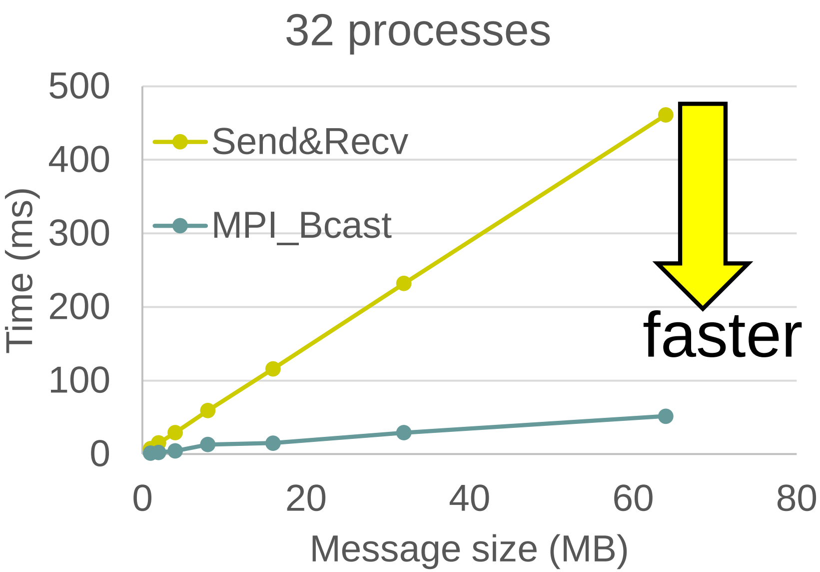
It is Better to Use Collective Communications if Appropriate

- Comparing MPI_Bcast and MPI_Send&Recv

  1 process per node is invoked (to measure network)

  In the latter, rank 0 called MPI_Send for p-1 times to other processes



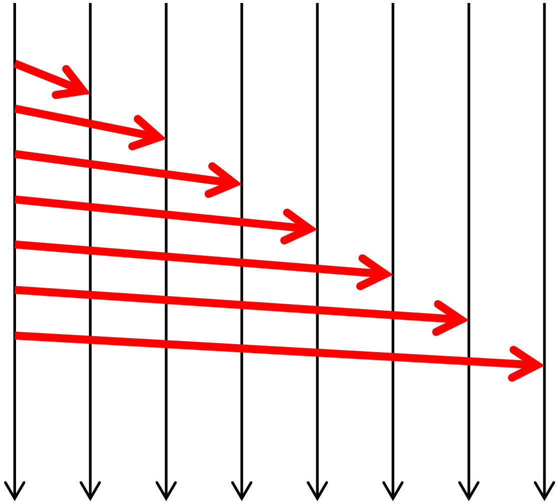


- In most cases, MPI_Bcast is faster

Why are Collective Communications Fast?

- Since Scalable communication algorithms are used inside MPI library

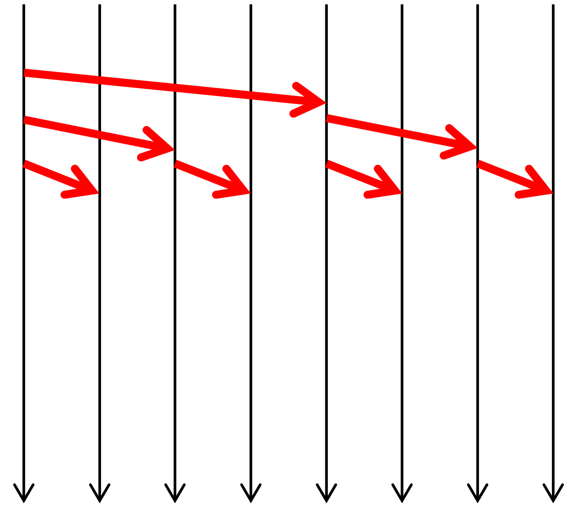
Flat tree algorithm



(p-1)(M/B+L)

