Collectives and Communicators: A Case for Orthogonality (Or: How to…)

Jesper Larsson Träff*, Sascha Hunold*, Guillaume Mercier+, Daniel J. Holmes#

traff@par.tuwien.ac.at

*TU Wien

+Bordeaux Institute of Technology, INRIA

#EPPC, University of Edinburg

Motivation I:

What's the difference?

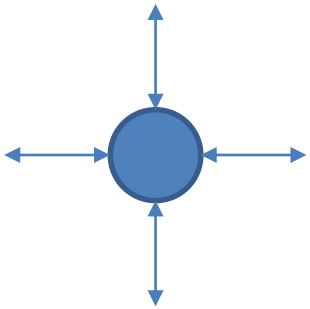
```
MPI_Neighbor_alltoallw(
  sendbuf,sendcounts[],senddispls[],sendtypes[],
  recvbuf,recvcounts[],recvdispls[],recvtypes[],
  neighborcomm)
```

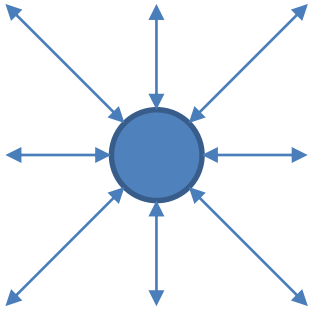
```
MPI_Alltoallw(
  sendbuf,sendcounts[],senddispls[],sendtypes[],
  recvbuf,recvcounts[],recvdispls[],recvtypes[],
  comm)
```

None: Neighborhood collectives and standard collectives have same interface signatures

Motivation II:

What's the difference?

1. 

2. 

1. Supported (reordering, collectives) by
`MPI_Cart_create(comm,…,&cart)`

2. Supported by? Perhaps via
`MPI_Dist_graph_create_adjacent(comm,…,&dist)`

This talk: Concepts and fundamentals

Leverage "orthogonality of MPI concepts" to

- unify intra-, inter-communicator and neighborhood collectives
- unify Cartesian and neighborhood collective communication

Cost: One new function, `MPI_Comm_base(comm,&base)`

Outcome: We can

- get rid of all special interface functions for `MPI_Neighbor_...` collective communication (regular, non-blocking, persistent: 15 interfaces)
- provide for additional (useful/strange) sparse collective operations at no interface burden
- provide for more powerful implementation support for Cartesian communication (reordering, collectives)
- beyond…

Unifying collective communication

Current MPI: Two schools of thought (re. collective communication and interfaces)

1. Separate interfaces for collective communication with different patterns and same communicator: `MPI_Alltoall` vs. `MPI_Neighbor_alltoall`.

2. Same interfaces for different patterns with different communicators: `MPI_Alltoall` on intra- vs. `MPI_Alltoall` on inter-communicator.

Semantics of collective operation determined by interface (1) or by communicator (2), `MPI_Neighbor_alltoall`: actually, both

Next MPI: Communication topology (as communicator) alone determines semantics of collective interface (Thought-school 2).

Types of topologies:

- Fully connected (standard intra-communicator): Logically flat, fully connected graph, any process can communicate with any other, collectives include all processes
- Bi-partite (inter-communicator): Two domains, process in one domain can communicate with processes in other domain, not in own domain, collective communication between domains
- (Sparse) Directed graphs: Processes have directed neighborhoods, process can communicate only with neighbors, collective communication over neighborhoods

`MPI_Alltoall(…,comm)`:

On standard intra-communicator, all-to-all exchange between all processes. Data in buffers in rank-order.

`MPI_Alltoall(…,intercomm)`:

On inter-communicator, all processes in one domain contributes data to all processes in other domain, *vice versa*. Point-to-point communication between processes in same domain not possible!

`MPI_Alltoall(…,distgraph)`:

On distributed graph communicator, collectives have neighborhood semantics. Point-to-point communication between non-adjancent processes disallowed! Data in buffers in order of neighbors, determined at communicator creation time.

Not possible/disallowed on distributed graphs:
Possibly helpful for debugging, and structured development to disallow and catch communication between processes that are not adjacent in topology?

