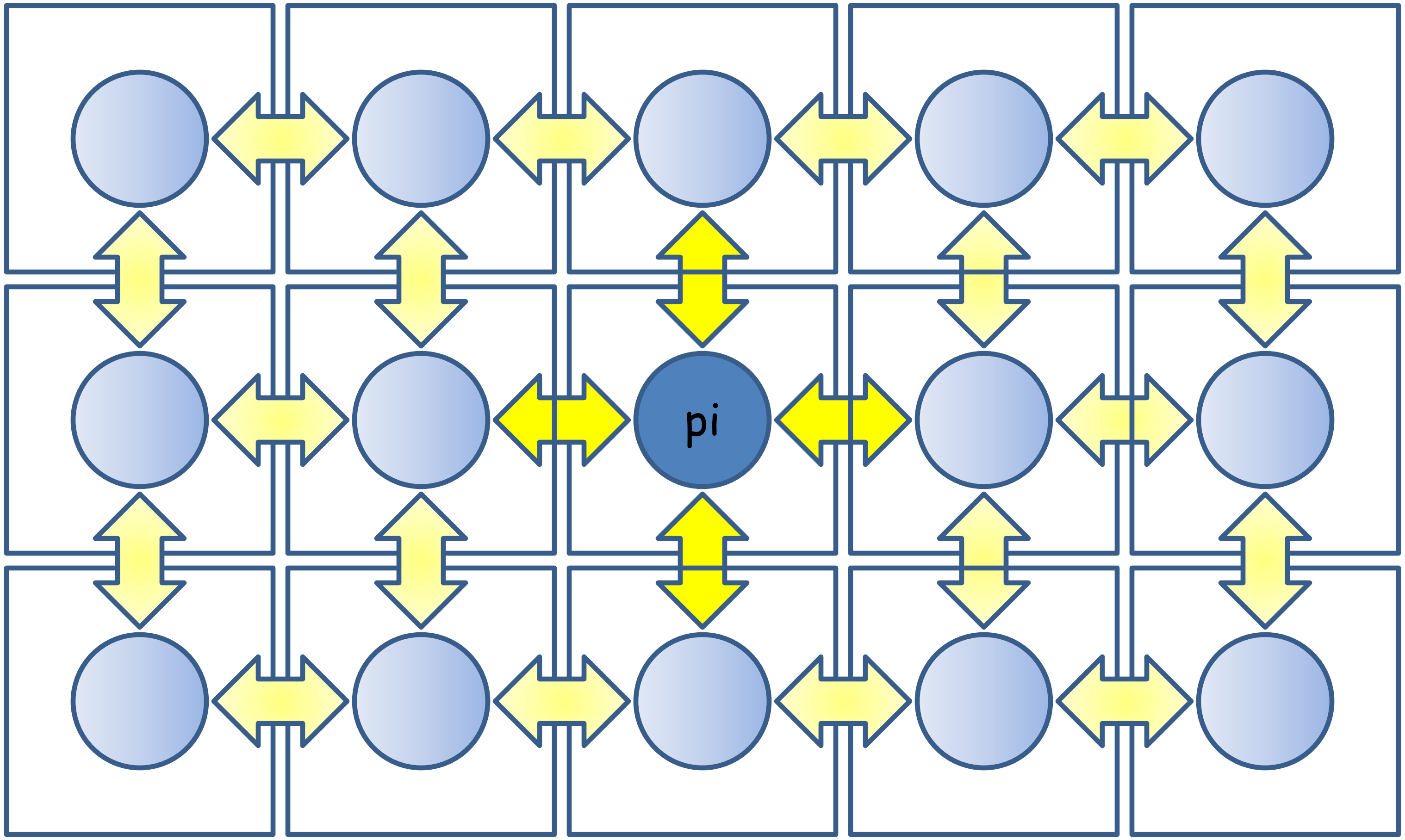
Isomorphic, Sparse MPI-like Collective Communication Operations for Parallel Stencil Computations

Jesper Larsson Träff, Felix Lübbe, Antoine Rougier, Sascha Hunold
{traff,luebbe,rougier,hunold}@par.tuwien.ac.at

Vienna University of Technology
Faculty of Informatics, Institute for Information Systems
Research Group Parallel Computing


pi

Situation

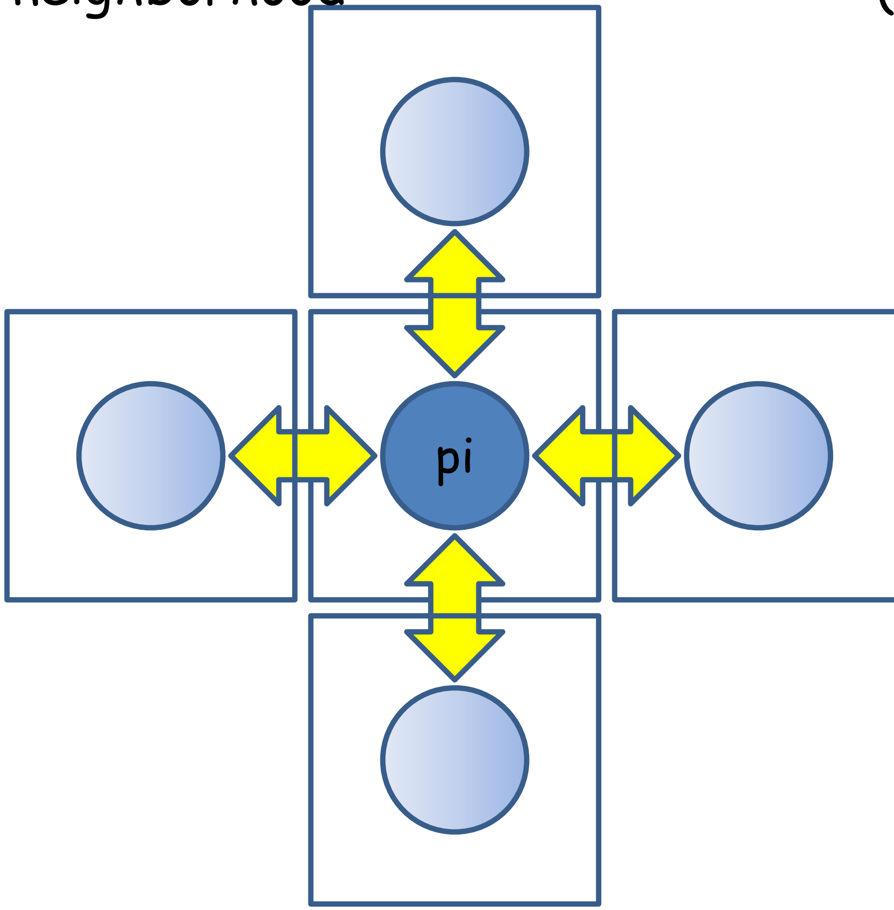
•Processes arranged in some regular structure (mesh, torus, …)
•Each process needs to communicate with a small neighborhood
•All processes have similar neighborhoods
•All processes involved: collective communication

Problem

•Is MPI support for this type of communication adequate/good?
•Can it potentially be supported better and more efficiently?

