# Collective Operations Workgroup

Torsten Hoefler
with edits from J.L. Traeff

Open Systems Lab
Indiana University
Bloomington, USA

$10^{th}$ MPI Forum Meeting June'09

San Jose, CA, USA

Jun, 8-10th 2009

# Overview

1. Nonblocking Collectives (nothing to say?)
2. Sparse Collectives
3. Topology Modifiers
4. Persistent Collectives
5. Other Items?

Any Comments/Discussions?

# Sparse Collectives

- application research (*"Sparse Non-Blocking Collectives in Quantum Mechanical Calculations"* in EuroPVM/MPI'08)
- $\longrightarrow$ Performance, Usability
- we did implementation research seven months ago (see wiki - *"Sparse Collective Operations for MPI"* in HIPS'09)
- $\longrightarrow$ Interface, Implementation options
- $\Rightarrow$ more research needed but benefits are shown
- we propose a complete interface
- more on the next slides

# Sparse Collectives

## Source

the following is based on: *"Sparse Collective Operations for MPI"* in HIPS'09

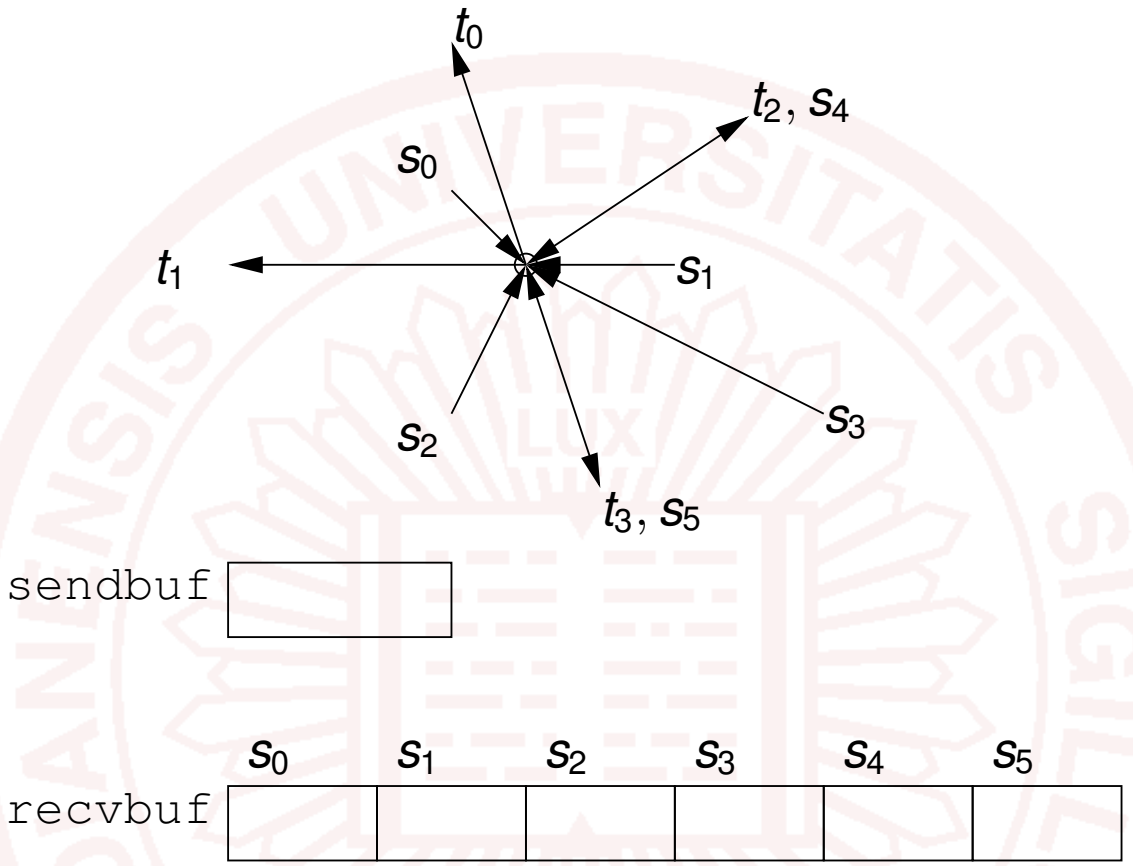
## Operations

- three operations: Gather(v), Alltoall(v), Reduce(v)
- also nonblocking variants (of course)

