Advanced MPI Programming

Tutorial at SC14, November 2014

Latest slides and code examples are available at

www.mcs.anl.gov/~thakur/sc14-mpi-tutorial

Pavan Balaji
Argonne National Laboratory
Email: balaji@mcs.anl.gov
Web: www.mcs.anl.gov/~balaji

William Gropp
University of Illinois, Urbana-Champaign
Email: wgropp@illinois.edu
Web: www.cs.illinois.edu/~wgropp

Torsten Hoefler
ETH Zurich
Email: htor@inf.ethz.ch
Web: http://htor.inf.ethz.ch/

Rajeev Thakur
Argonne National Laboratory
Email: thakur@mcs.anl.gov
Web: www.mcs.anl.gov/~thakur

ETH Zürich

Outline

Morning

- Introduction
  - MPI-1, MPI-2, MPI-3
- Running example: 2D stencil code
  - Simple point-to-point version
- Derived datatypes
  - Use in 2D stencil code
- One-sided communication
  - Basics and new features in MPI-3
  - Use in 2D stencil code
  - Advanced topics
    - Global address space communication

Afternoon

- MPI and Threads
  - Thread safety specification in MPI
  - How it enables hybrid programming
  - Hybrid (MPI + shared memory) version of 2D stencil code
- Nonblocking collectives
  - Parallel FFT example
- Process topologies
  - 2D stencil example
- Neighborhood collectives
  - 2D stencil example
- Recent efforts of the MPI Forum
- Conclusions

Advanced Topics: Nonblocking Collectives, Topologies, and Neighborhood Collectives

Nonblocking Collective Communication

- Nonblocking (send/recv) communication
  - Deadlock avoidance
  - Overlapping communication/computation
- Collective communication
  - Collection of pre-defined optimized routines
- → Nonblocking collective communication
  - Combines both techniques (more than the sum of the parts ☺)
  - System noise/imbalance resiliency
  - Semantic advantages
  - Examples

Nonblocking Collective Communication

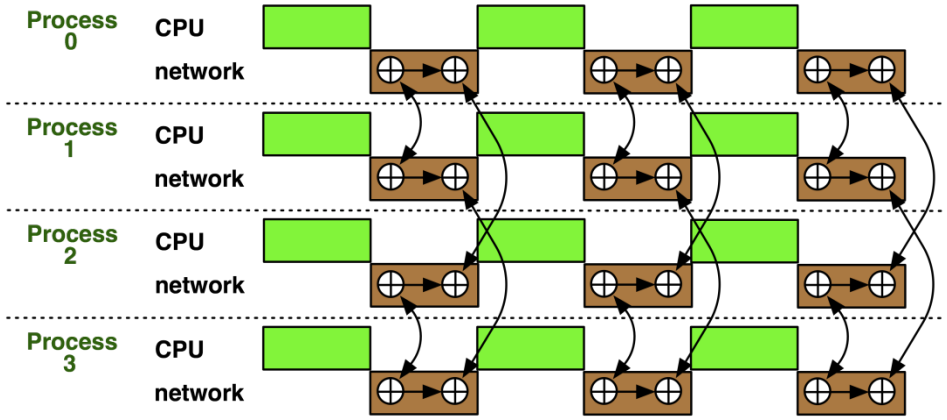
- Nonblocking variants of all collectives
  - MPI_Ibcast(<bcast args>, MPI_Request *req);
- Semantics
  - Function returns no matter what
  - No guaranteed progress (quality of implementation)
  - Usual completion calls (wait, test) + mixing
  - Out-of order completion
- Restrictions
  - No tags, in-order matching
  - Send and vector buffers may not be touched during operation
  - MPI_Cancel not supported
  - No matching with blocking collectives

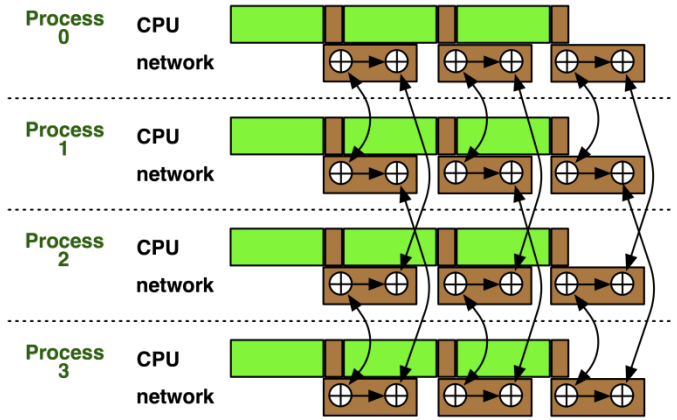
Nonblocking Collective Communication

- Semantic advantages
  - Enable asynchronous progression (and manual)
    - Software pipelinling
  - Decouple data transfer and synchronization
    - Noise resiliency!
  - Allow overlapping communicators
    - See also neighborhood collectives
  - Multiple outstanding operations at any time
    - Enables pipelining window

Nonblocking Collectives Overlap

- Software pipelining
  - More complex parameters
  - Progression issues
  - Not scale-invariant



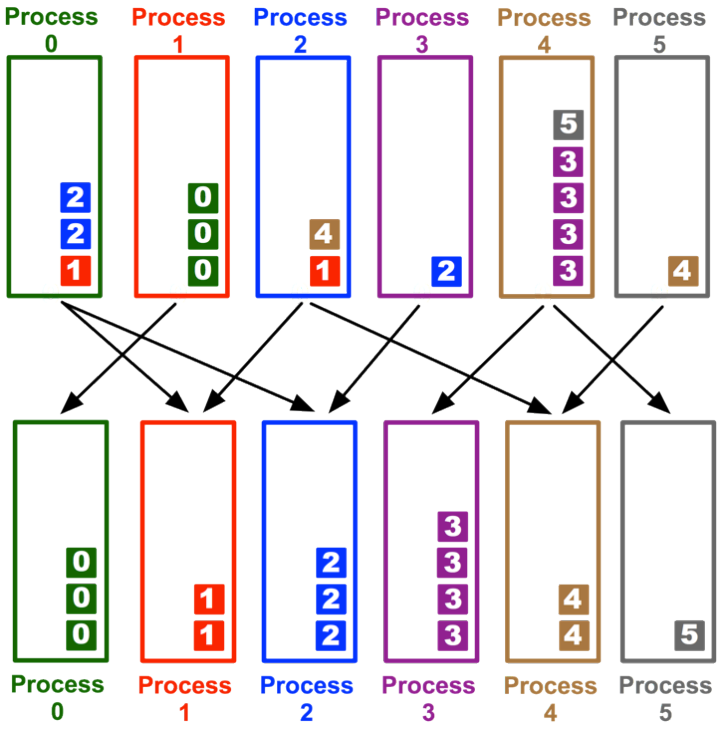


A Non-Blocking Barrier?

- What can that be good for? Well, quite a bit!
- Semantics:
  - MPI_Ibarrier() – calling process entered the barrier, **no** synchronization happens
  - Synchronization **may** happen asynchronously
  - MPI_Test/Wait() – synchronization happens **if** necessary
- Uses:
  - Overlap barrier latency (small benefit)
  - Use the split semantics! Processes **notify** non-collectively but **synchronize** collectively!

A Semantics Example: DSDE

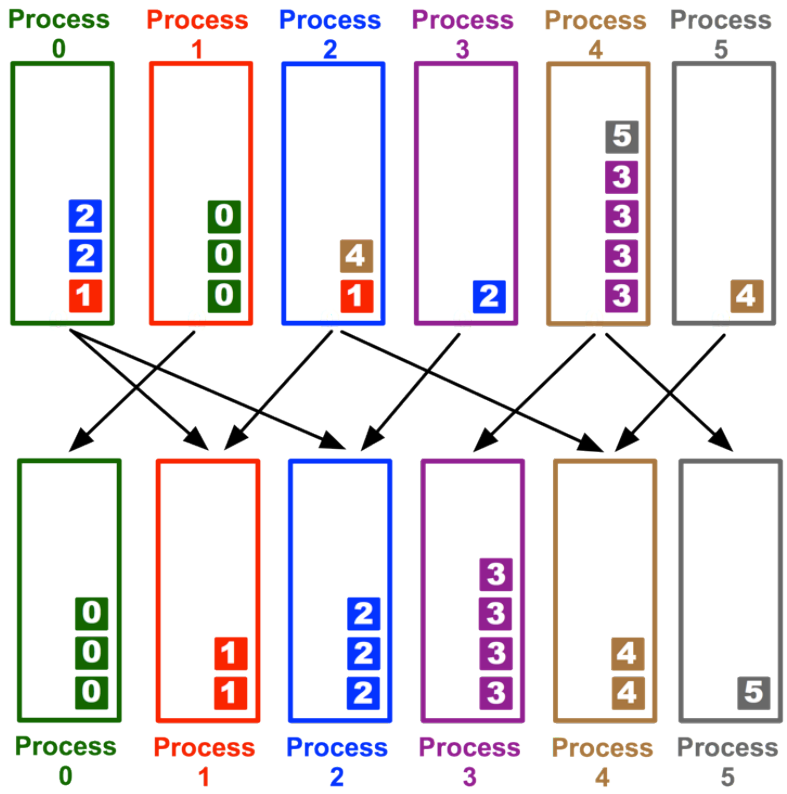
- Dynamic Sparse Data Exchange
  - Dynamic: comm. pattern varies across iterations
  - Sparse: number of neighbors is limited ($\mathcal{O}(\log P)$)
  - Data exchange: only senders know neighbors



Dynamic Sparse Data Exchange (DSDE)