"Advice to users":
If communication between non-adjacent processes is needed, use a (fully connected) communicator that allows it.

"Advice to users":
If standard, collective communication is needed on inter-communicator or distributed graph communicator, use a different communicator with fully connected topology.

"Advice to users":
If standard, collective communication is needed on inter-communicator, or distributed graph communicator, use a different communicator with fully connected topology.

The base communicator (for lack of better name) of a communicator (inter-communicator, distributed graph communicator) is a standard, fully connected intra-communicator over all the processes that are not connected in the communicator:

```
MPI_Comm_base(comm,&base)
```

One, new interface function (but not even necessary):

Collective semantics(?), can be implemented as non-synchronizing, non-blocking function.

Getting the base communicator can be implemented with standard functionality (with collective semantics)

```
MPI_Comm_group(comm,&localgroup);
if (process in localgroup) {
  MPI_Comm_create(comm,localgroup,&base);
}
```

Further, deprecate (save 1 interface)

```
MPI_Comm_test_inter(comm,&flag);
```

replace in code by

```
MPI_Topo_test(comm,&status);
flag = (status==MPI_INTER); // new status kind
```

Future MPI: Additional topologies

- Multi-partite: Several domains, collective communication across domains, possibly no point-to-point communication, …
- Non-disjoint, multi-partite: What if some domains overlap, what kinds of collectives?
- …

More collectives on distributed graphs

All MPI collective interfaces applicable on distributed graph communicators, and some of them could be given good meaning (not just "invalid", as `MPI_Scan` on inter-communicator); all with collective, non-synchronizing semantics.

`MPI_Bcast`: Broadcast data from root (all processes give same root) to all processes in neighborhood of root. No effect for processes not in root neighborhood.

`MPI_Allreduce`: Reduction in all neighborhoods, commutative operators only.

Etc.

Cartesian communication

Current MPI: Cartesian communicators specify both a naming scheme (rank = process coordinate vector) AND a communication pattern (each process neighbor to processes along dimensions).

Drawbacks:

- Inflexible, limited (non-weighted, fixed neighbor order)
- Inconsistent with distributed graph communicators (weights, info)
- Reordering wrt. implicit neighborhood only

Jesper Larsson Träff, Felix Donatus Lübbe, Antoine Rougier, Sascha Hunold: Isomorphic, Sparse MPI-like Collective Communication Operations for Parallel Stencil Computations. EuroMPI 2015: 10:1-10:10
Jesper Larsson Träff: SMP-Aware Message Passing Programming. HIPS 2003: 56-65

Next MPI: Cartesianness is purely and only a naming scheme. All topological aspects (reordering, collectives) by distributed graph communicators.

```
MPI_Cart_name(d,dims,periods,…,&comm)
```

Row major or column major coordinates…

Naming scheme (d, dims, periods, etc.) attached to communicator, no new communicator created. Might or might not be collective(?): All processes in `comm` must specify same scheme. No reordering here.

Naming scheme gives names to processes (coordinates in d-dimensional space), and translations between names and ranks. Essential functions:

- From name (coordinate) to rank
- From rank to name (coordinate)

```
MPI_Cart_rank(cart,coords,&rank);
MPI_Cart_coords(cart,rank,maxdims,coords);
```

Naming scheme in communicator

All else: Can be implemented on top of MPI in separate libraries

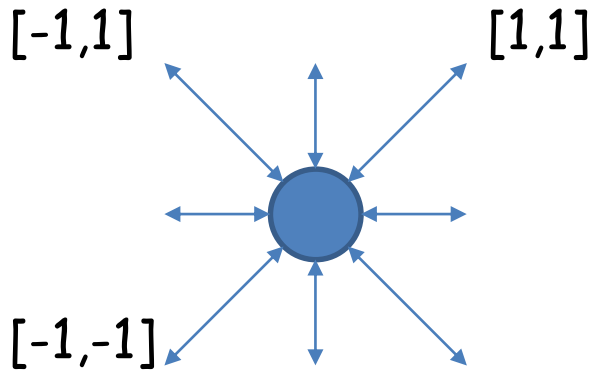
Cartesianness is not at topological property of a communicator.
No (deprecate)

```
MPI_Topo_test(comm,&status);
flag = (status==MPI_CART); // new status kind
```

but maybe some other query function

Example: Stencil communication pattern

Neighborhood described by list of relative coordinates (offsets):