## Neighborhoods

- concept of (send- and receive-) neighborhoods
- each process sends data to target neighbors and receives data from source neighbors

# Gather




## Interface

```
MPI_Neighbor_gather(sendbuf,sendcount,sendtype,
                    recvbuf,recvcount,recvtype,
                    comm)
```

# Gather continued

## Buffer Semantics

- data is stored continuously in buffers
- usual MPI semantics (address, count, datatype)
- single-item sendbuf
- $k$-item recvbuf ($k$ sources)
- data size of the $k$-th recvbuf must match the send-size of the $k$-th source

# Alltoall

## Interface

```
MPI_Neighbor_alltoall(sendbuf,sendcount,sendtype,
                      recvbuf,recvcount,recvtype,
                      comm)
```

## Semantics

- like gather, but each target receives personalized data
- sendbuf holds *k* elements (*k* targets)

# Reduce

## Interface

```
MPI_Neighbor_reduce(sendbuf,sendcount,sendtype,
                    recvbuf,recvcount,recvtype,
                    op,comm)
```

## Semantics

- processes received reduced data from all sources into single buffer
- normal MPI reduction rules apply
- each process contributes the *same* data block to all targets
- this can not be expressed in MPI easily
- $\rightarrow$ all processes in a connected component must have same datasize

# Vector Variants

```
MPI_Neighbor_gatherv(sendbuf,sendcount,sendtype,
                     recvbuf,
                     recvcounts,recvdispls,recvtype,
                     comm)
MPI_Neighbor_alltoallv(sendbuf,
                       sendcounts,senddispls,sendtype,
                       recvbuf,
                       recvcounts,recvdispls,recvtype,
                       comm)
MPI_Neighbor_alltoallw(sendbuf,
                       sendcounts,senddispls,sendtypes,
                       recvbuf,
                       recvcounts,recvdispls,recvtypes,
                       comm)
MPI_Neighbor_reducev(sendbuf,
                     sendcounts,senddispls,sendtype,
                     recvbuf,recvcount,recvtype,
                     op,comm)
```