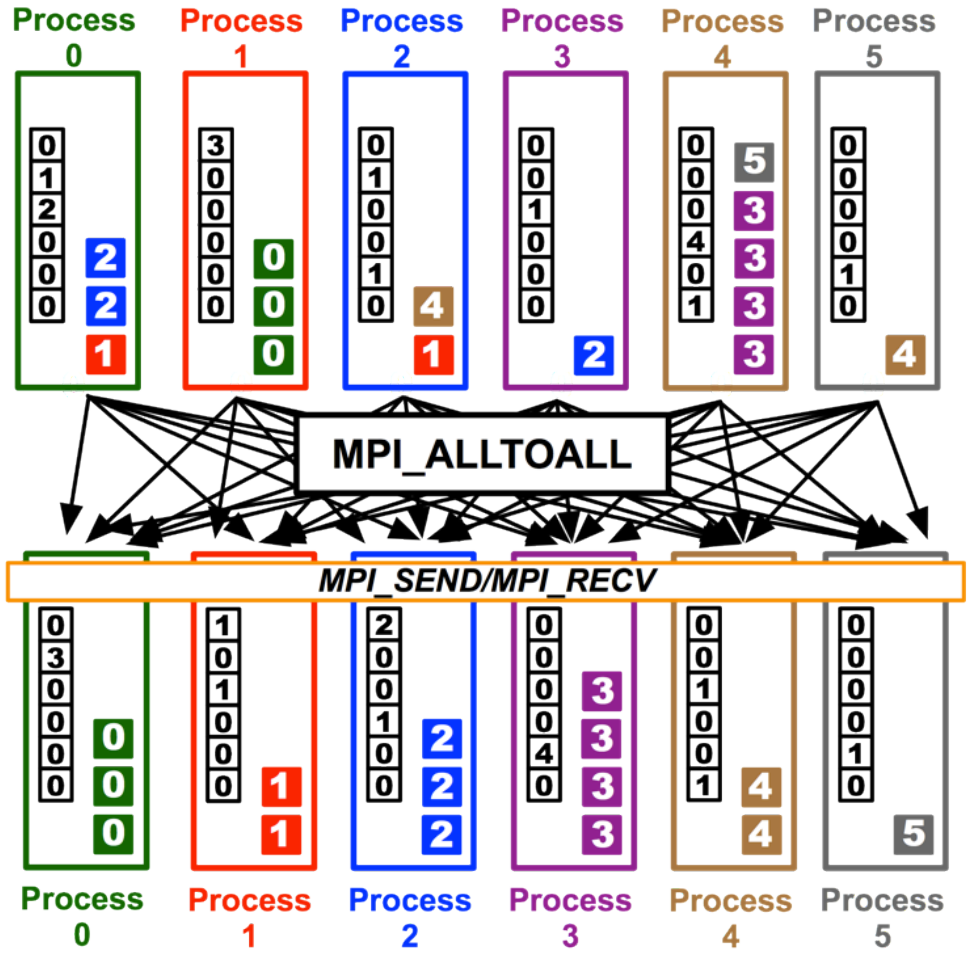
- Main Problem: metadata
  - Determine who wants to send how much data to me (I must post receive and reserve memory)

  OR:
  - Use MPI semantics:
    - Unknown sender
      - MPI_ANY_SOURCE
    - Unknown message size
      - MPI_PROBE
    - Reduces problem to counting the number of neighbors
    - Allow faster implementation!



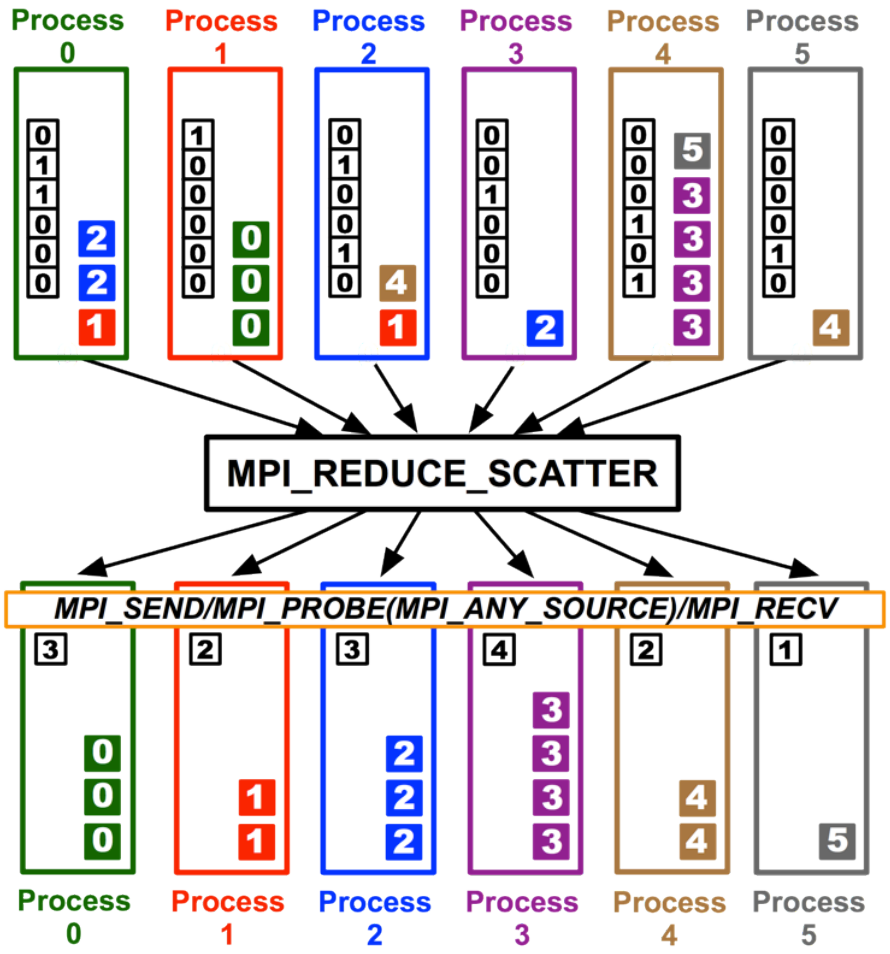
Using Alltoall (PEX)

- Based on Personalized Exchange ($\Theta(P)$)
  - Processes exchange metadata (sizes) about neighborhoods with all-to-all
  - Processes post receives afterwards
  - Most intuitive but least performance and scalability!



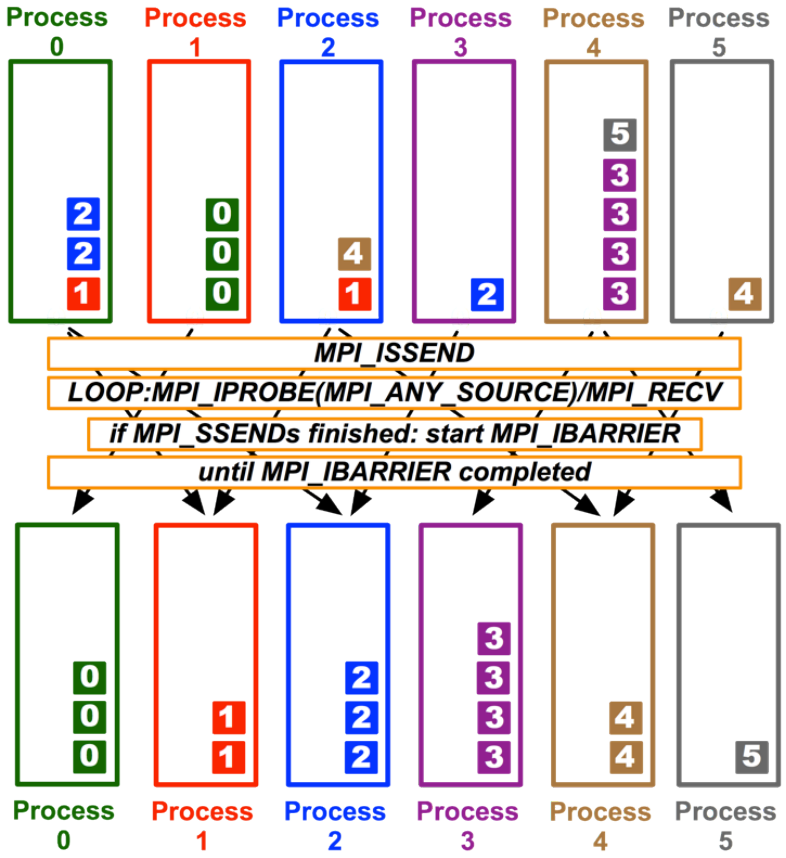
Reduce_scatter (PCX)

- Bases on Personalized Census ($\Theta(P)$)
  - Processes exchange metadata (counts) about neighborhoods with reduce_scatter
  - Receivers checks with wildcard MPI_IPROBE and receives messages
  - Better than PEX but non-deterministic!



MPI_Ibarrier (NBX)

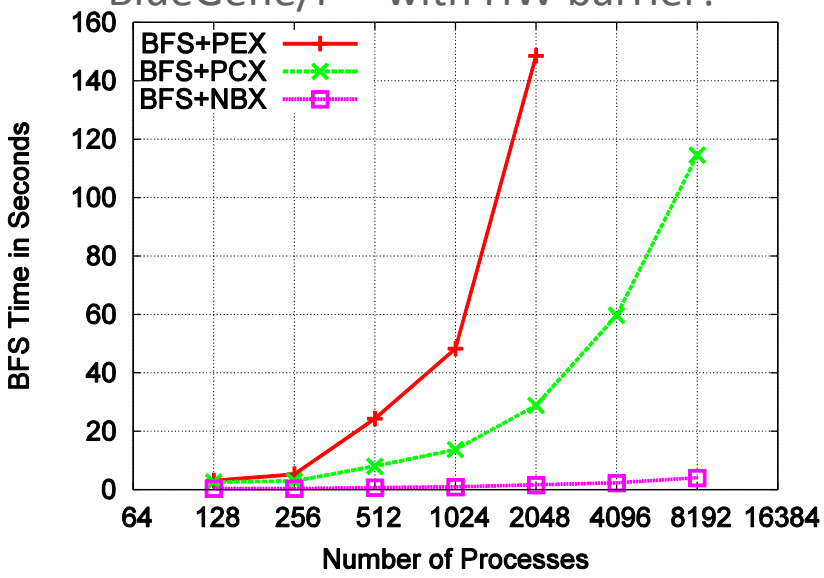
- Complexity - census (barrier): ($\Theta(\log(P))$)
  - Combines metadata with actual transmission
  - Point-to-point synchronization
  - Continue receiving until barrier completes
  - Processes start coll. synch. (barrier) when p2p phase ended
    - barrier = distributed marker!
  - Better than PEX, PCX, RSX!



