MPI: Basic Performance Considerations

Thomas Hauser, thomas.hauser@colorado.edu
Nick Featherstone, feathern.colorado.edu
University of Colorado Boulder

Mar 14-16, 2017

Outline

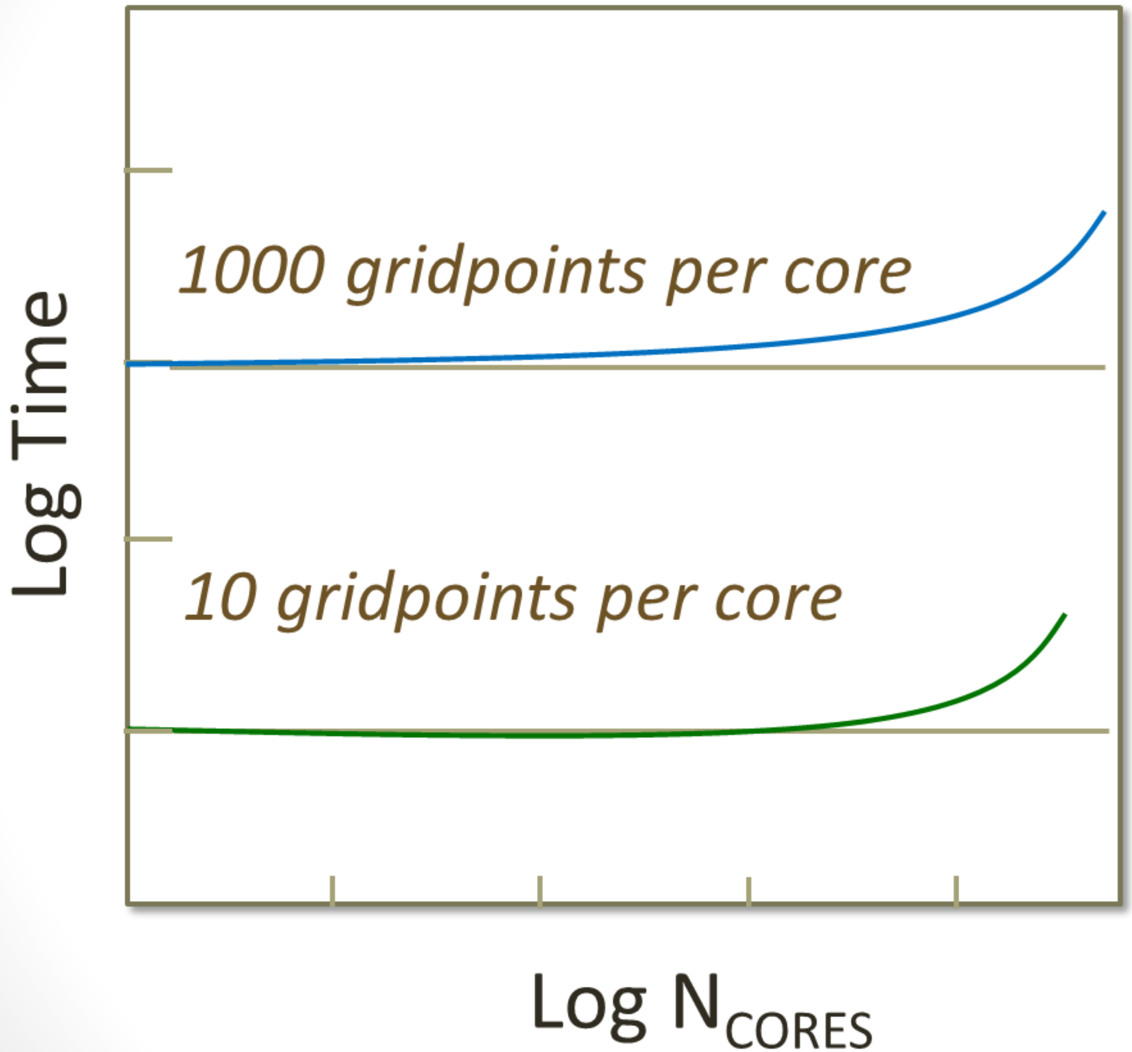
- Strong/Weak Scaling
- MPI Subcommunicators
- Domain Decomposition / Process Grids
- Optimal message-passing considerations
- Load-balancing 2-D diffusion problem

Measuring Performance

- Performance typically measured using two metrics:
  - Weak scaling analysis:
    - Increase problem size and process count together
  - Strong scaling analysis:
    - Fix problem size and increase process count
- In each instance, measure performance* for a series of process counts.