MPI 3.0 intention: non-blocking neighborhood collectives

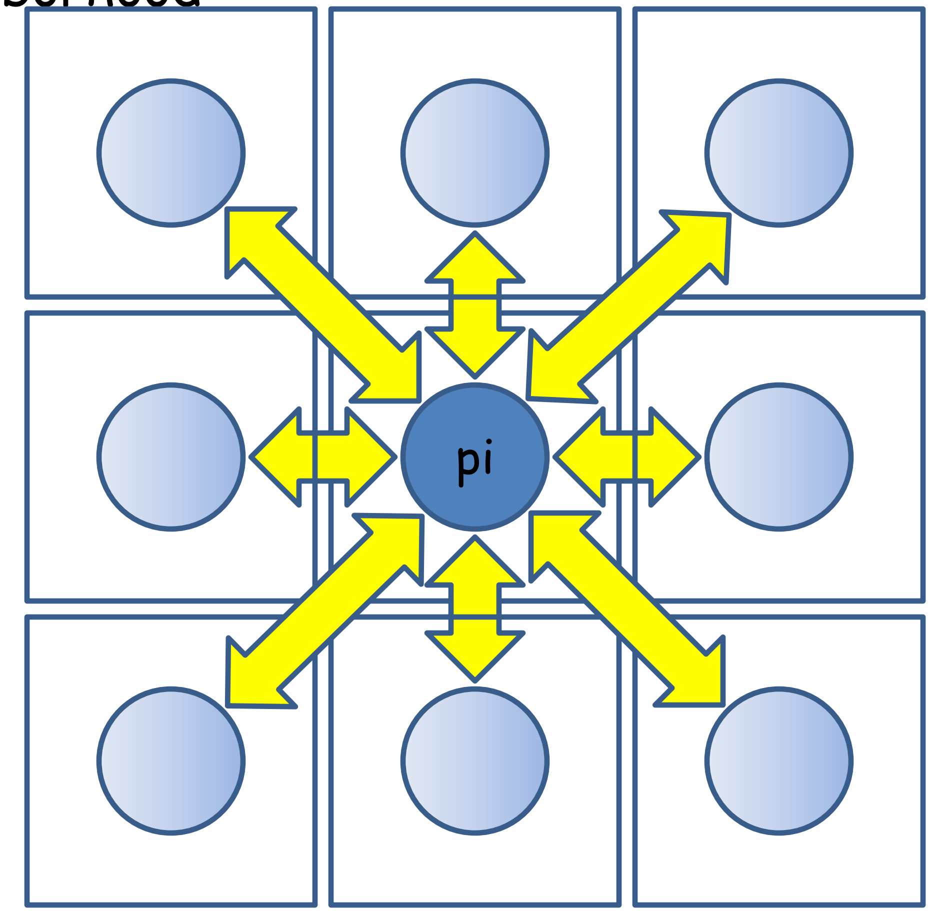
von Neumann neighborhood (5-point stencil)



Each process communicates with processes in mesh/torus that are exactly one "hop" away (Manhattan distance 1)

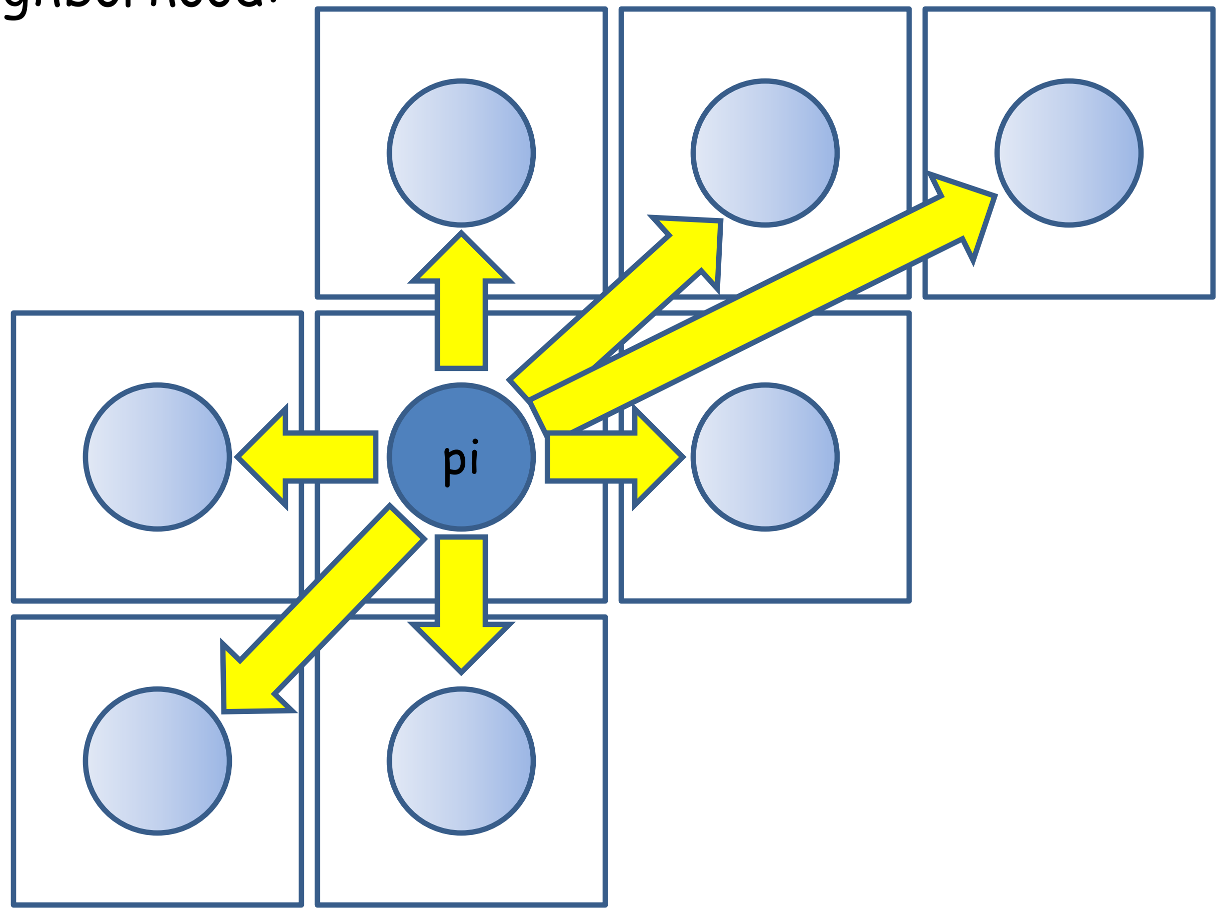
Moore neighborhood

(9-point stencil)