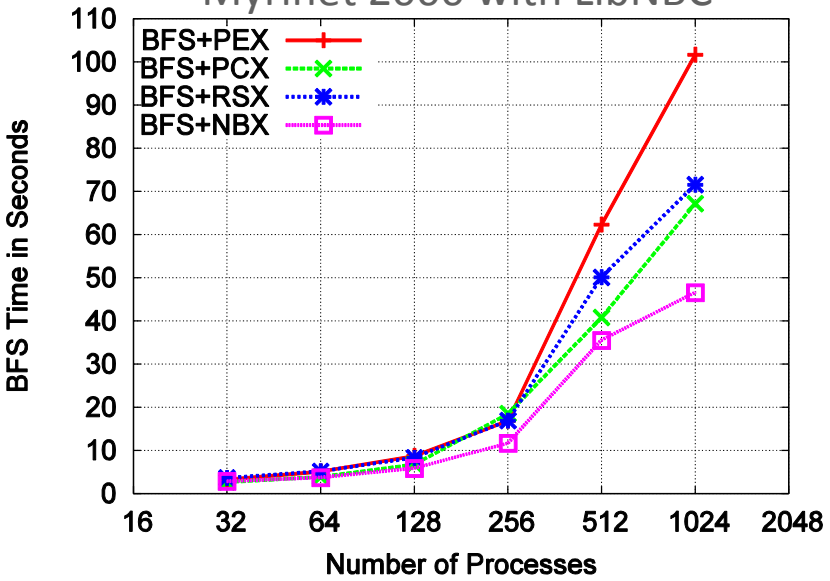
Parallel Breadth First Search

- On a clustered Erdős-Rényi graph, weak scaling
  - 6.75 million edges per node (filled 1 GiB)

BlueGene/P – with HW barrier!



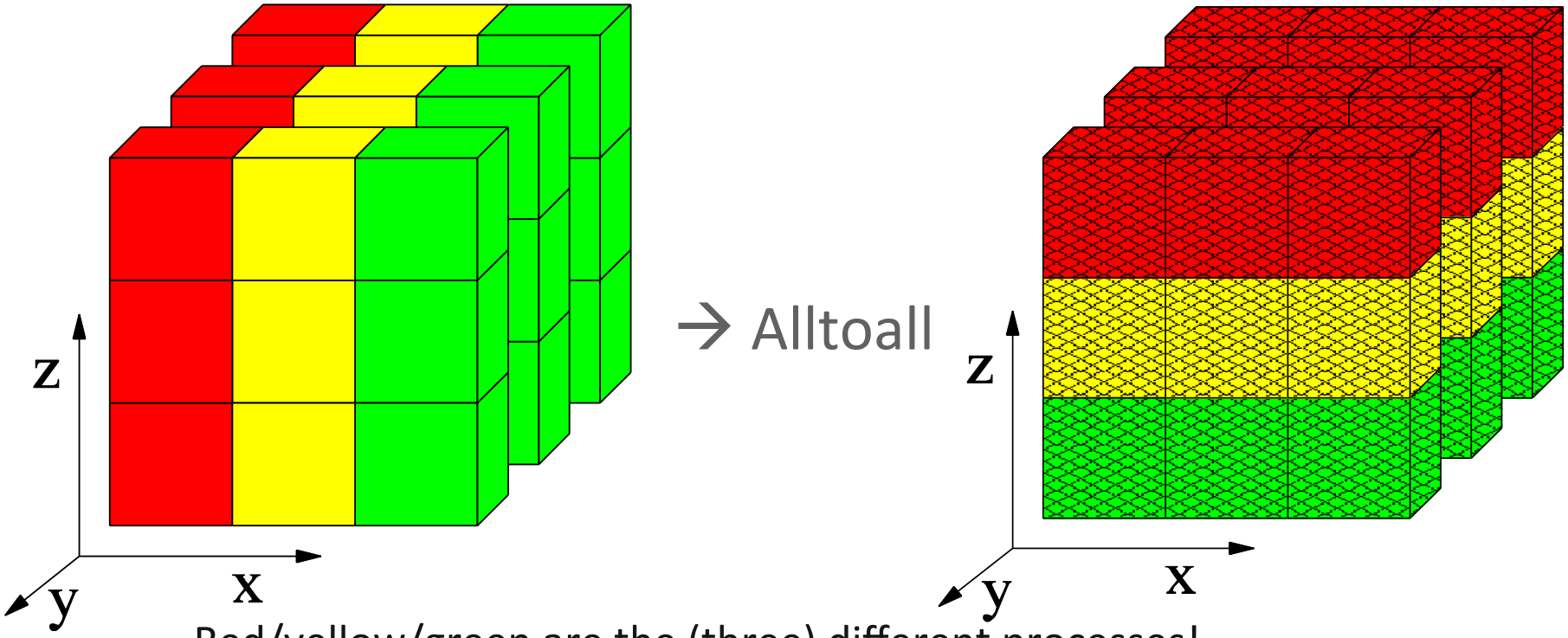
Myrinet 2000 with LibNBC



- HW barrier support is significant at large scale!

Parallel Fast Fourier Transform

- 1D FFTs in all three dimensions
  - Assume 1D decomposition (each process holds a set of planes)
  - Best way: call optimized 1D FFTs in parallel → alltoall



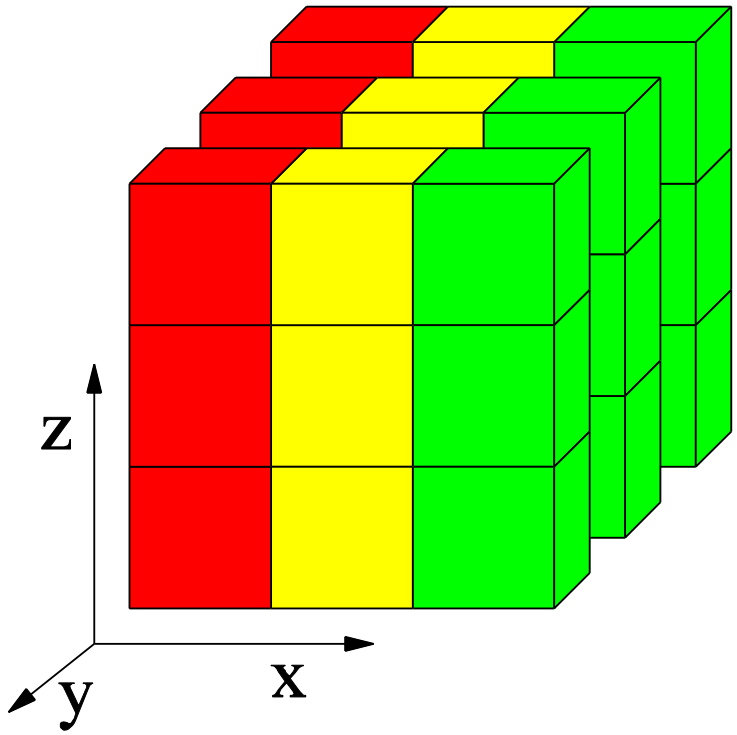
  - Red/yellow/green are the (three) different processes!

A Complex Example: FFT

```
for(int x=0; x<n/p; ++x) 1d_fft(/* x-th stencil */);

// pack data for alltoall
MPI_Alltoall(&in, n/p*n/p, cplx_t, &out, n/p*n/p, cplx_t, comm);
// unpack data from alltoall and transpose

for(int y=0; y<n/p; ++y) 1d_fft(/* y-th stencil */);

// pack data for alltoall
MPI_Alltoall(&in, n/p*n/p, cplx_t, &out, n/p*n/p, cplx_t, comm);
// unpack data from alltoall and transpose
```