* (e.g., FLOPS, time steps per second, etc.)

Weak Scaling



Best Case:

$$\text{Time} = \text{Constant}$$

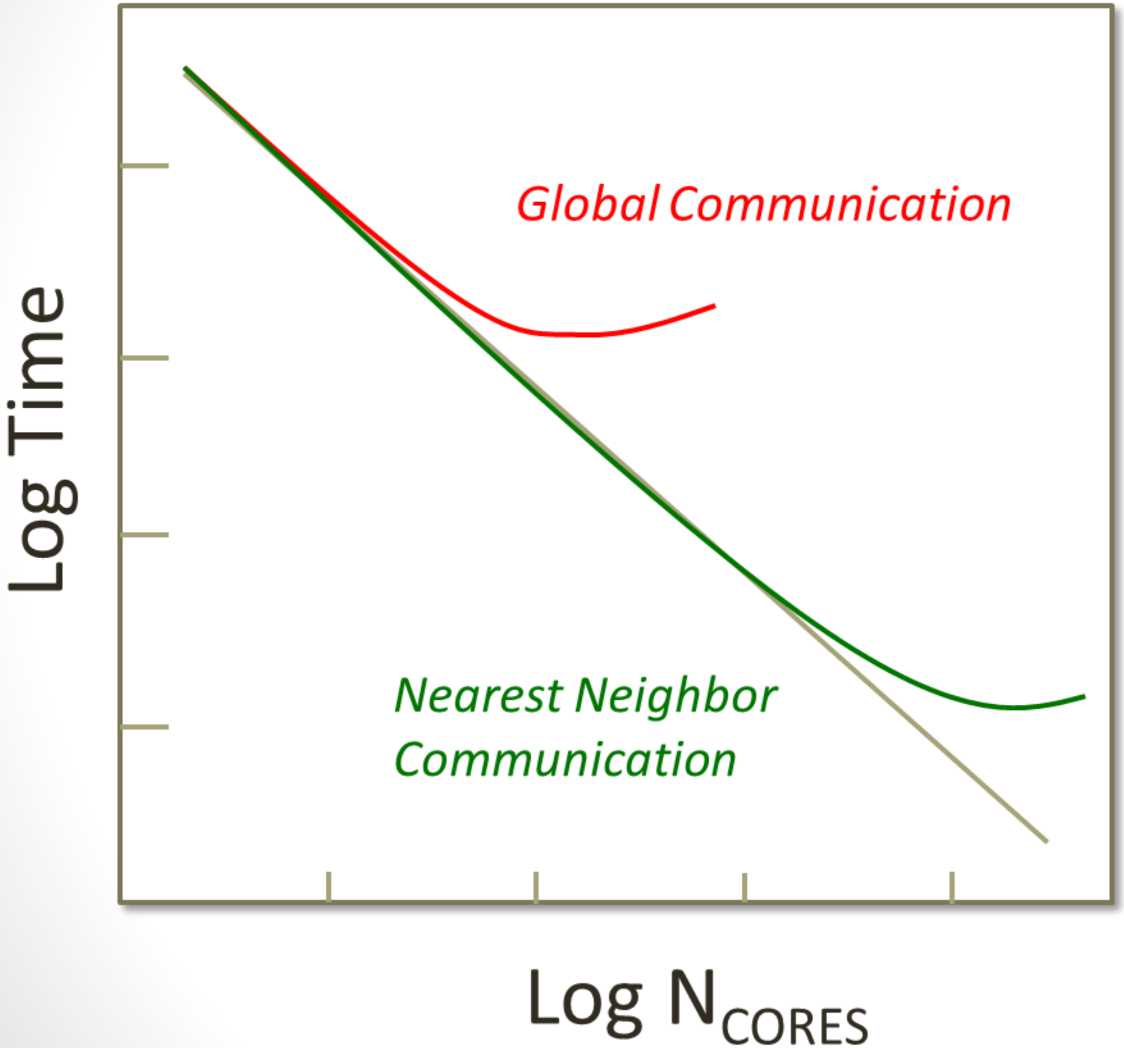
(when work per core constant)