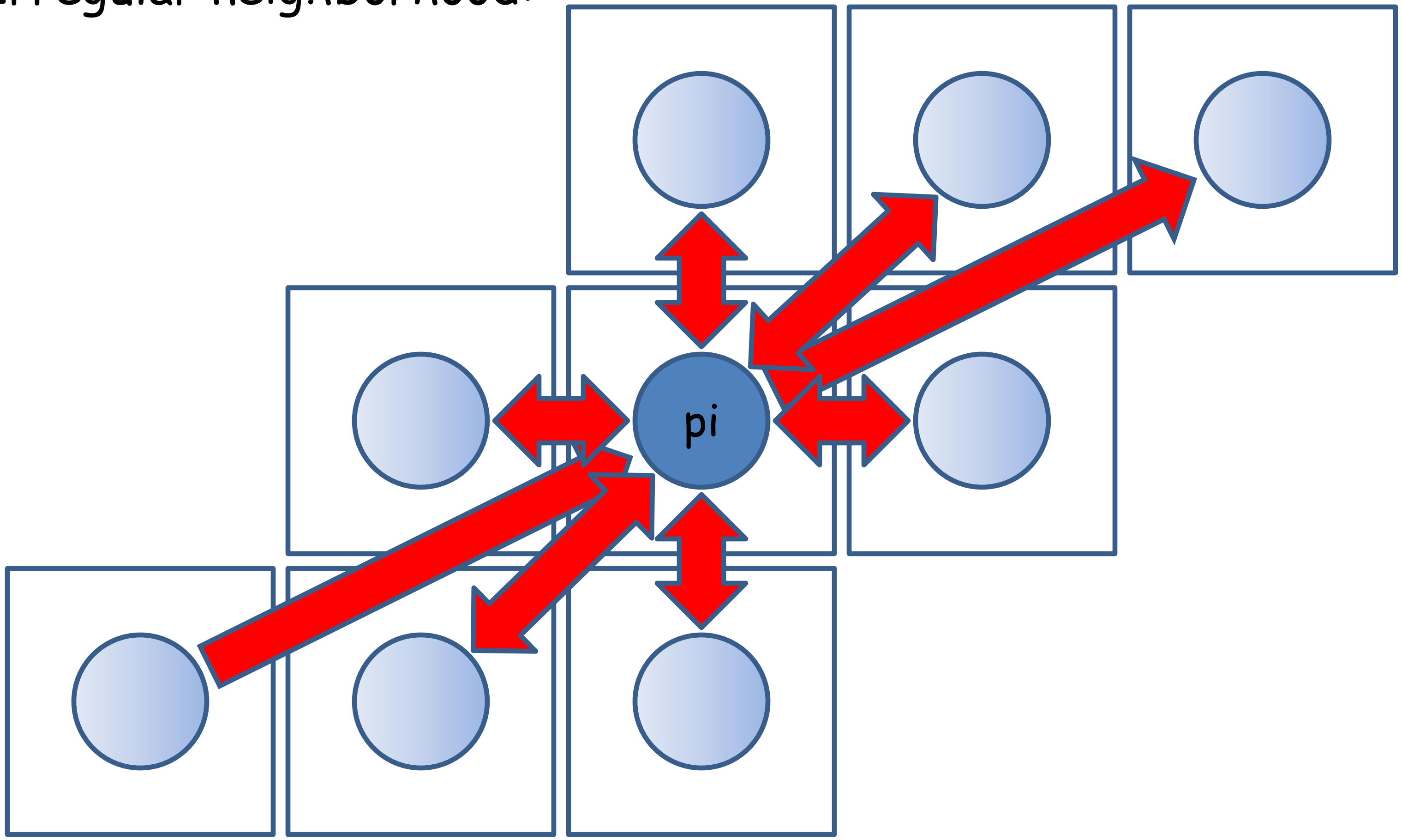
Each process communicates with processes in mesh/torus where at least one coordinate differs by one (Chebyshev distance 1)

Irregular neighborhood:



Each process sends data to s other processes. All processes use same pattern, per symmetry receives also from s processes

Irregular neighborhood:



Each process sends data to s other processes. All processes use same pattern, per symmetry receives also from s processes

Observations

- All processes use same collective communication pattern

Isomorphic, collective communication

- Convenient to describe neighborhoods relative to some underlying, regular structure (d-dimensional mesh, torus, …)

"Isomorphic" requirement easily fulfilled

MPI 3.0 solution

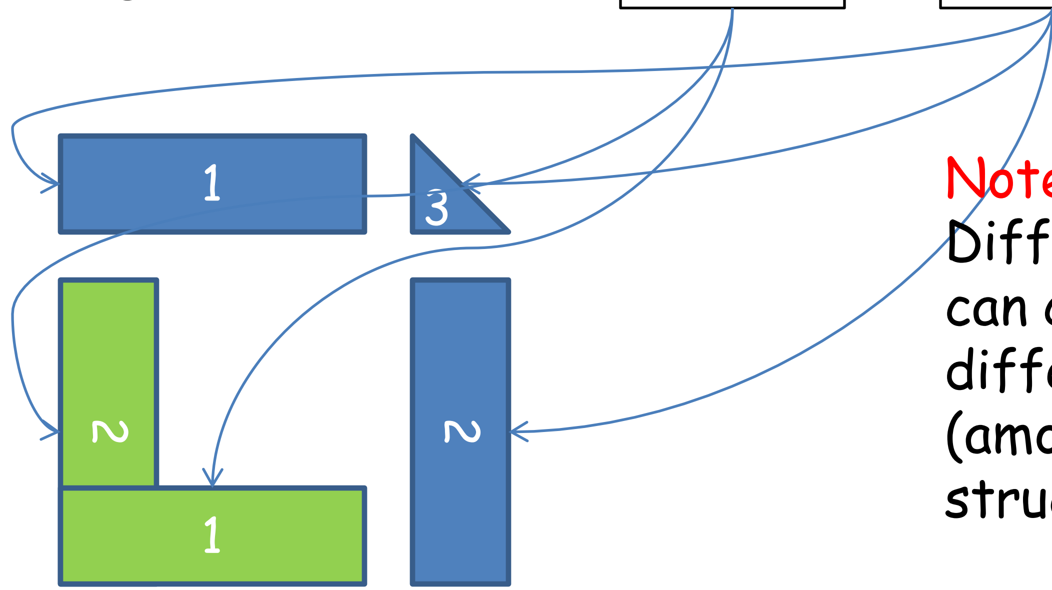
Use sparse, non-blocking collectives for neighborhood communication:

```
MPI_Ineighbor_alltoall()
MPI_Ineighbor_alltoallv()
MPI_Ineighbor_alltoallw()

MPI_Ineighbor_allgather()
MPI_Ineighbor_allgatherv()
```