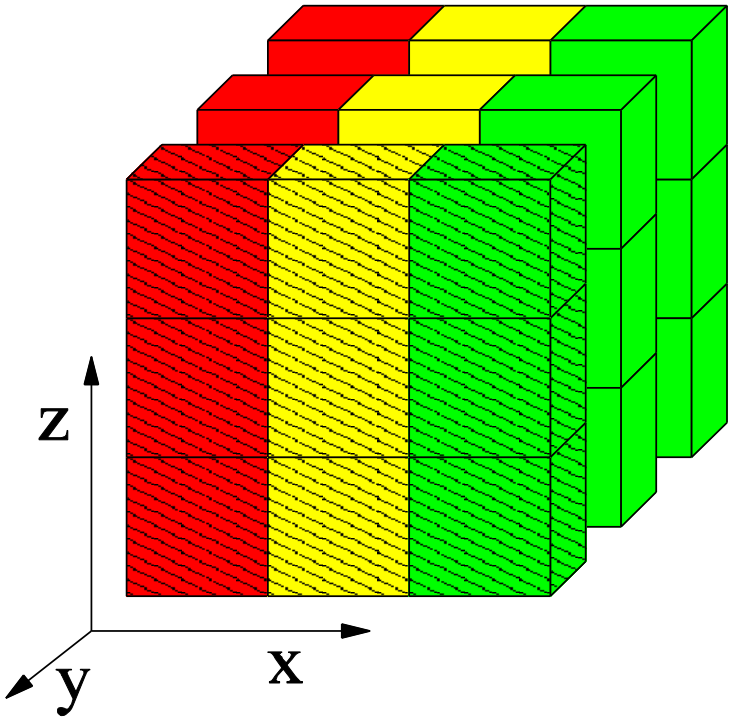
Parallel Fast Fourier Transform

- Data already transformed in y-direction



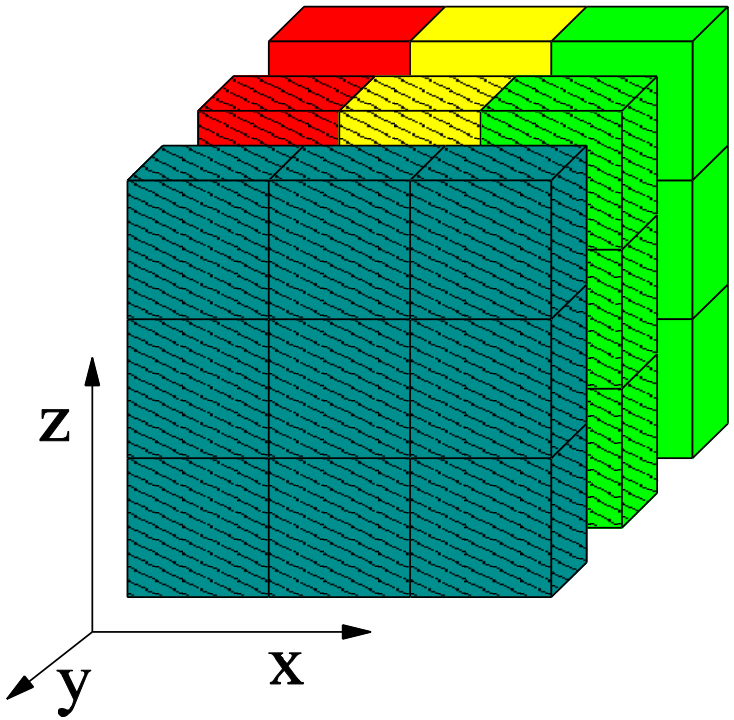
Parallel Fast Fourier Transform

- Transform first y plane in z



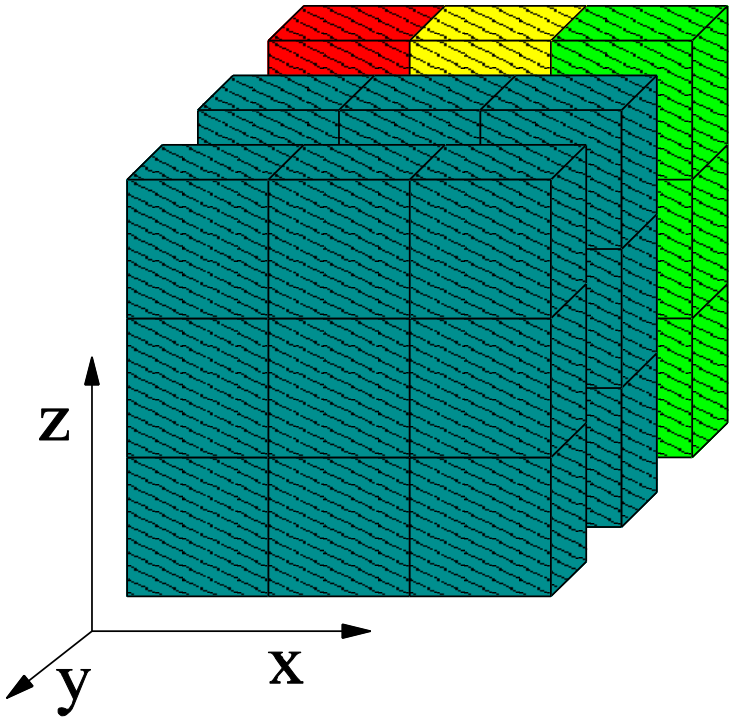
Parallel Fast Fourier Transform

- Start ialltoall and transform second plane



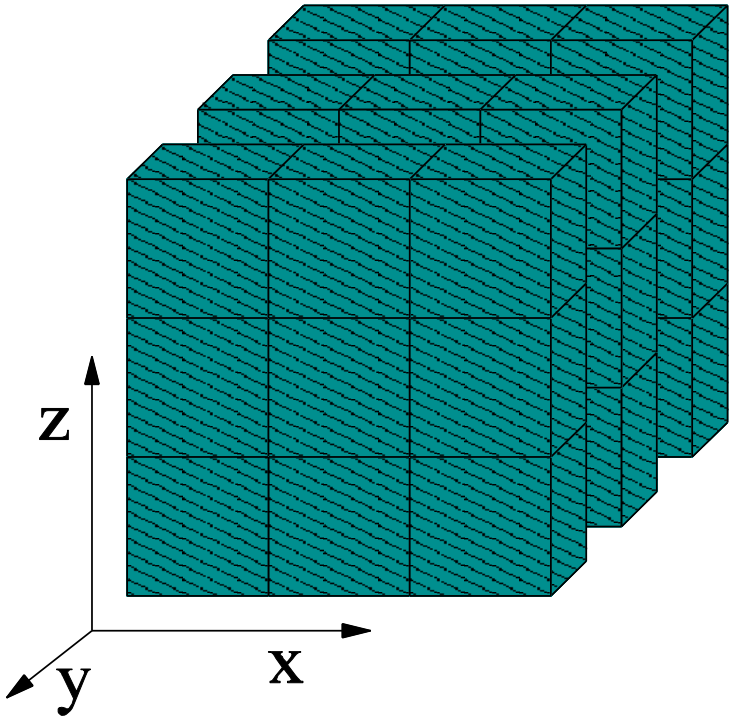
Parallel Fast Fourier Transform

- Start ialltoall (second plane) and transform third



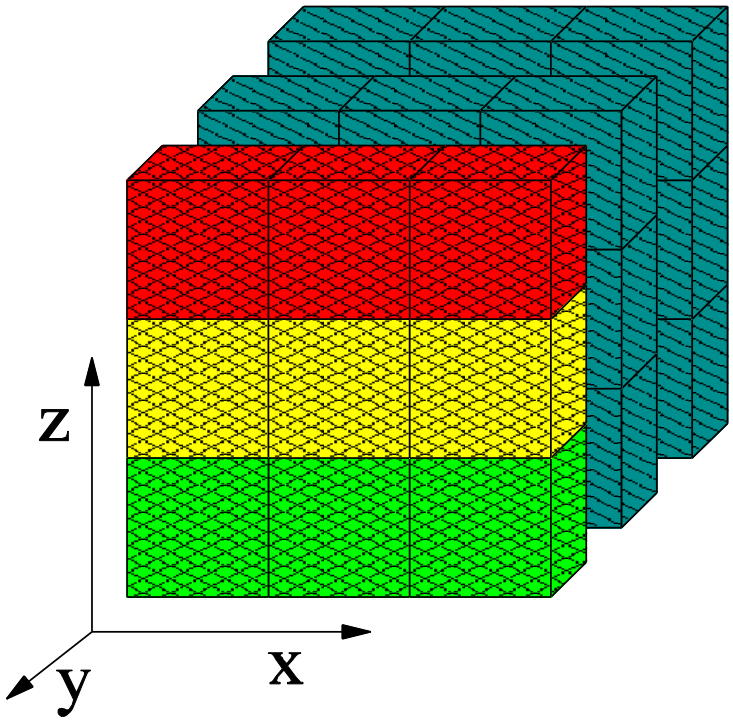
Parallel Fast Fourier Transform

- Start ialltoall of third plane and …



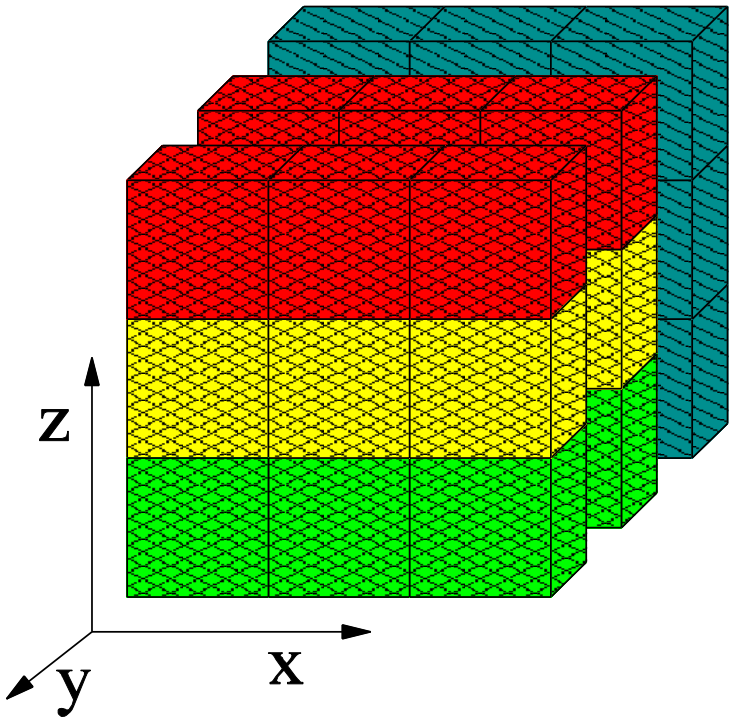
Parallel Fast Fourier Transform

- Finish ialltoall of first plane, start x transform



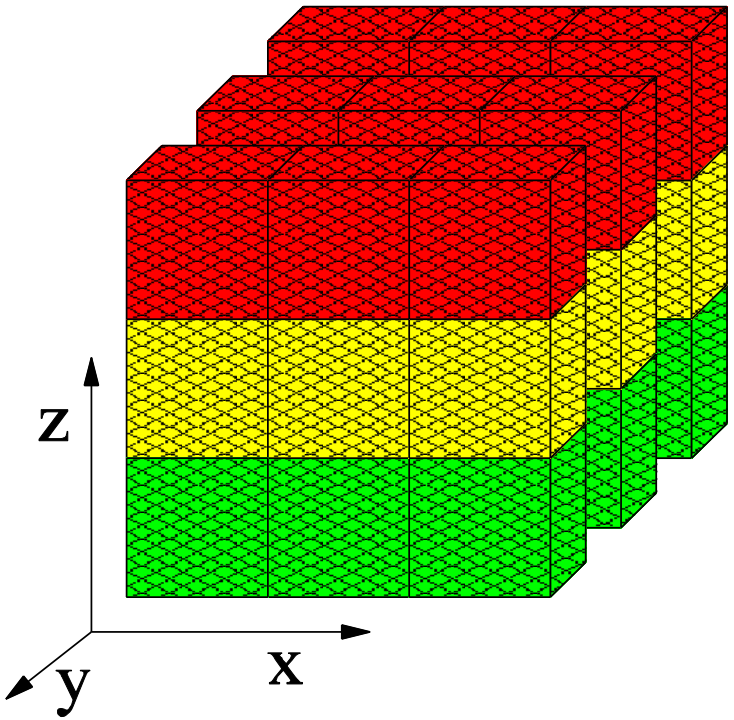
Parallel Fast Fourier Transform

- Finish second ialltoall, transform second plane



Parallel Fast Fourier Transform

- Transform last plane → done



FFT Software Pipelining

```
MPI_Request req[nb];
for(int b=0; b<nb; ++b) { // loop over blocks
  for(int x=b*n/p/nb; x<(b+1)n/p/nb; ++x) 1d_fft(/* x-th stencil*/);

  // pack b-th block of data for alltoall
  MPI_Ialltoall(&in, n/p*n/p/bs, cplx_t, &out, n/p*n/p, cplx_t, comm, &req[b]);
}
MPI_Waitall(nb, req, MPI_STATUSES_IGNORE);

// modified unpack data from alltoall and transpose
for(int y=0; y<n/p; ++y) 1d_fft(/* y-th stencil */);
// pack data for alltoall
MPI_Alltoall(&in, n/p*n/p, cplx_t, &out, n/p*n/p, cplx_t, comm);
// unpack data from alltoall and transpose
```