Weak Scaling

- Weak scaling is VERY problem dependent

- Non-ideal scaling can occur due to factors such as
  - Mathematical operations that grow nonlinearly with problem size (e.g. matrix multiplication)
  - Memory overhead that grows nonlinearly relative to problem size

Strong Scaling



Ideal Scaling:

$$\text{Time} \propto \frac{1}{N_{CORES}}$$

The game:

mitigation

Messaging Time

Communication Time = *Initiation Time* + *Transmission Time*

Global Problem Size: G
Number of MPI Ranks: N
# of Ghost Zones: f(G/N)
Single Message Initiation Time: I
Bandwidth: B

Transmission Time = f(G/N)/ B = … *decreasing*

Initiation Time = I … *constant*

General Strategy: *always try* to limit message count

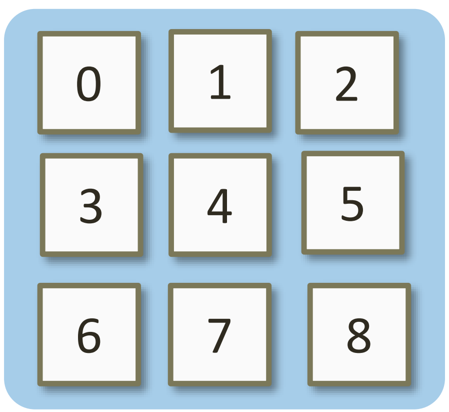
Best Practices

- Non-blocking sends and receives (almost always!)
- Bundle messages when possible
- Process Grid + Related Messaging patterns
  - Sub-communicators
  - Also useful for file I/O

Process Grids

- MPI ranks numbered sequentially from 0 through N -1

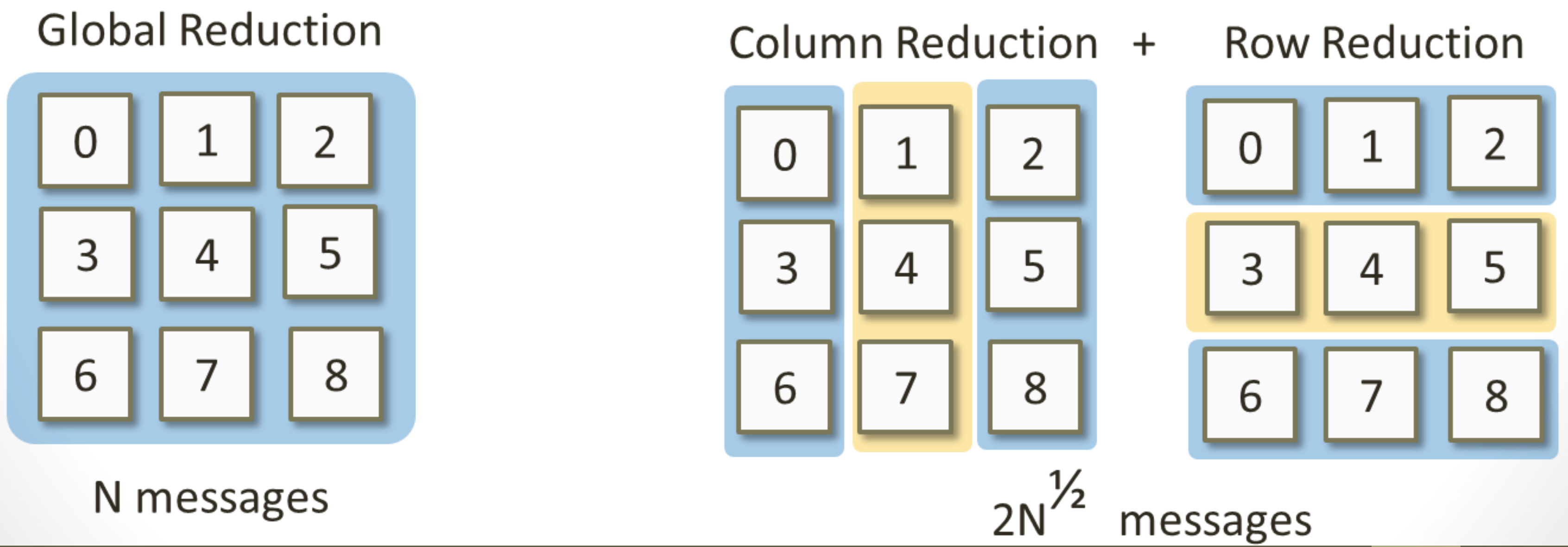
- We can envision them as being arranged in some pattern. For instance, a grid…



- Communicate only along rows or columns
- Why is this advantageous?