→ *Slowest*

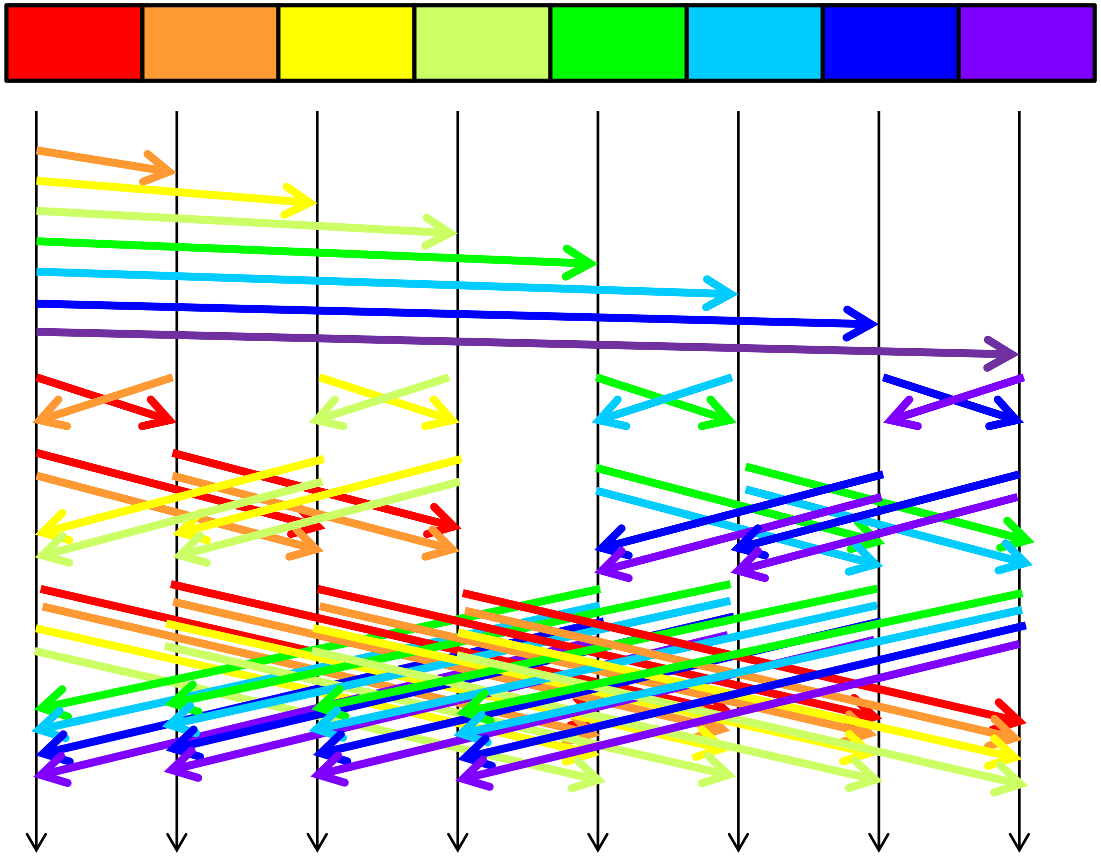
Binomial tree algorithm



(log p)(M/B+L)

One of Scalable "Bcast" Algorithms

- Scatter&Allgather algorithm
  - Message is divided into p parts
  - Better than "binomial tree" if M is larger



pL + M/B +
(log p)L + M/B

R. Thakur and W. Gropp. Improving the performance of collective operations in mpich. EuroPVM/MPI conference, 2003.

- We have finished
  - Part 1: OpenMP for shared memory parallel programming
  - Part 2: MPI for distributed memory parallel programming

- Why are “parallel programs” slower than expectation?
  - “p times speed-up with p processor cores” (linear scaling) is ideal, but…
  - parallel software is often less scalable

Too Many Factors that Limit Performance of Programs

- Factors in algorithm
  - Load imbalance between threads, processes
  - Bottlenecks due to mutual exclusions
  - Communication costs
- Factors related to OpenMP/MPI system
  - Too many parallel region
  - Too many message
- Factors related to hardware
  - Memory access costs
  - Congestion in network

and many, many factors

How Should We Tackle Performance Limiting Factors?

- It is important to know “why it is slow now”
- Consider what should be measured in order to specify current problem
  - Measuring time part by part may be helpful
  - Comparing computation time and communication time separately
  - Comparing 1-node performance and multi-node performance may be helpful
- It is good to use knowledge of computer hardware

Assignments in MPI Part (Abstract)

Choose *one of* [M1]—[M3], and submit a report

Due date: May 28 (Monday)

[M1] Parallelize "diffusion" sample program by MPI.

[M2] Improve mm-mpi sample in order to reduce memory consumption.

[M3] (Freestyle) Parallelize *any* program by MPI.

For more detail, please see No. 7 slides or OCW-i.

Next Class

- Part 3 starts
  - GPU parallel programming
    - OpenACC is planned