Nonblocking And Collective Summary

- Nonblocking comm does two things:
  - Overlap and relax synchronization
- Collective comm does one thing
  - Specialized pre-optimized routines
  - Performance portability
  - Hopefully transparent performance
- They can be composed
  - E.g., software pipelining

Topologies and Topology Mapping

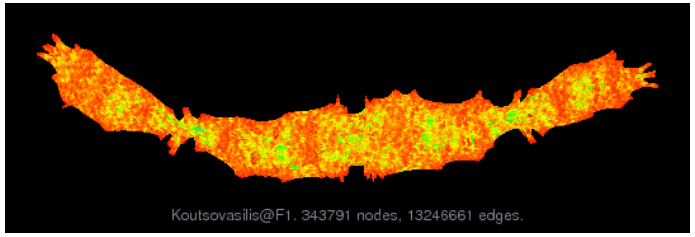
Topology Mapping and Neighborhood Collectives

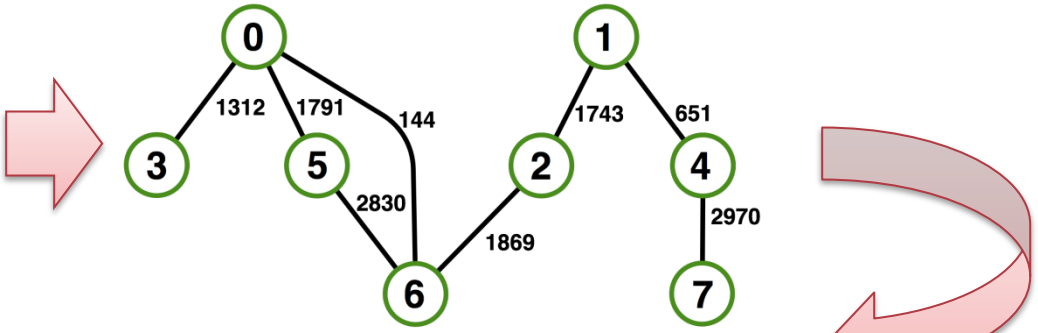
- Topology mapping basics
  - Allocation mapping vs. rank reordering
  - Ad-hoc solutions vs. portability
- MPI topologies
  - Cartesian
  - Distributed graph
- Collectives on topologies – neighborhood collectives
  - Use-cases

Topology Mapping Basics

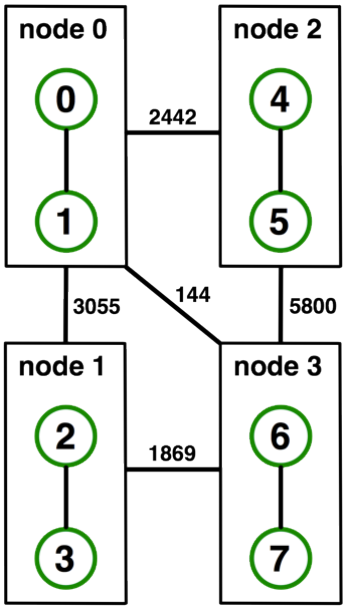
- MPI supports rank reordering
  - Change numbering in a given allocation to reduce congestion or dilation
  - Sometimes automatic (early IBM SP machines)
- Properties
  - Always possible, but effect may be limited (e.g., in a bad allocation)
  - Portable way: MPI process topologies
    - Network topology is not exposed
  - Manual data shuffling after remapping step

Example: On-Node Reordering



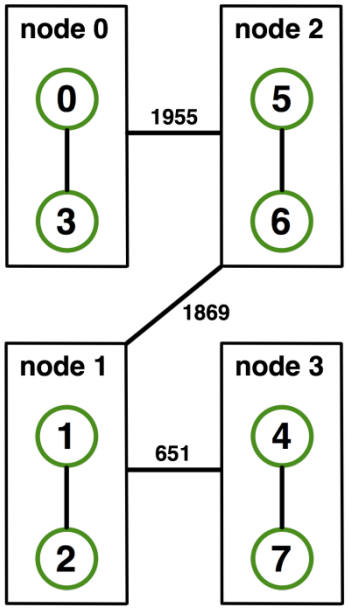


Naïve Mapping



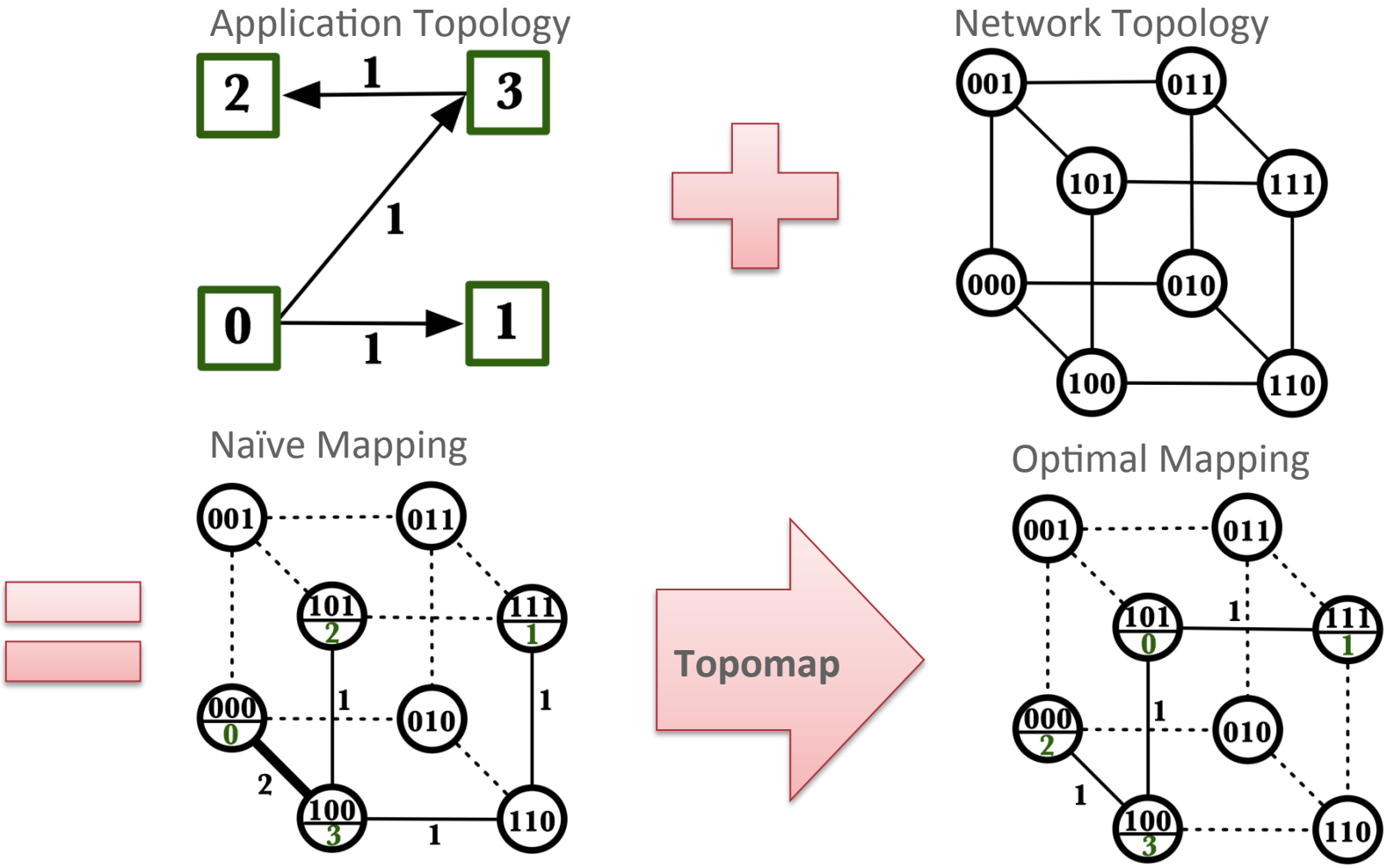
Topomap

Optimized Mapping



Gottschling et al.: Productive Parallel Linear Algebra Programming with Unstructured Topology Adaption

Off-Node (Network) Reordering



MPI Topology Intro

- Convenience functions (in MPI-1)
  - Create a graph and query it, nothing else
  - Useful especially for Cartesian topologies
    - Query neighbors in n-dimensional space
  - Graph topology: each rank specifies full graph ☹
- Scalable Graph topology (MPI-2.2)
  - Graph topology: each rank specifies its neighbors **or** an arbitrary subset of the graph
- Neighborhood collectives (MPI-3.0)
  - Adding communication functions defined on graph topologies (neighborhood of distance one)

MPI_Cart_create

```
MPI_Cart_create(MPI_Comm comm_old, int ndims, const int *dims,
                const int *periods, int reorder, MPI_Comm *comm_cart)
```

- Specify ndims-dimensional topology
  - Optionally periodic in each dimension (Torus)
- Some processes may return MPI_COMM_NULL
  - Product sum of dims must be <= P
- Reorder argument allows for topology mapping
  - Each calling process may have a new rank in the created communicator
  - Data has to be remapped manually

MPI_Cart_create Example

```
int dims[3] = {5,5,5};
int periods[3] = {1,1,1};
MPI_Comm topocomm;
MPI_Cart_create(comm, 3, dims, periods, 0, &topocomm);
```

- Creates logical 3-d Torus of size 5x5x5
- But we're starting MPI processes with a one-dimensional argument (-p X)
  - User has to determine size of each dimension
  - Often as "square" as possible, MPI can help!