Process Grids

- Programming communication becomes more natural.
- E.g., think in terms of rank within row or column
- Useful for I/O (only subset of process grid reads/writes)
- Can limit message count for global communications



Recall: MPI Communicator

- A collection of processors of an MPI program
- Used as a parameter for most MPI calls.
- Processors with in a communicator have a number
  - Rank: 0 to n-1
- `MPI_COMM_WORLD`
  - Contains all processors of your program run
- You can create **sub-communicators** that are subsets
  - All even processors
  - The first processor
  - All but the first processor

Sub-communicator Creation

- call MPI_COMM_SPLIT(

comm,	the communicator to split (e.g., MPI_COMM_WORLD),
color,	integer; same color grouped into same new communicator
key,	integer; processes with the same color assigned sub-communicator rank based on key.
newcomm,	integer; the sub-communicator we wish to create
ierr	error tag (return value)

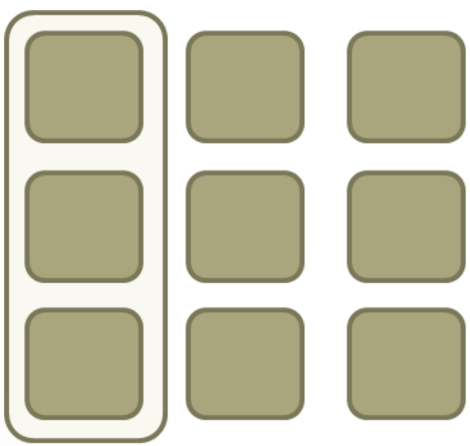
)

All arguments are inputs (except ierr).

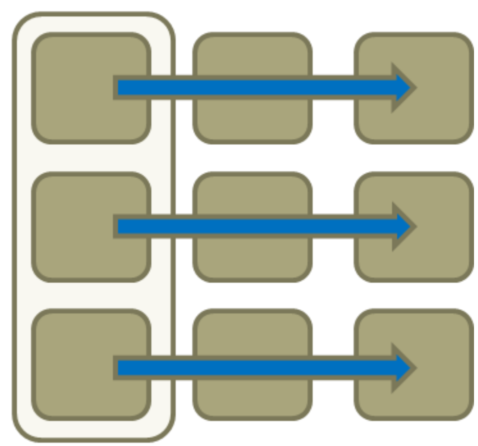
Example: Process Grid I/O

- Examine the code in:

  MPI/Lab/session2/examples/row_read.f90
- Only processes in column 0 access input file.
- Column 0 broadcasts to other columns
- Note the use of the row_comm sub-communicator



Step 1: Read



Step 2: Broadcast

Broadcasting

- call MPI_COMM_SPLIT(

buffer,	data to broadcast
count,	number of elements to broadcast
datatype,	integer; e.g., MPI_INTEGER
root,	integer; rank of broadcaster
comm	the communicator to broadcast across (e.g., MPI_COMM_WORLD)
Ierr	error flag

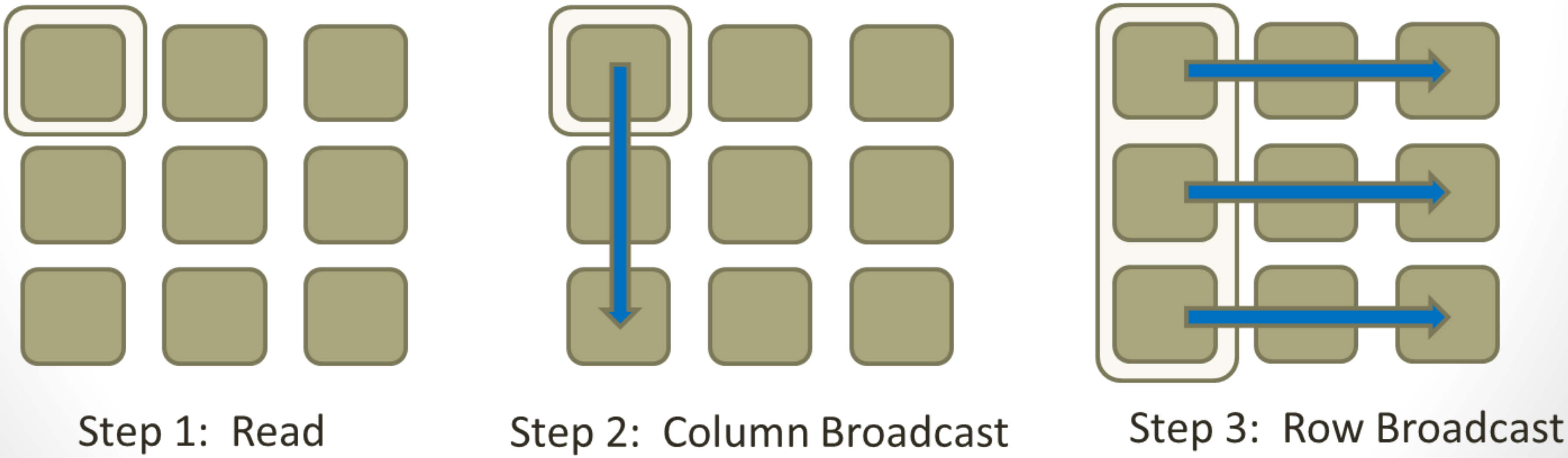
)