Collective data exchange operations for some given, per process neighborhood


```
MPI_Neighbor_alltoallw(sendbuf,…,recvbuf,…,comm);
```



Note (-w collective): Different neighbors can communicate different data (amount and structure)

- Local neighborhood: implicit list of source (incoming) and destination (outgoing) neighbors (MPI processes)
- Order of neighbors determine order of communication buffers
- Correctness/deadlock freedom: Destinations and sources must match
- Neighborhoods are fixed and associated with communicator

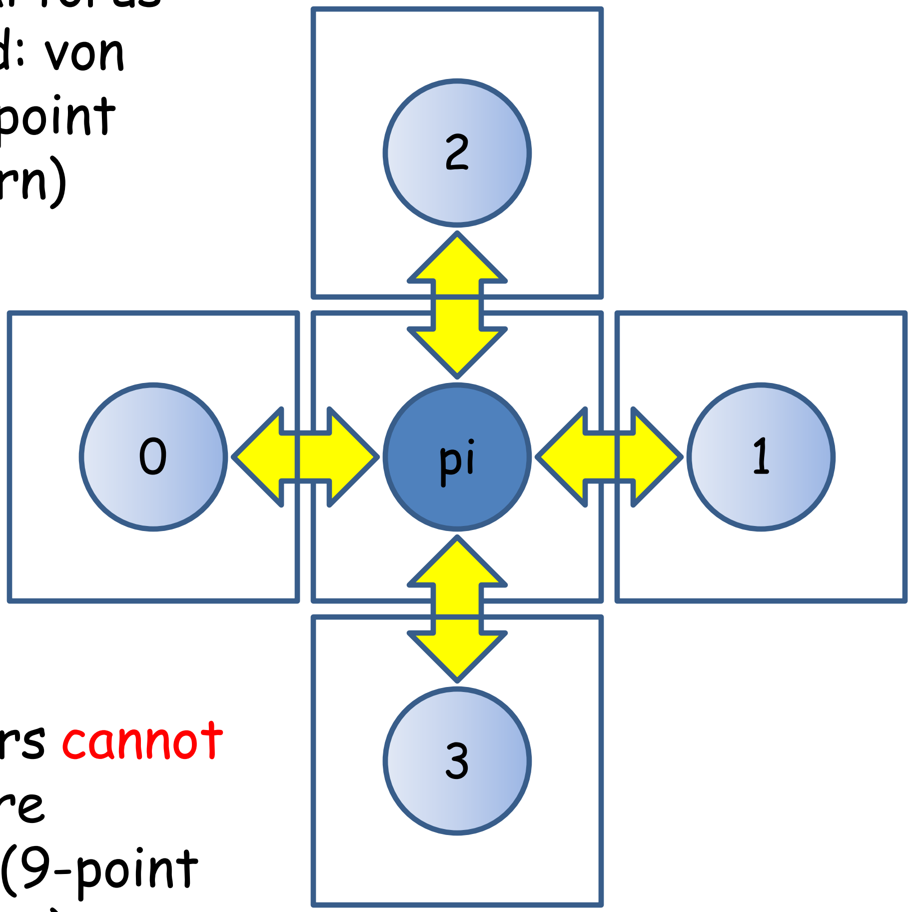
Implicit neighborhoods: Cartesian communicators

d-dimensional mesh/torus MPI process structures created with

```
MPI_Cart_create(comm,d,…,reorder,&cartcomm);
```

- Processes organized into d-dimensional torus pattern (with given size of dimensions)
- Isomorphic: neighborhoods defined for all processes as immediate torus neighbors (-/+ 1 hop in each dimension)
- Symmetric: all neighbors both source and destination
- Order of neighbors is fixed, dimensionwise (0 to d-1), -/+
- Number of neighbors is always 2*d (for meshes, some non-existent, MPI_PROC_NULL)

2-dimensional torus neighborhood: von Neumann (5-point stencil pattern)

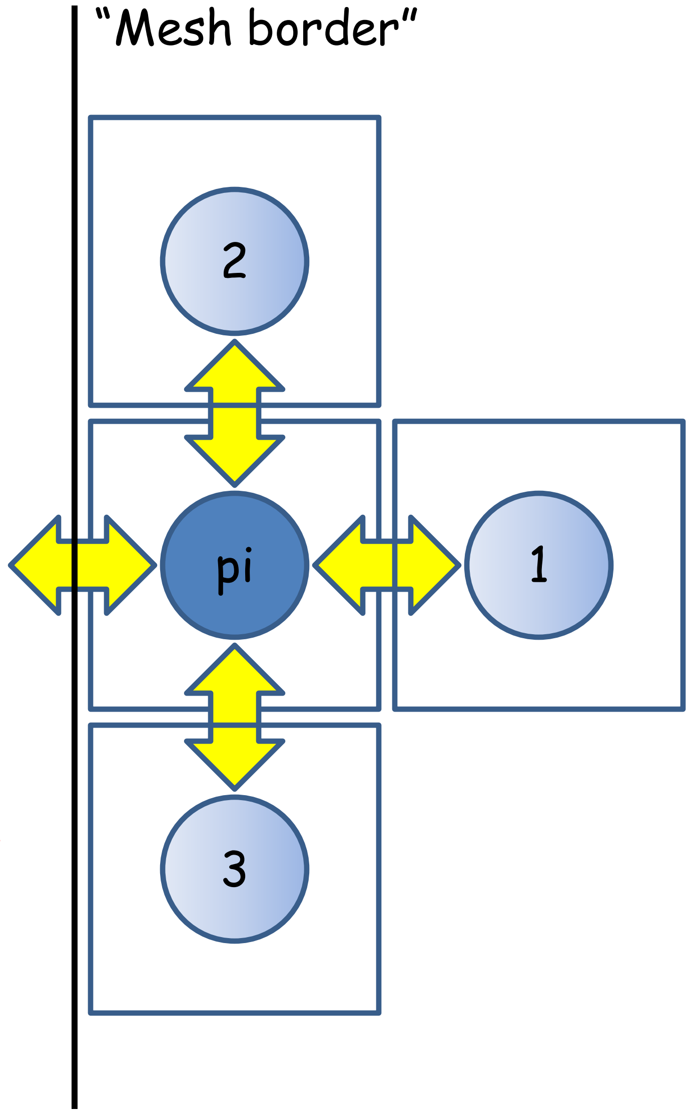


Cartesian communicators cannot express Moore neighborhod (9-point stencil pattern)