MPI_Dims_create

MPI_Dims_create(int nnodes, int ndims, int *dims)

- Create dims array for Cart_create with nnodes and ndims
  - Dimensions are as close as possible (well, in theory)
- Non-zero entries in dims will not be changed
  - nnodes must be multiple of all non-zeroes

MPI_Dims_create Example

```
int p;
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Dims_create(p, 3, dims);

int periods[3] = {1,1,1};
MPI_Comm topocomm;
MPI_Cart_create(comm, 3, dims, periods, 0, &topocomm);
```

- Makes life a little bit easier
  - Some problems may be better with a non-square layout though

Cartesian Query Functions

- Library support and convenience!
- MPI_Cartdim_get()
  - Gets dimensions of a Cartesian communicator
- MPI_Cart_get()
  - Gets size of dimensions
- MPI_Cart_rank()
  - Translate coordinates to rank
- MPI_Cart_coords()
  - Translate rank to coordinates

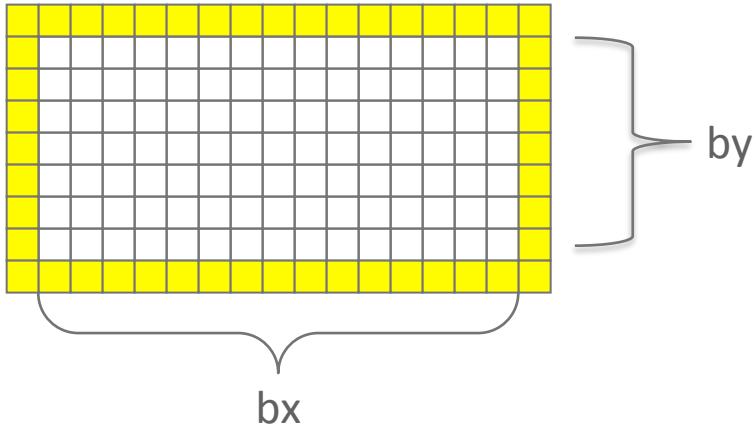
Cartesian Communication Helpers

```
MPI_Cart_shift(MPI_Comm comm, int direction, int disp,
               int *rank_source, int *rank_dest)
```

- Shift in one dimension
  - Dimensions are numbered from 0 to ndims-1
  - Displacement indicates neighbor distance (-1, 1, …)
  - May return MPI_PROC_NULL
- Very convenient, all you need for nearest neighbor communication
  - No “over the edge” though

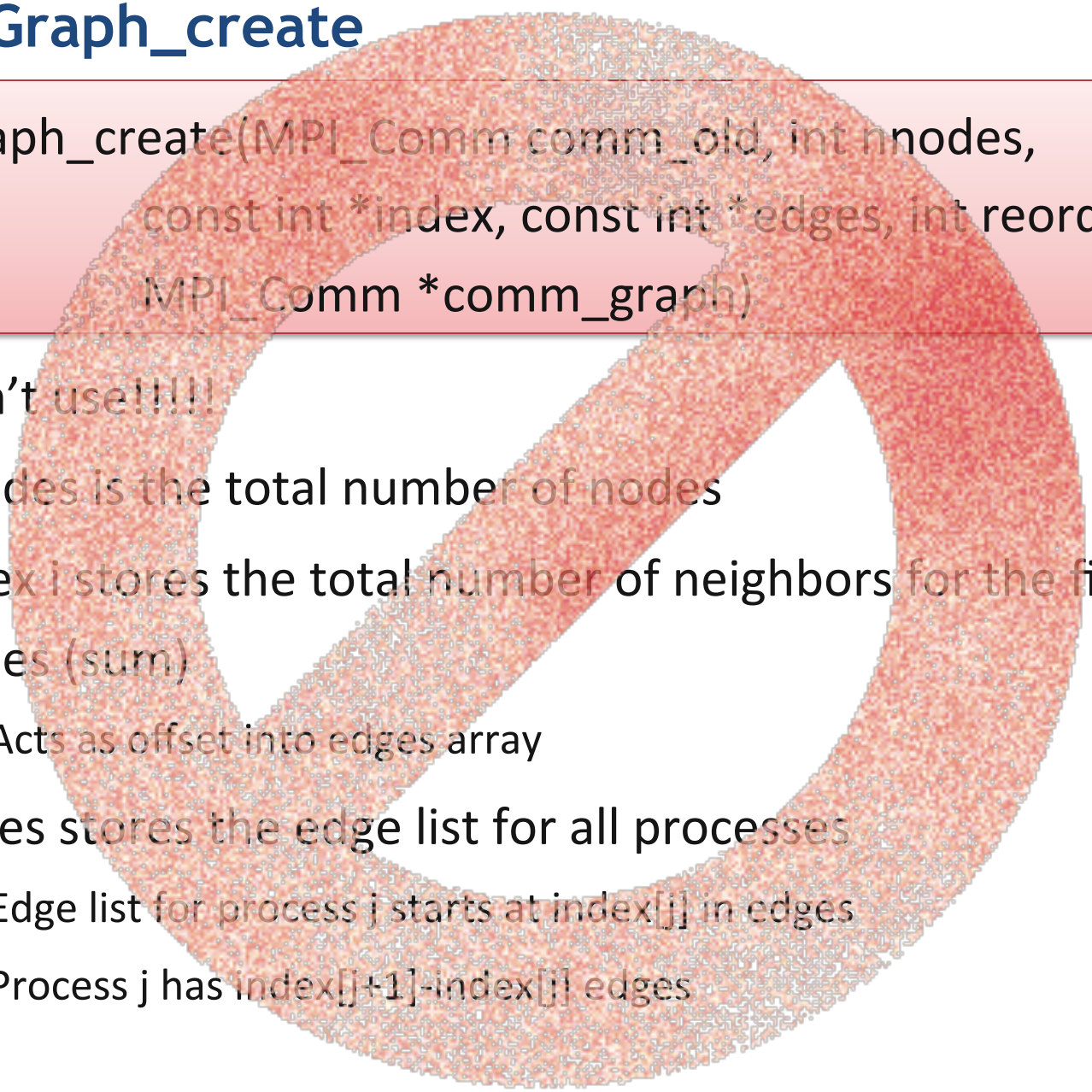
Code Example

- *stencil-mpi-carttopo.c*
- Adds calculation of neighbors with topology



MPI_Graph_create

```
MPI_Graph_create(MPI_Comm comm_old, int nnodes,
                 const int *index, const int *edges, int reorder,
                 MPI_Comm *comm_graph)
```



- Don't use!!!!!
- nnodes is the total number of nodes
- index i stores the total number of neighbors for the first i nodes (sum)
  - Acts as offset into edges array
- edges stores the edge list for all processes
  - Edge list for process j starts at index[j] in edges
  - Process j has index[j+1]-index[j] edges

Distributed graph constructor

- MPI_Graph_create is discouraged
  - Not scalable
  - Not deprecated yet but hopefully soon
- New distributed interface:
  - Scalable, allows distributed graph specification
    - Either local neighbors **or** any edge in the graph
  - Specify edge weights
    - Meaning undefined but optimization opportunity for vendors!
  - Info arguments
    - Communicate assertions of semantics to the MPI library
    - E.g., semantics of edge weights

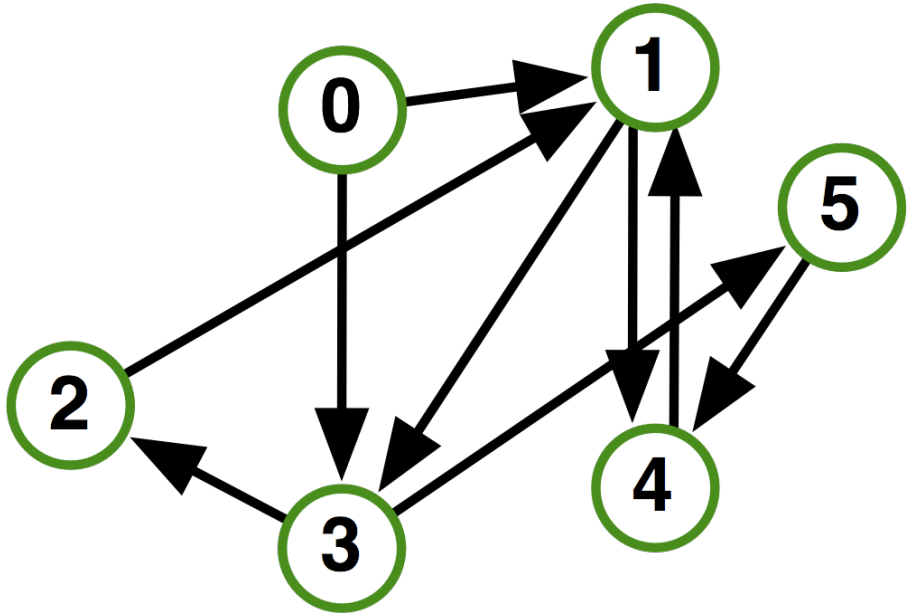
MPI_Dist_graph_create_adjacent

```
MPI_Dist_graph_create_adjacent(MPI_Comm comm_old,
        int indegree, const int sources[], const int sourceweights[],
        int outdegree, const int destinations[],
        const int destweights[], MPI_Info info, int reorder,
        MPI_Comm *comm_dist_graph)
```

- indegree, sources, ~weights – source proc. Spec.
- outdegree, destinations, ~weights – dest. proc. spec.
- info, reorder, comm_dist_graph – as usual
- directed graph
- Each edge is specified twice, once as out-edge (at the source) and once as in-edge (at the dest)

MPI_Dist_graph_create_adjacent

- Process 0:
  - Indegree: 0
  - Outdegree: 2
  - Dests: {3,1}
- Process 1:
  - Indegree: 3
  - Outdegree: 2
  - Sources: {4,0,2}
  - Dests: {3,4}
- …