# General Semantics

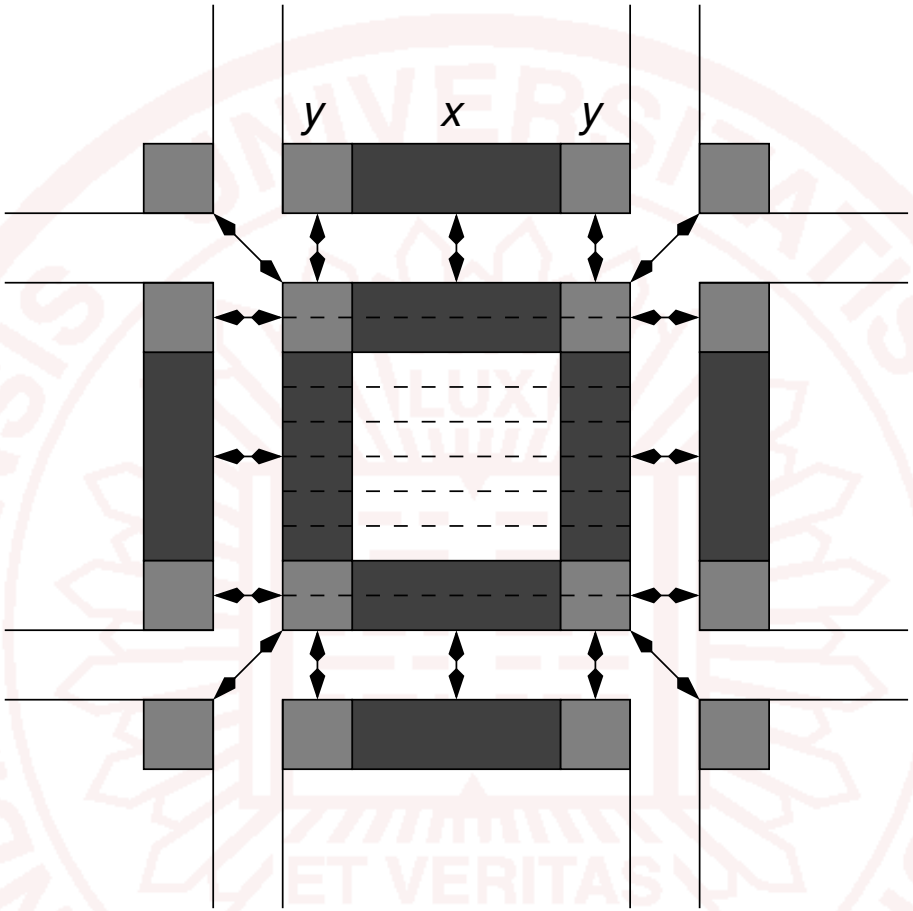
## Correctness

- source/target lists must match
  (e.g., $i$ is target of $j \Leftrightarrow j$ is source of $i$)
- should such calls be collective over the whole communicator or only involved processes?
  - provides more flexibility (limiting collectiveness to connected components)
  - routing through “not involved” processes is not possible

## Nonblocking variants

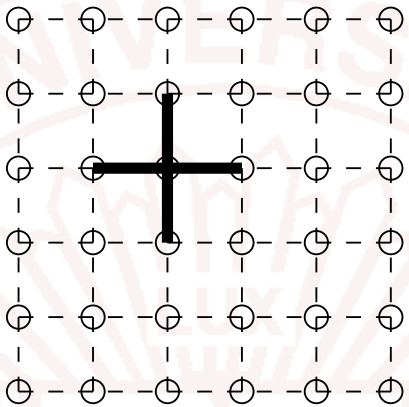
- straight forward, similar to NBC
- add “I” and MPI_Request to each call