2-dimensional mesh neighborhood

MPI_PROC_NULL neighbor

Cartesian communicators cannot express Moore neighborhod (9-point stencil pattern)



```
MPI_Cart_create(comm,d,…,reorder,&cartcomm);
```

Makes it possible for MPI library to map torus communication topology to actual network topology

Contrast to explicit graph communicators:

- No weights on communication edges (could help mapping)
- No MPI_Info object (could pass information on how to map)

- No query functions for neighborhood: Cart_neighborhood_get()

Get neighbors with MPI_Cart_shift()

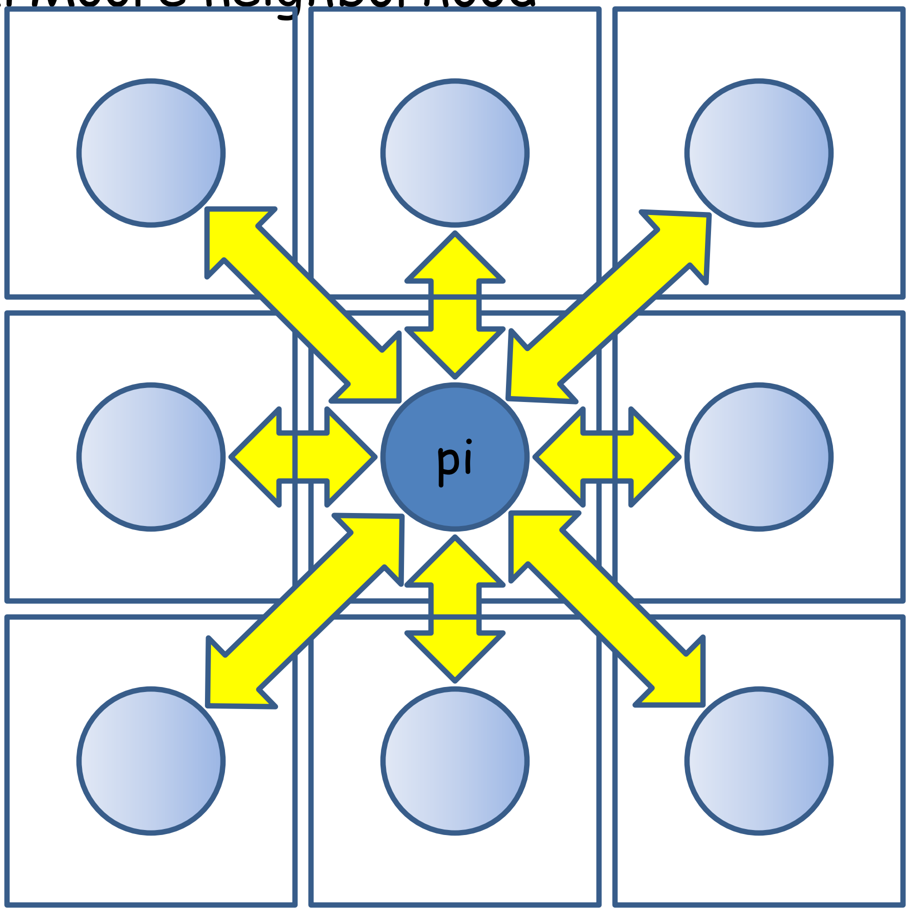
Explicit neighborhoods: general graph communicators

Arbitrary directed communication (multi-)graph created with

```
MPI_Dist_graph_create(_adjacent)(comm,
                                 <edges>,<weights>,
                                 info,reorder,
                                 &graphcomm);
```

- Communication edges between "real" MPI processes only (no MPI_PROC_NULL)
- Local neighborhoods consist of adjacent processes: sources and destinations (not necessarily symmetric)
- Not necessarily isomorphic neighborhoods
- Neighborhood query functions: MPI_Dist_graph_neighbors()
- Edge weights and MPI_Info object to guide mapping

2-dimensional Moore neighborhood



… must be specified as general graph topology. User must determine explicit ranks of all 8 neighbors, must use query function to later determine order of neighbors

Summary: MPI 3.0 implicit and explicit neighborhoods

- Cartesian communicators severely limited: express only von Neumann type neighborhoods

- General graph topologies perhaps too general for many common situations, no information on global graph structure

- Unfortunate differences between Cartesian and general graph communicator functionalities (info/weights, query functionality, MPI_PROC_NULL)

Same neighborhood set up in two different ways can have quite different properties

"Isomorphic neighborhoods" a valuable assertion for sparse collective communication algorithms

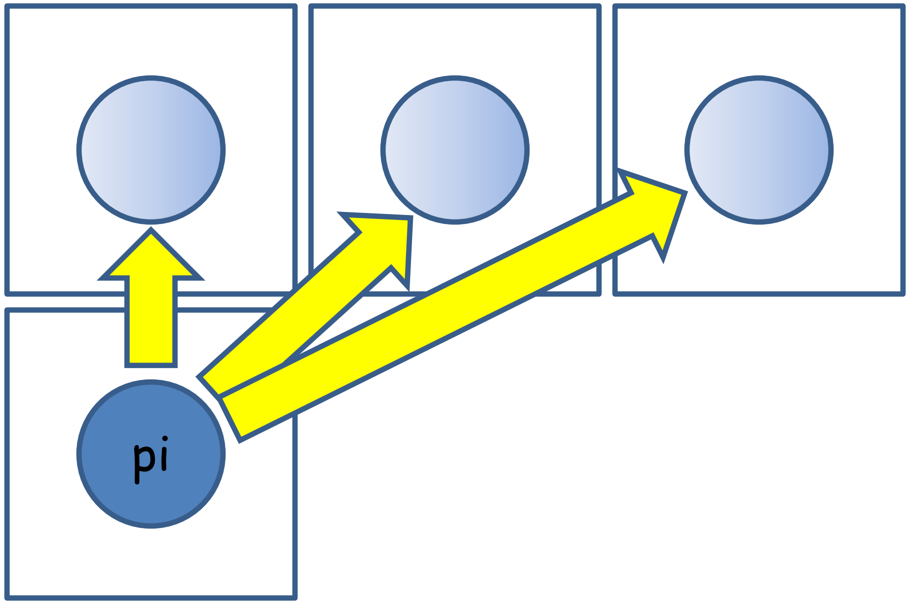
An extension and a restriction: isomorphic, sparse collectives