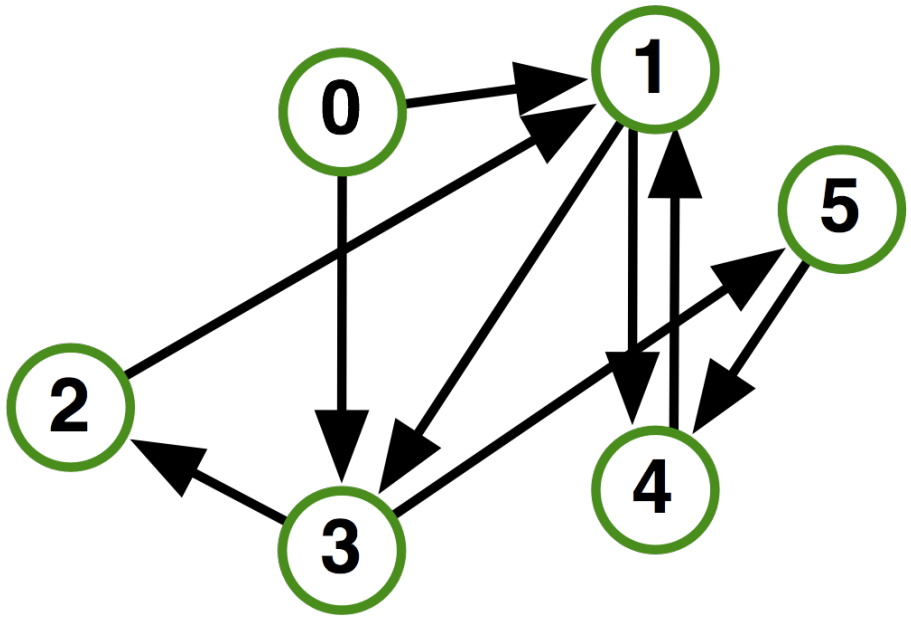
MPI_Dist_graph_create

```
MPI_Dist_graph_create(MPI_Comm comm_old, int n,
        const int sources[], const int degrees[],
        const int destinations[], const int weights[], MPI_Info info,
        int reorder, MPI_Comm *comm_dist_graph)
```

- n – number of source nodes
- sources – n source nodes
- degrees – number of edges for each source
- destinations, weights – dest. processor specification
- info, reorder – as usual
- More flexible and convenient
  - Requires global communication
  - Slightly more expensive than adjacent specification

MPI_Dist_graph_create

- Process 0:
  - N: 2
  - Sources: {0,1}
  - Degrees: {2,1} *
  - Dests: {3,1,4}
- Process 1:
  - N: 2
  - Sources: {2,3}
  - Degrees: {1,1}
  - Dests: {1,2}
- …



* Note that in this example, process 0 specifies only one of the two outgoing edges of process 1; the second outgoing edge needs to be specified by another process

Distributed Graph Neighbor Queries

```
MPI_Dist_graph_neighbors_count(MPI_Comm comm,
        int *indegree,int *outdegree, int *weighted)
```

- Query the number of neighbors of **calling process**
- Returns indegree and outdegree!
- Also info if weighted

```
MPI_Dist_graph_neighbors(MPI_Comm comm, int maxindegree,
        int sources[], int sourceweights[], int maxoutdegree,
        int destinations[],int destweights[])
```

- Query the neighbor list of **calling process**
- Optionally return weights

Further Graph Queries

```
MPI_Topo_test(MPI_Comm comm, int *status)
```

- Status is either:
  - MPI_GRAPH (ugs)
  - MPI_CART
  - MPI_DIST_GRAPH
  - MPI_UNDEFINED (no topology)
- Enables to write libraries on top of MPI topologies!

Neighborhood Collectives

Neighborhood Collectives

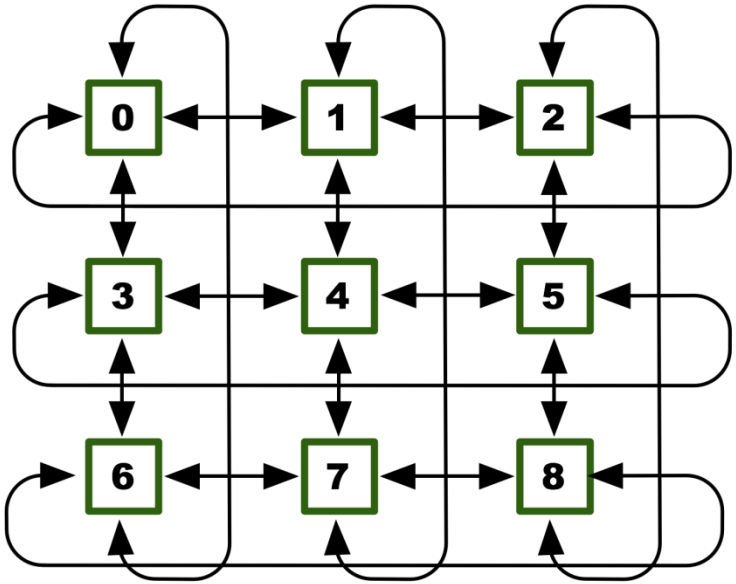
- Topologies implement no communication!
  - Just helper functions
- Collective communications only cover some patterns
  - E.g., no stencil pattern
- Several requests for "build your own collective" functionality in MPI
  - Neighborhood collectives are a simplified version
  - Cf. Datatypes for communication patterns!

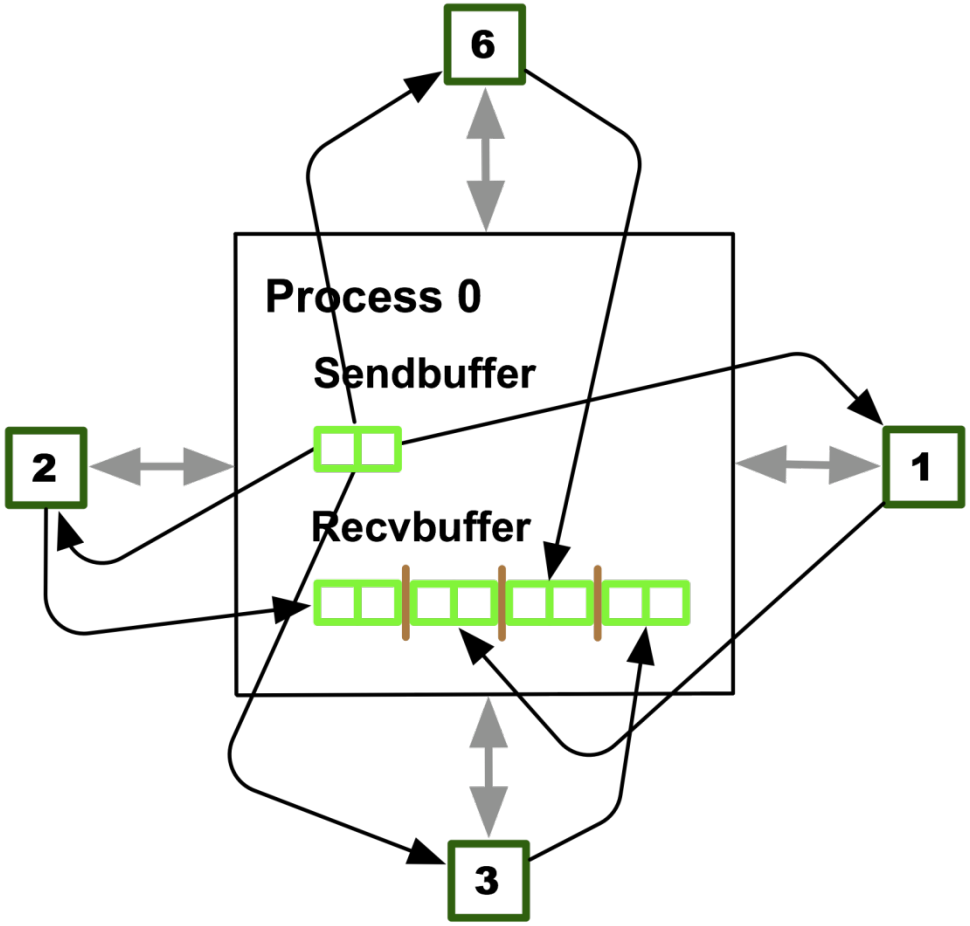
Cartesian Neighborhood Collectives

- Communicate with direct neighbors in Cartesian topology
  - Corresponds to cart_shift with disp=1
  - Collective (all processes in comm must call it, including processes without neighbors)
  - Buffers are laid out as neighbor sequence:
    - Defined by order of dimensions, first negative, then positive
    - 2*ndims sources and destinations
    - Processes at borders (MPI_PROC_NULL) leave holes in buffers (will not be updated or communicated)!

Cartesian Neighborhood Collectives

- Buffer ordering example:





Graph Neighborhood Collectives

- Collective Communication along arbitrary neighborhoods
  - Order is determined by order of neighbors as returned by (dist_)graph_neighbors.
  - Distributed graph is directed, may have different numbers of send/recv neighbors
  - Can express dense collective operations ☺
  - Any persistent communication pattern!

MPI_Neighbor_allgather

```
MPI_Neighbor_allgather(const void* sendbuf, int sendcount,
        MPI_Datatype sendtype, void* recvbuf, int recvcount,
        MPI_Datatype recvtype, MPI_Comm comm)
```

- Sends the same message to all neighbors
- Receives indegree distinct messages
- Similar to MPI_Gather
  - The all prefix expresses that each process is a “root” of his neighborhood
- Vector version for full flexibility

MPI_Neighbor_alltoall

```
MPI_Neighbor_alltoall(const void* sendbuf, int sendcount,
        MPI_Datatype sendtype, void* recvbuf, int recvcount,
        MPI_Datatype recvtype, MPI_Comm comm)
```

- Sends outdegree distinct messages
- Received indegree distinct messages
- Similar to MPI_Alltoall
  - Neighborhood specifies full communication relationship
- Vector and w versions for full flexibility

Nonblocking Neighborhood Collectives

```
MPI_Ineighbor_allgather(…, MPI_Request *req);
MPI_Ineighbor_alltoall(…, MPI_Request *req);
```

- Very similar to nonblocking collectives
- Collective invocation
- Matching in-order (no tags)
  - No wild tricks with neighborhoods! In order matching per communicator!