# A Halo Example



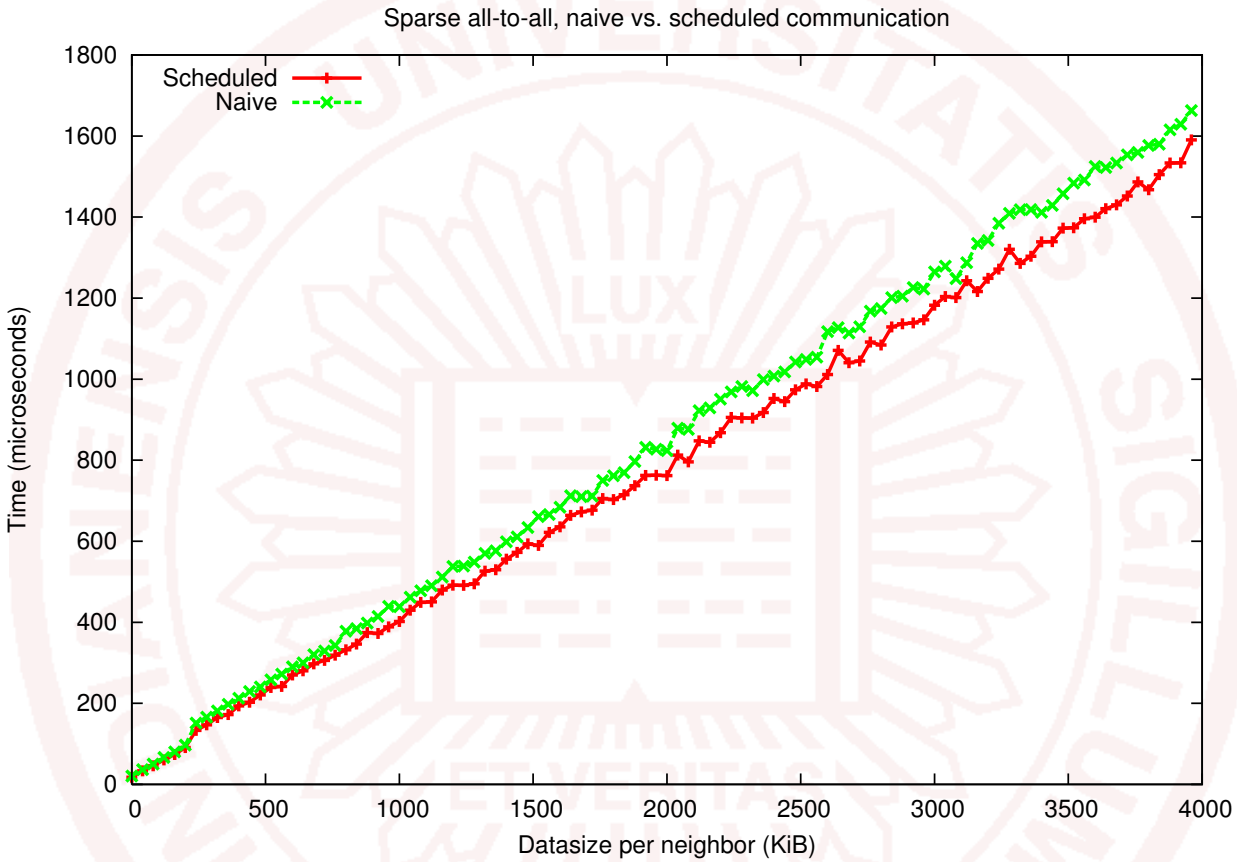

- needs overlapping Neighbor_alltoallw
- common optimization (piggy backing) for diagonal communications can be performed transparently

# Optimization Potential



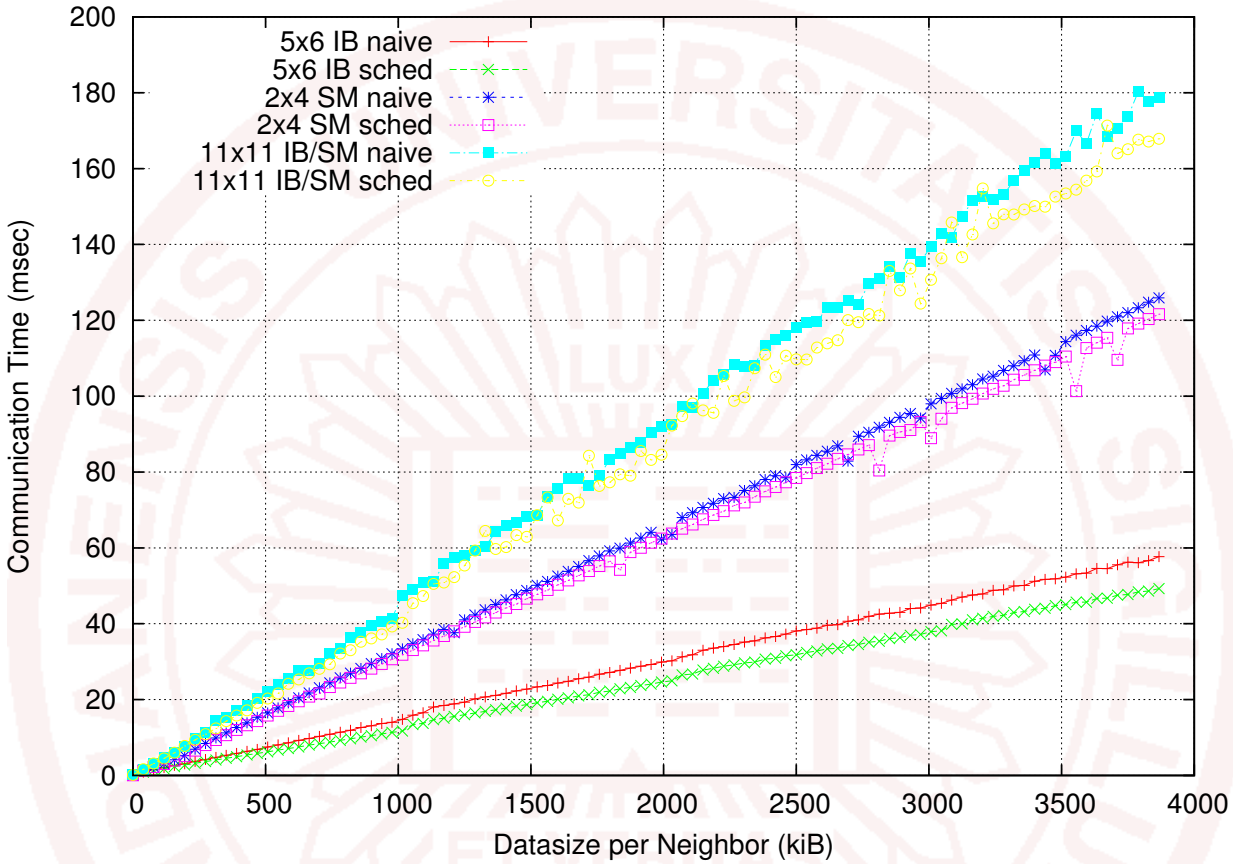
- 2d mesh, five point stencil, four neighbors
- portable implementation: start all isend, irecv + waitall
- more intelligent message scheduling could avoid contention
- depends on underlying network
- we used MPI_Sendrecv in dimension order