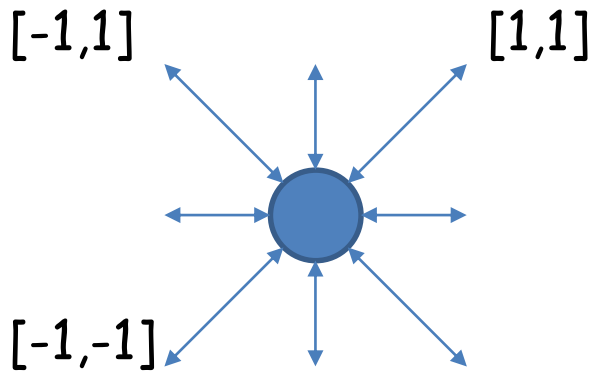


Naming scheme in communicator

Library functions:

```
Cart_offset_rank(cart,offset,&rank);
Cart_offset_coords(cart,rank,maxdims,offset);

Cart_offset_shift(cart,offset,
                  &sourcerank,&destrank);
Cart_neighborhood(cart,t,neighbors,
                  sources,destinations);
```



IN: Symmetric neighborhood as list of outgoing neighbors, flattened offsets

```
int neighbors[] =
  {1,0, 1,1, 0,1, -1,1, -1,0, -1,-1, 0,-1, 1,-1};
int t= 8;
Cart_neighborhood(cart,t,neighbors,
                  sources,destinations);
```

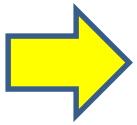
In suitable order (buffers)

OUT: Rank lists of incoming and outgoing neighbors, in form suitable for `MPI_Dist_graph_create_adjacent`

```
int neighbors[] =
  {1,0, 1,1, 0,1, -1,1, -1,0, -1,-1, 0,-1, 1,-1};
int t= 8;
Cart_neighborhood(cart,t,neighbors,
                  sources,destinations);

MPI_Dist_graph_create_adjacent(cart,t,
                               sources,
                               MPI_UNWEIGHTED,
                               destinations,
                               MPI_UNWEIGHTED,
                               MPI_INFO_NULL,
                               reorder,
                               &distgraph);
```

Reordering only here



IN: Stencil types (D2Q5, D3Q27, Moore, vonNeumann, …) with parameters

Helpful additional library functions (stencil library):

```
Cart_stencil_neighbors_count(cart,stencil_type,…,
                             &neighbors);

Cart_stencil_neighbors(cart,stencil_type,…,
                       sources,destinations);
```

OUT: Rank lists of incoming and outgoing neighbors, in form suitable for `MPI_Dist_graph_create_ajacent`

Current MPI: Reordering at the wrong places, with wrong information

```
MPI_Cart_create(comm,d,dims,periods,reorder,&cart);
int neighbors[] =
  {1,0, 1,1, 0,1, -1,1, -1,0, -1,-1, 0,-1, 1,-1};
int t= 8;

// create neighborhood by hand

MPI_Dist_graph_create_adjacent(cart,t,
                               sources,
                               MPI_UNWEIGHTED,
                               destinations,
                               MPI_UNWEIGHTED,
                               MPI_INFO_NULL,
                               reorder,
                               &distgraph);
```

Reorder here too early, no information on stencil

Reorder here might loose stencil information

Observation:

1. Better reorderings can be obtained by taking full stencil communication pattern into account
2. Factorization not always the best approach

Konrad von Kirchbach, Markus Lehr, Sascha Hunold, Christian Schulz, Jesper Larsson Träff: Efficient process-to-node mapping algorithms for stencil computations. CLUSTER 2020.