Given d-dimensional Cartesian communicator: mesh (periodic in some dimensions), torus

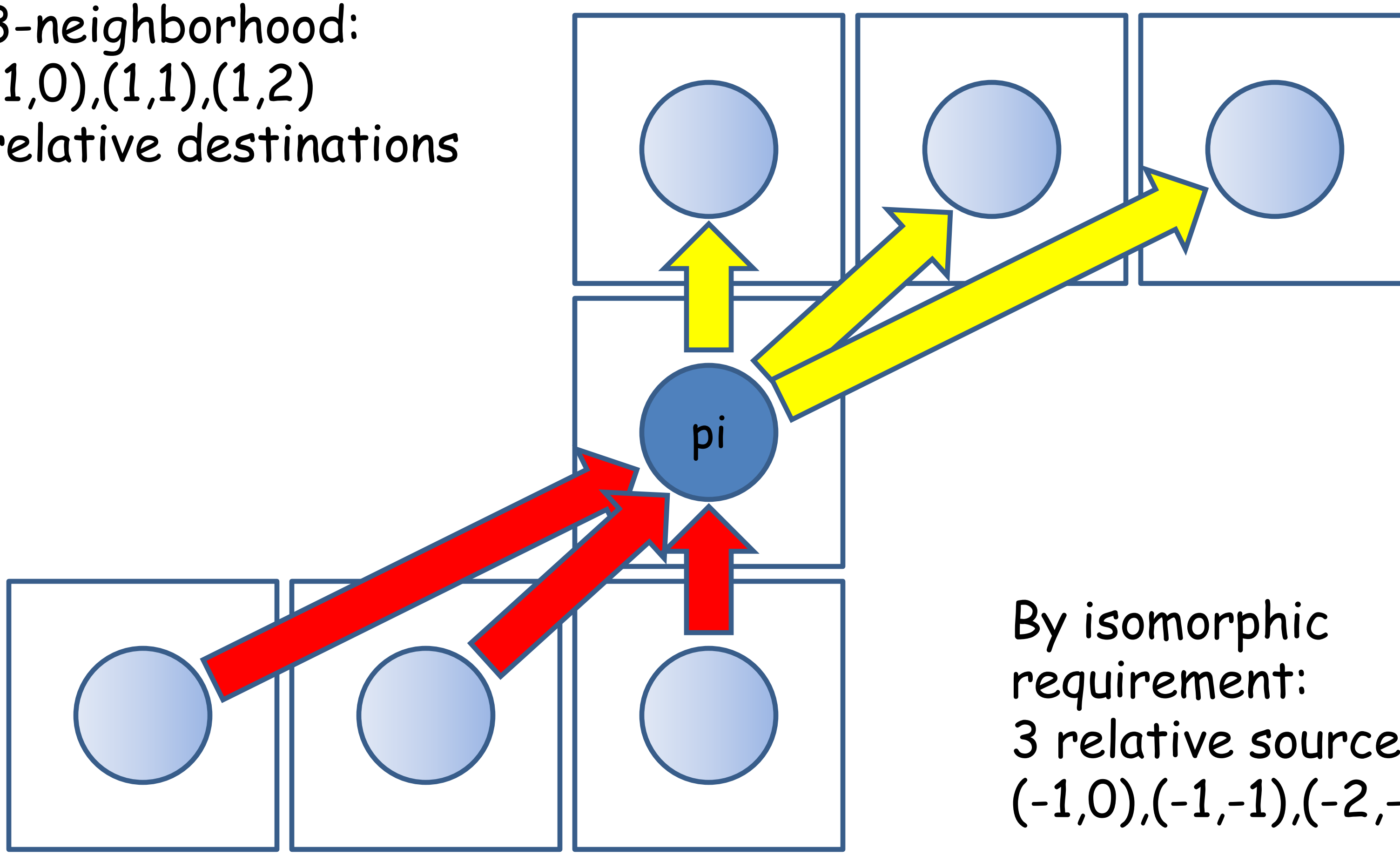
Isomorphic (sparse) neighborhoods

- Neighborhood given by (ordered) list of relative coordinates (d-vectors)
- Only destination coordinates listed (sources implied by isomorphic requirement)
- Isomorphic: all processes use (exactly!) same neighborhood (same coordinates, same order)