Walkthrough of 2D Stencil Code with Neighborhood Collectives

- Code can be downloaded from

  www.mcs.anl.gov/~thakur/sc14-mpi-tutorial

Why is Neighborhood Reduce Missing?

MPI_Ineighbor_allreducev(…);

- Was originally proposed (see original paper)
- High optimization opportunities
  - Interesting tradeoffs!
  - Research topic
- Not standardized due to missing use-cases
  - My team is working on an implementation
  - Offering the obvious interface

Topology Summary

- Topology functions allow to specify application communication patterns/topology
  - Convenience functions (e.g., Cartesian)
  - Storing neighborhood relations (Graph)
- Enables topology mapping (reorder=1)
  - Not widely implemented yet
  - May requires manual data re-distribution (according to new rank order)
- MPI does not expose information about the network topology (would be very complex)

Neighborhood Collectives Summary

- Neighborhood collectives add communication functions to process topologies
  - Collective optimization potential!
- Allgather
  - One item to all neighbors
- Alltoall
  - Personalized item to each neighbor
- High optimization potential (similar to collective operations)
  - Interface encourages use of topology mapping!

Section Summary

- Process topologies enable:
  - High-abstraction to specify communication pattern
  - Has to be relatively static (temporal locality)
    - Creation is expensive (collective)
  - Offers basic communication functions
- Library can optimize:
  - Communication schedule for neighborhood colls
  - Topology mapping

Recent Efforts of the MPI Forum for MPI-3.1, MPI-4, and Future MPI Standards

Introduction

- The MPI Forum continues to meet once every 3 months to define future versions of the MPI Standard
  - The next Forum meeting is December 8-11, 2014, in San Jose
- We describe some of the proposals the Forum is currently considering

Improved Support for Fault Tolerance

- MPI always had support for error handlers and allows implementations to return an error code and remain alive
- MPI Forum working on additional support for MPI-4
- Current proposal handles fail-stop process failures (not silent data corruption or Byzantine failures)
  - If a communication operation fails because the other process has failed, the function returns error code MPI_ERR_PROC_FAILED
  - User can call MPI_Comm_shrink to create a new communicator that excludes failed processes
  - Collective communication can be performed on the new communicator
  - Lots of other details in the proposal…

Better Hybrid Programming: Extending MPI to Support Multiple Endpoints Per Process

- In MPI today, each process has a single communication endpoint (rank in MPI_COMM_WORLD)
- Multiple threads of a process communicate through that single endpoint, requiring the implementation to use locks etc., which are expensive
- MPI Forum is discussing a proposal (for MPI-4) that allows a process to have multiple endpoints
- Threads within a process can attach to different endpoints and communicate through those endpoints as if they are separate ranks
- The MPI implementation can avoid using locks if each thread communicates on a separate endpoint
- This allows the MPI standard to support "MPI + X" more efficiently without specifying what X is

Other concepts being considered

- MPI Streams interface
  - Streaming data between sender and receiver
- Nonblocking File Manipulation routines
  - Nonblocking versions of file open, close, set_view, etc.
- Active Messages
  - Initiate operations on remote processes
  - Possibly as an addition to MPI RMA
- Tools Interface
  - Scalable process acquisition interface
  - Introspection of MPI handles

Argonne NATIONAL LABORATORY

Concluding Remarks

ETH Zürich

Conclusions

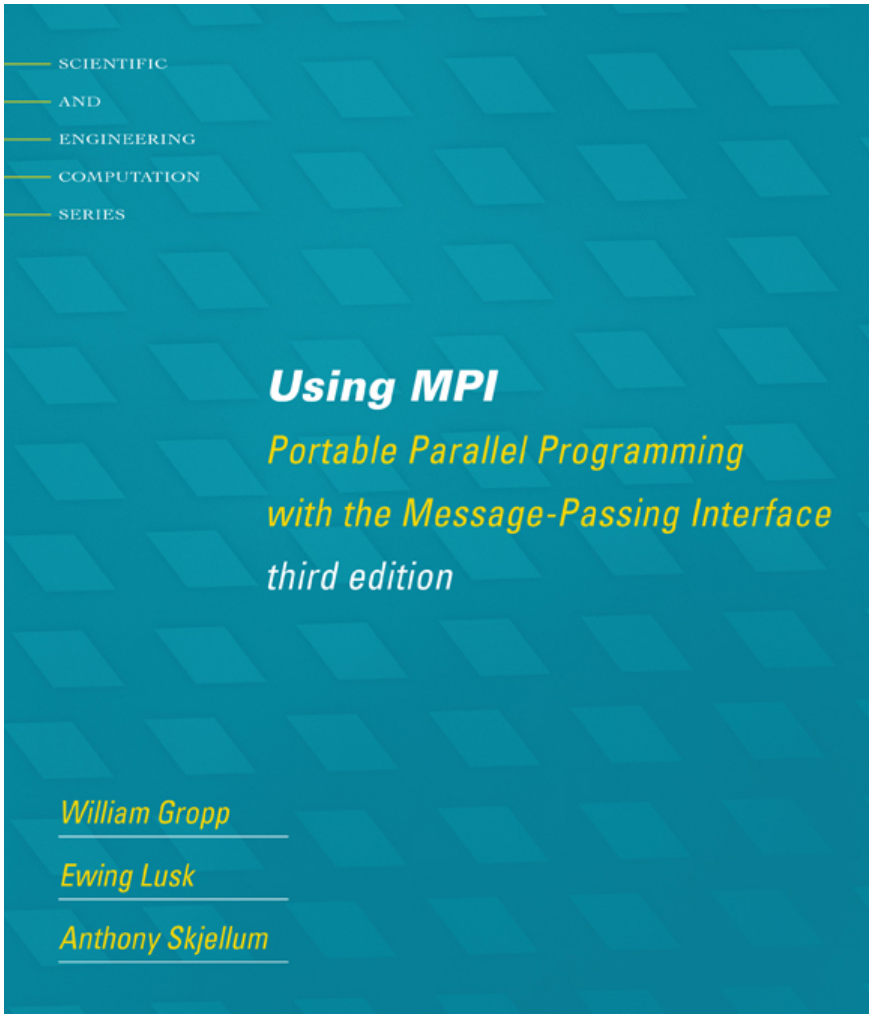
- Parallelism is critical today, given that it is the only way to achieve performance improvement with modern hardware
- MPI is an industry standard model for parallel programming
  - A large number of implementations of MPI exist (both commercial and public domain)
  - Virtually every system in the world supports MPI
- Gives user explicit control on data management
- Widely used by many scientific applications with great success
- Your application can be next!

Web Pointers

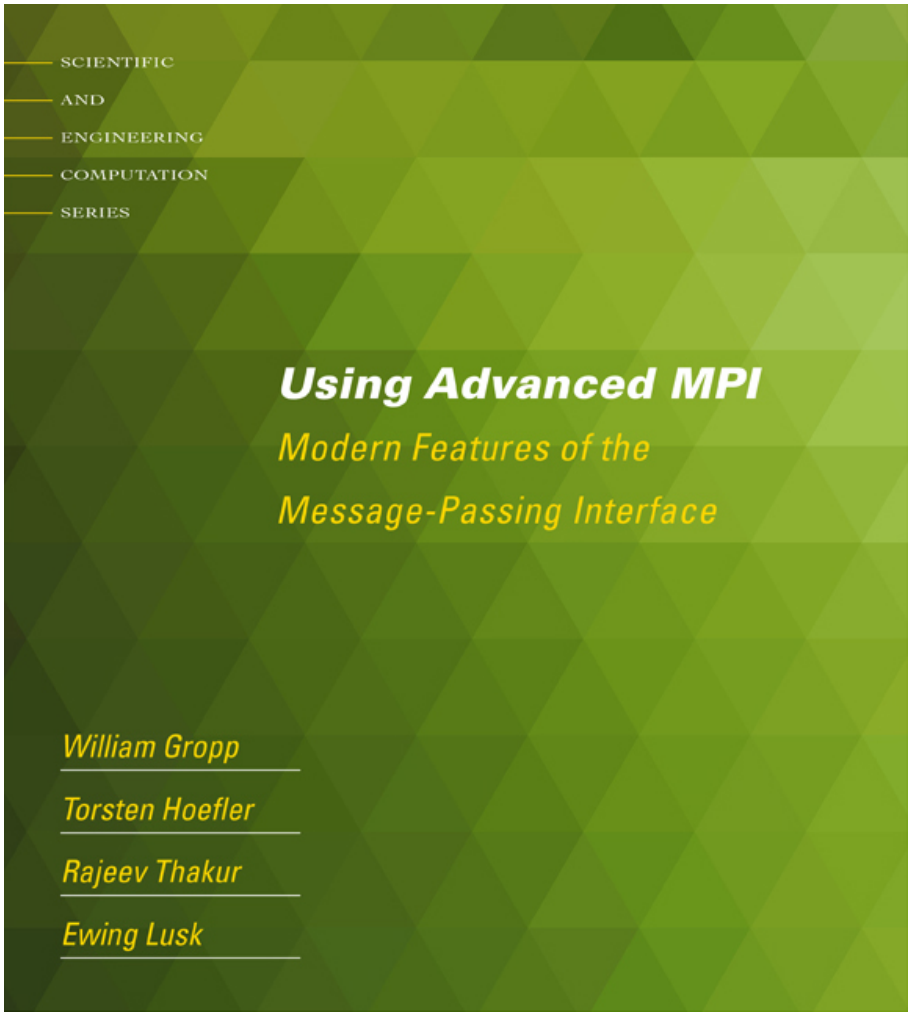
- MPI standard : http://www.mpi-forum.org/docs/docs.html
- MPI Forum : http://www.mpi-forum.org/

- MPI implementations:
  - MPICH : http://www.mpich.org
  - MVAPICH : http://mvapich.cse.ohio-state.edu/
  - Intel MPI: http://software.intel.com/en-us/intel-mpi-library/
  - Microsoft MPI: www.microsoft.com/en-us/download/details.aspx?id=39961
  - Open MPI : http://www.open-mpi.org/
  - IBM MPI, Cray MPI, HP MPI, TH MPI, …
- Several MPI tutorials can be found on the web

New Tutorial Books on MPI



Basic MPI



Advanced MPI, including MPI-3