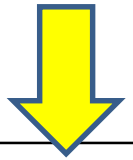
Christoph Niethammer, Rolf Rabenseifner: An MPI interface for application and hardware aware cartesian topology optimization. EuroMPI 2019: 6:1-6:8
William D. Gropp: Using Node Information to Implement MPI Cartesian Topologies. EuroMPI 2018: 18:1-18:9

Observation:

- In stencil-patterns, all processes have same relative coordinate neighborhood ("isomorphic neighborhoods").
- This information can sometimes be used to improve neighborhood collective operations ("Cartesian Collective Communication").

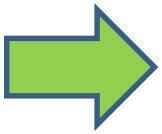
Jesper Larsson Träff, Sascha Hunold: Cartesian Collective Communication. ICPP 2019: 48:1-48:11

Naming scheme in communicator makes it possible to detect whether specified neighborhoods are "isomorphic"

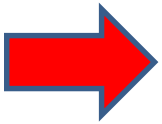


```
MPI_Dist_graph_create_adjacent(cart,t,
                               sources,
                               MPI_UNWEIGHTED,
                               destinations,
                               MPI_UNWEIGHTED,
                               MPI_INFO_NULL,
                               reorder,
                               &distgraph);
```

Order determines order of buffers



No information in INFO needed



Information provided with communicator naming scheme makes it possible for graph creation function to select proper collective algorithms

General problem for graph creating functions

Discovering structural properties of distributed communication graph to improve collective operations and/or perform reordering for given communication graph G=(V,E) is often a hard problem:

- Property: |V| can be factorized – easy (for given d)
- Property: G is k-regular – easy (for given k)
- Property: G is stencil-like (k-regular, all processes have similar neighborhoods) - hard

Our solution:
Help the implementation by providing background information, e.g., that processes are located in Cartesian space.

Very hard to do via INFO object (that also has different semantics)

Checking for stencil pattern easy in Cartesian space

1. Check whether communicator has Cartesian naming
2. Create lists of neighbors (incoming, outgoing)
3. Translate lists of neighbors into lists of coordinates
4. Sort lexicographically (such that all processes have neighbors in same order)
5. Check that for each outgoing neighbor there is an incoming, opposite neighbor, *vice versa*
6. Process 0 broadcasts its number of neighbors, all processes check for same number of neighbors
7. Process 0 broadcasts its lists of neighbors, all processes check whether they have identical lists

Algorithm inside `MPI_Dist_graph_create(_adjacent)` takes only a few broadcast and allreduce operations

Collective communication with stencil patterns

Get base communicator; essential that base retains naming scheme.

```
MPI_Comm_base(distgraph,&base);
while (iterate) {
  // compute on matrix

  // exchange
  MPI_Alltoallw(matrix,sendcount,senddisp,sendtype,
                matrix,recvcount,recvdisp,recvtype,
                distgraph);

  // check for convergence
  MPI_Allreduce(local,&iterate,1,MPI_INT,MPI_LAND,
                base);
}
```

Neighborhood collective on distributed graph

Convergence check on fully connected communicator

Future MPI: Other naming schemes

- Polar coordinates
- Exotic geometries
- Regularly tiled structures
- …

Christoph Niethammer, Rolf Rabenseifner: An MPI interface for application and hardware aware cartesian topology optimization. EuroMPI 2019: 6:1-6:8

Next MPI: Deprecate (save 1 interface)

```
MPI_Dims_create(size,d,dims);
```

Finding the right factorization of a number of MPI processes into dimension sizes is outside the scope of MPI:

- Hard problem (factorization)
- Best factorization is application dependent
- Best factorization is dependent on communication pattern
- Best factorization is dependent on hardware topology

MPI should enable writing good application specific libraries for finding the right factorization

Summary

Contributions/suggestions:

- Get rid of separate interfaces for neighborhood collective communication, `MPI_Neighbor_allgather`, `MPI_Neighbor_alltoallw`, etc.
- Provide reordering and collective support for arbitrary stencils by simplifying Cartesian interface

Orthogonal concepts:

- Communicator defines topology (fully connected, bi-partite, sparse graph) and semantics of collective interfaces
- Cartesian naming scheme
- Single interface class for creating graph topologies and performing reordering: `MPI_Dist_graph_create`, `MPI_Dist_graph_create_adjacent`