# Optimization Potential



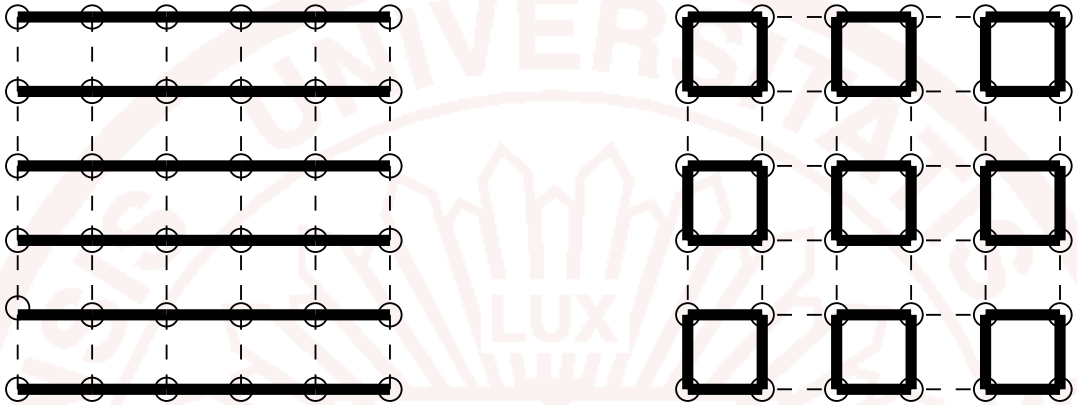

- on NEC SX-8

# Optimization Potential



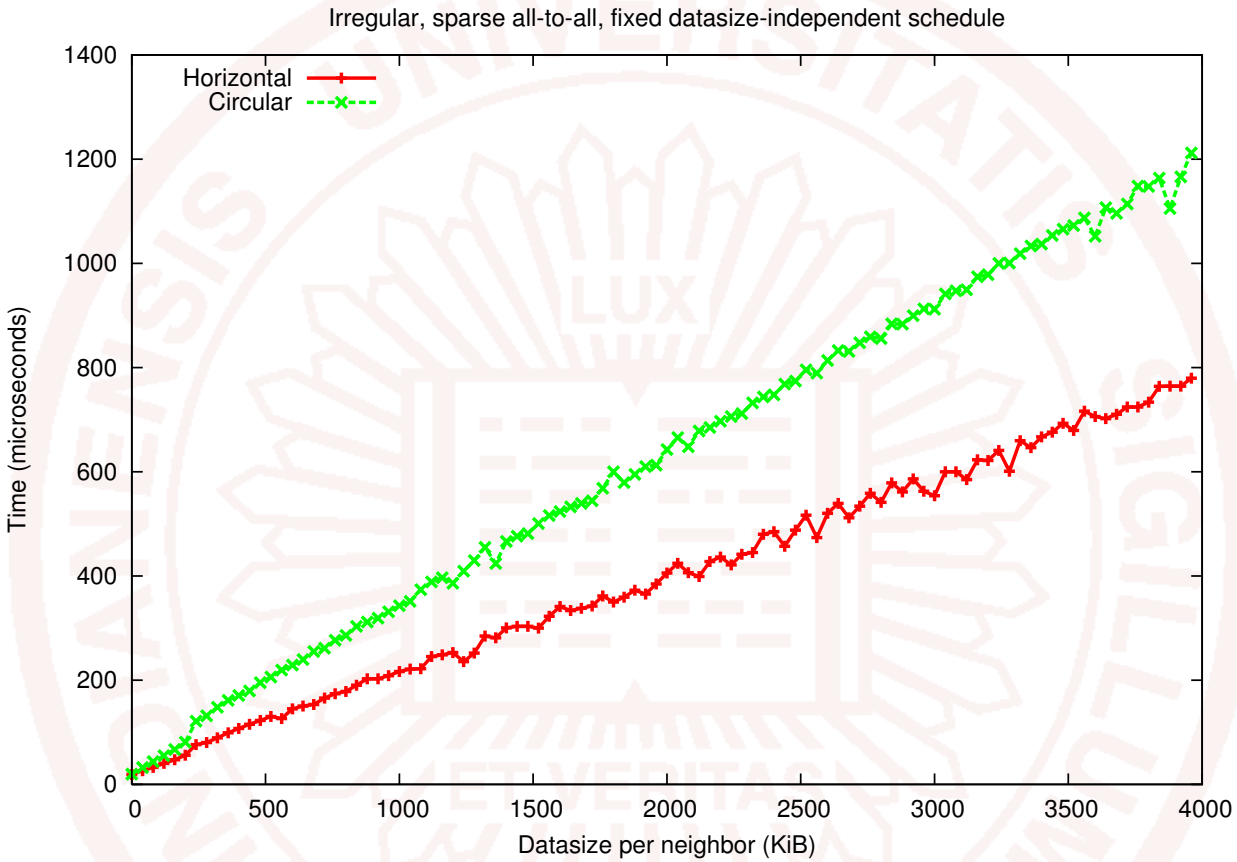

- on InfiniBand (IB) and shared memory (SM) and mixed
- used 32 nodes, 1 ppn on 5x6 grid and 4ppn on 11x11 grid
- SM results on single node