3-neighborhood:
(1,0),(1,1),(1,2)
relative destinations

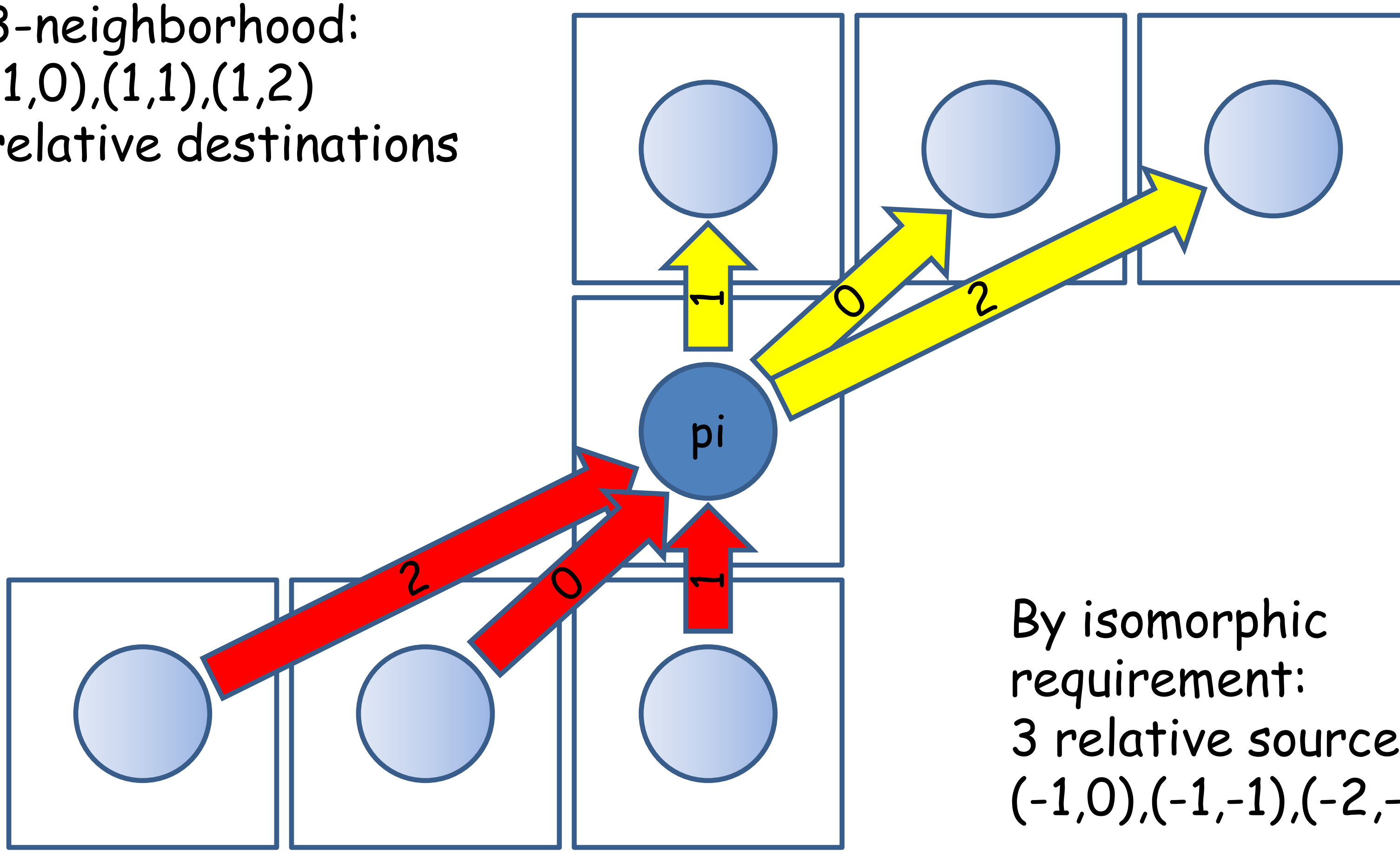


3-neighborhood:
(1,0),(1,1),(1,2)
relative destinations



By isomorphic requirement:
3 relative sources
(-1,0),(-1,-1),(-2,-1)

3-neighborhood:
(1,0),(1,1),(1,2)
relative destinations



By isomorphic requirement:
3 relative sources
(-1,0),(-1,-1),(-2,-1)

Repetitions allowed

s-neighborhood: list of s relative coordinates [C0, C1, C2, C3, …]

Example, 5-dimensions: [(0,1,1,3,1), (1,1,0,1,2), (-2,-2,-2,-2,-2), (0,1,1,3,1), …] (C-style: flattened into simple array)

•s destination neighbors for rank R in absolute coordinates: R+C0, R+C1, R+C2, R+C3, …

•s source neighbors for rank R: R-C0, R-C1, R-C2, R-C3, …

```
Iso_neighborhood_create(MPI_Comm cartcomm,
                        int s,
                        int relative_coordinates[],
                        MPI_Comm *isocomm)
```

isocomm still Cartesian

```
Iso_neighborhood_create(MPI_Comm cartcomm,
                        int s,
                        int relative_coordinates[],
                        MPI_Comm *isocomm)
```

Implementation (very cheap):
Attaches neighborhood list to cartcomm, precomputes absolute ranks, etc.

(a "real" implementation would need to create new isocomm, but this can partly be precomputed with cartcomm and amortized)

Design decision: no reorder possibility, no weights, no MPI_Info. Process mapping done by Cart_create()

Convenience functions: navigation

```
Cart_relative_rank(MPI_Comm cartcomm, int relative[],
                   int *rank)
Cart_relative_coord(MPI_Comm cartcomm, int rank,
                    int relative)
Cart_relative_shift(MPI_Comm cartcomm,
                    int relative_shift[],
                    int *source, int *target)
```

Implementation: easy, on top of MPI Cartesian functionality

Convenience functions: query

Similar functions for Cartesian communicators

```
Iso_neighborhood_count(MPI_Comm isocomm,
                       int *s,
                       int *indegree, int *outdegree)
Iso_neighborhood_get(MPI_Comm isocomm, int max_s,
                     int sources[], int destinations[])

Iso_neighborhood_graph_get(MPI_Comm isocomm, int max_s,
                           int sources[],
                           int destinations[])
```

1. First get function: return all neighbors, including MPI_PROC_NULL ones
2. Second get function: return only "real" neighbors, no MPI_PROC_NULL
3. Output format for MPI_Dist_graph_create_adjacent()

Convenience functions: query

Similar functions for Cartesian communicators

```
Iso_neighborhood_count(MPI_Comm isocomm,
                       int *s,
                       int *indegree, int *outdegree)
Iso_neighborhood_get(MPI_Comm isocomm, int max_s,
                     int sources[], int destinations[])

Iso_neighborhood_graph_get(MPI_Comm isocomm,   t max_s,
                           int sources[],
                           int destinations[])
```

Implementation: easy, return attached, precomputed r k information

List of absolute process ranks

Isomorphic blocking collectives

```
Iso_neighbor_alltoall(void *sendbuf, …
                      void *recvbuf, …,
                      MPI_Comm isocomm)
Iso_neighbor_alltoallv(void *sendbuf, …
                       void *recvbuf, …,
                       MPI_Comm isocomm)
Iso_neighbor_alltoallw(void *sendbuf, …,
                       void *recvbuf, …,
                       MPI_Comm isocomm)
```

Probably most useful alltoall variant: different neighbors have different structure (datatype) and volume

Interface scalablity not an issue for sparse neighborhoods

Isomorphic blocking collectives

```
Iso_neighbor_allgather(void *sendbuf, …
                       void *recvbuf, …,
                       MPI_Comm isocomm)
Iso_neighbor_allgatherv(void *sendbuf, …
                        void *recvbuf, …,
                        MPI_Comm isocomm)
Iso_neighbor_allgatherw(void *sendbuf, …,
                        void *recvbuf, …,
                        MPI_Comm isocomm)
```

Not in MPI 3.0 but equally useful

Isomorphic blocking collectives

```
Iso_neighbor_reduce (void *sendbuf, void *recvbuf, …,
                     MPI_Comm isocomm)
Iso_neighbor_reduce_scatter(void *sendbuf, …
                            void *recvbuf, …,
                            MPI_Comm isocomm)
```

Reduction neighbor collectives dropped for MPI 3.0

And non-blocking variants: Iiso_neighbor_alltoallw(), …

Prototype library implementation at
www.par.tuwien.ac.at/Downloads/TUWMPI/tuwisosparse.tgz