Exercise: Process Grid I/O

- Examine the code in:

  MPI/Lab/session2/exercise1/ex1.f90

- Modify this program that so
  - Rank 0 broadcasts the input data to column 0.
  - Lead process in each column broadcasts input value to its row.



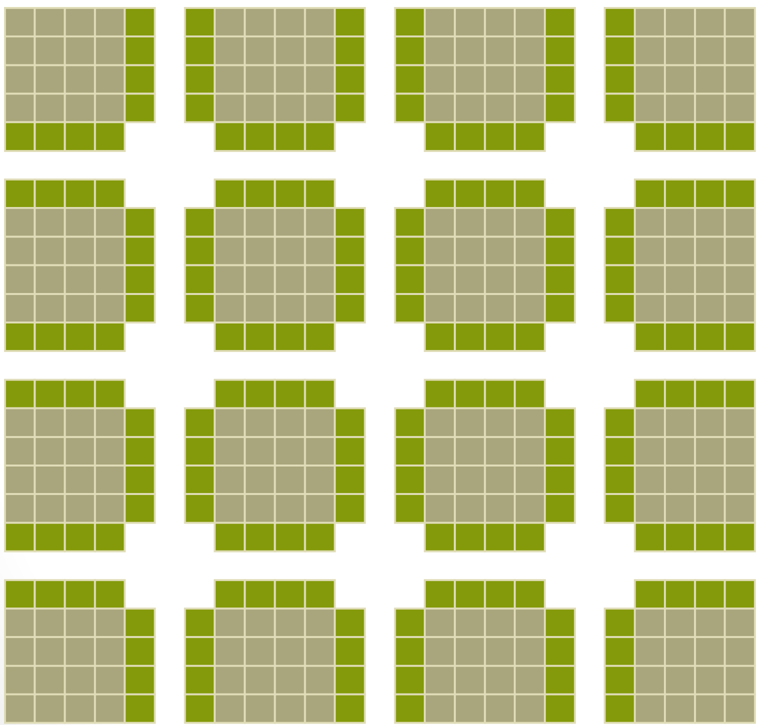
Exercise: Process Grid Reduction

- Examine the code in:

  MPI/Lab/session2/exercise2/ex2.f90
- Each process computes the maximum of its local portion of the global array var.
- Modify this program that so that processes reduce across rows and columns to find the global maximum.

Exercise 3: 2-D Diffusion Problem

$$f_{x,y,t+1} = \tfrac{1}{4}\left(f_{x-1,y,t} + f_{x+1,y,t} + f_{x,y-1,t} + f_{x,y+1,t} \right)$$



- Similar to 1-D, but **four *sets*** of ghost zones.
- Examine the code in:

  ../exercise3/ex3.f90
- Fill in the body of the ghost_zone_comm routine as appropriate for 2-D.

Exercise 4: 3-D Diffusion Problem

- Examine the code in:

  MPI/Lab/session3/exercise4/ex4.f90

- Load-balancing is still based on 2-D process grid. Each process owns all z-levels.

- Modify this code in two stages:
  - (a) Convert the sends/recvs to Isends/Irecvs
  - (b) Pack ghost zones for all z-levels into single buffer

- At each stage, produce weak- and strong-scaling curves for a range of problem sizes and process counts.