# Weighted Performance



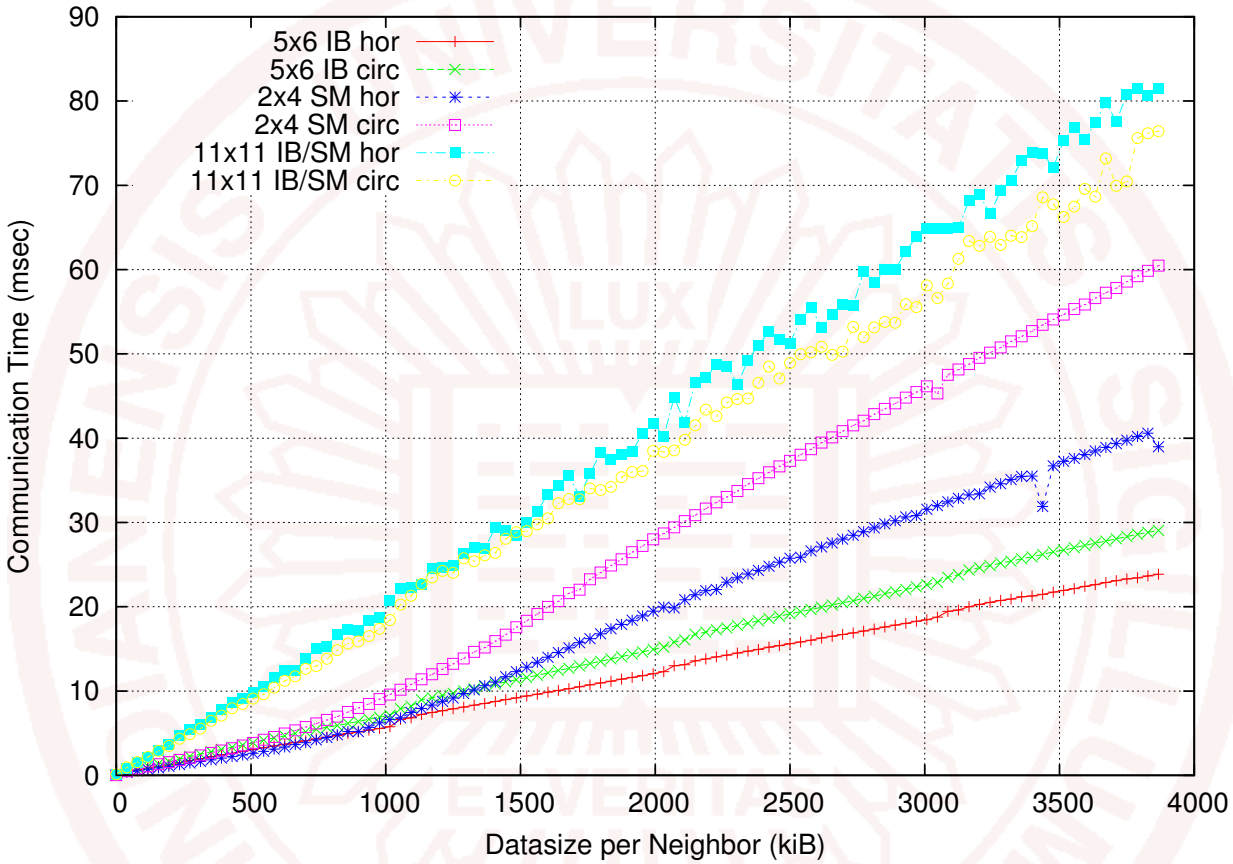
- heavy and light communications
- complexity is the same in both meshes
- best schedule is different
- left: dimension order, 1 heavy round
- right: dimension order, 2 heavy rounds
- right scheme can be fixed to one heavy round!

# Weighted Performance Results




- NEC SX-8

# Weighted Performance Results




- 32 IB nodes (IB, SM)

# Specifying Neighborhoods - Explicit Specification

## Explicit Specification – Separate for each Collective

```
MPI_Neighbor_alltoall_set(sources,sourceweights,
                          targets,targetweights,
                          info,comm)
```

## Explicit Specification – One for all Collectives

```
MPI_Neighborhood_set(operation,
                     sources,sourceweights,
                     targets,targetweights,
                     info,comm)
```

- operation (optional) would be a bit-vector that specifies which operations are configured

# Specifying Neighborhoods - Virtual Topologies

- second option would be to use virtual topology functions
- builds on (scalable) graph or cartesian topologies
- depends on ticket #33
- weights already exist
- schedules could be created during construction
- changed neighborhoods require new communicators
- enables process reordering (very powerful)

# Sparse Collectives

Any Comments/Discussions?

# Topology Modifiers

## Why?

- Cartesian topologies are useful (easy to define, easy to store/map)
- often more flexibility is needed (e.g., diagonals - see #72)
- topology modifiers would provide MPI with more knowledge

## How?

- two ways:
  1. query Cartesian topology for neighbors and create graph topology
  2. derive graph topology from Cartesian topology

# Topology Modifiers

## Derive Graph Topology from Cartesian Topology

```
MPI_CART_DERIVE_GRAPH(comm_cart, ndirs, displs,
                      reorder, comm_graph)
 - IN comm_cart   communicator with Cartesian
                  structure (handle)
 - IN ndirs       number of directions
 - IN displs      array of ndirs vectors (array of
                  integer) giving the distance of
                  shift in each coordinate dimension
 - IN reorder     reorder (same as for graph_create)
 - OUT comm_graph graph communicator
```

- add weights and info (#33)
- seems somewhat complex (array of shift vectors)
- better proposals?

# Ticket #163

- proposal to comments list to allow MPI_IN_PLACE in each buffer
- potentially simplifies user-code
- adds another branch to each collective :-(
- any other comments?
- do we want to pursue this for MPI-3?

# Persistent Collectives

- more research needed!
- was there anything done?
- do we want to focus on it?

# Collective Operations

Any other Items or Discussions?