Algorithms' observations

Isomorphic property makes deadlock-free implementations trivial

```
// pseudo: negate all coordinates
minusrelative = -relative

for (i=0; i<s; i++) {
  Cart_relative_rank(isocomm,relative+i*d,&destrank);
  Cart_relative_rank(isocomm,minusrelative+i*d,
                     &sourcerank);
  MPI_Sendrecv(sendbuf[i],…,destrank,ISOTAG,
               recvbuf[i],…,sourcerank,ISOTAG,
               comm,MPI_STATUS_IGNORE);
}
```

Does not work with graph neighborhoods: not known that i'th destination matches i'th source

Key observation:

By isomorphic condition (a global graph property), communication schedules can be computed locally and still be deadlock free processes will compute same schedule

Idea 1: Dimension-wise message combining

Write C as linear combination of coordinate basis vectors

$C = \{a0\ V0\}+\{a1\ V1\}+\{a2\ V2\}+\ldots+\{ad\ Vd\}$, $Vi = (0,0,\ldots 0,1,0,\ldots 0)$

Send (and receive) in d' rounds, in round i combine all messages with same {ai Vi} component

Key observation:

By isomorphic condition (a global graph property), communication schedules can be computed locally and still be deadlock free processes will compute same schedule

Idea 2: Logarithmic round message combining (Bruck)

Assume all neighbors Ci are linearly dependent, Ci = i*a*K for some relative coordinate K.

With message-combining, log s rounds suffice, s/2 blocks per round

Not yet implemented

Experimental work

Implementations:

•Iso_neighborhood_create() by attaching information to Cartesian communicator

•Iso_neighbor_alltoallw() with non-blocking send-recive operations

Towards a systematic benchmark for neighborhood communication

Questions:

•How expensive is neighborhood creation in comparison to Cartesian and graph communicators? Do Cartesian and graph communicator setup times differ? Does the communication graph play a role?

•How well does the simple alltoallw implementation compare to MPI 3.0 functionality?

Systems

Small, 36-node InfiniBand cluster, each node with two 8-core AMD 6134 Opteron processors at 2.3GHz, Mellanox IB MT4036 QDR

Three different MPI libraries:
necMPI (1.3.1), mvapich (2.2.1), OpenMPI (1.8.4)

Not here/not in paper:
Experiments also on 64 32-core nodes of Cray XC40 system with Cray-mpich (7.0.4) at KTH, Sweden

Communicator creation times

von Neumann and Moore neighborhoods with different radii

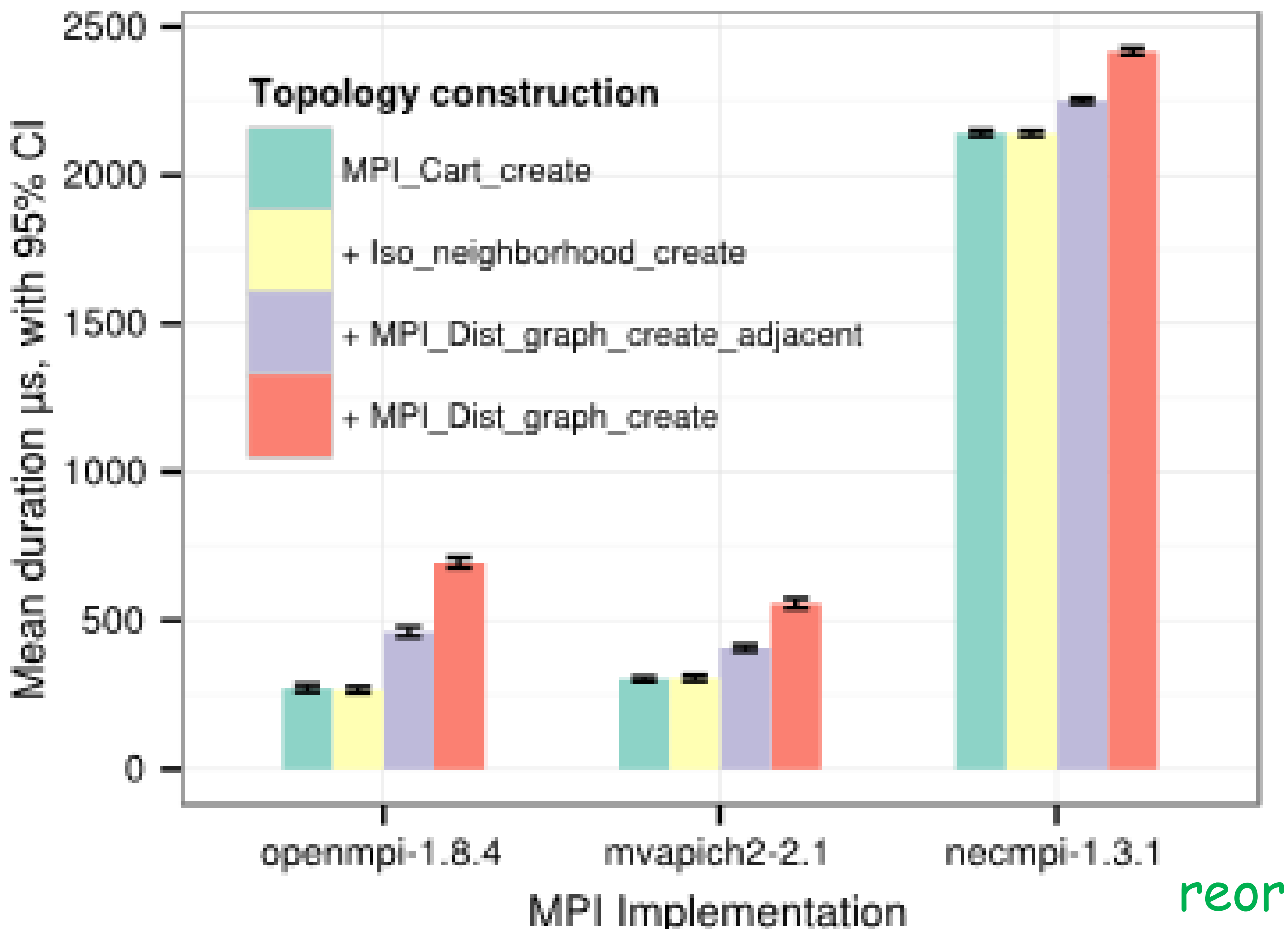
Setup times:

- MPI_Cart_create()
- MPI_Dist_graph_create_adjacent()

Attachment time:

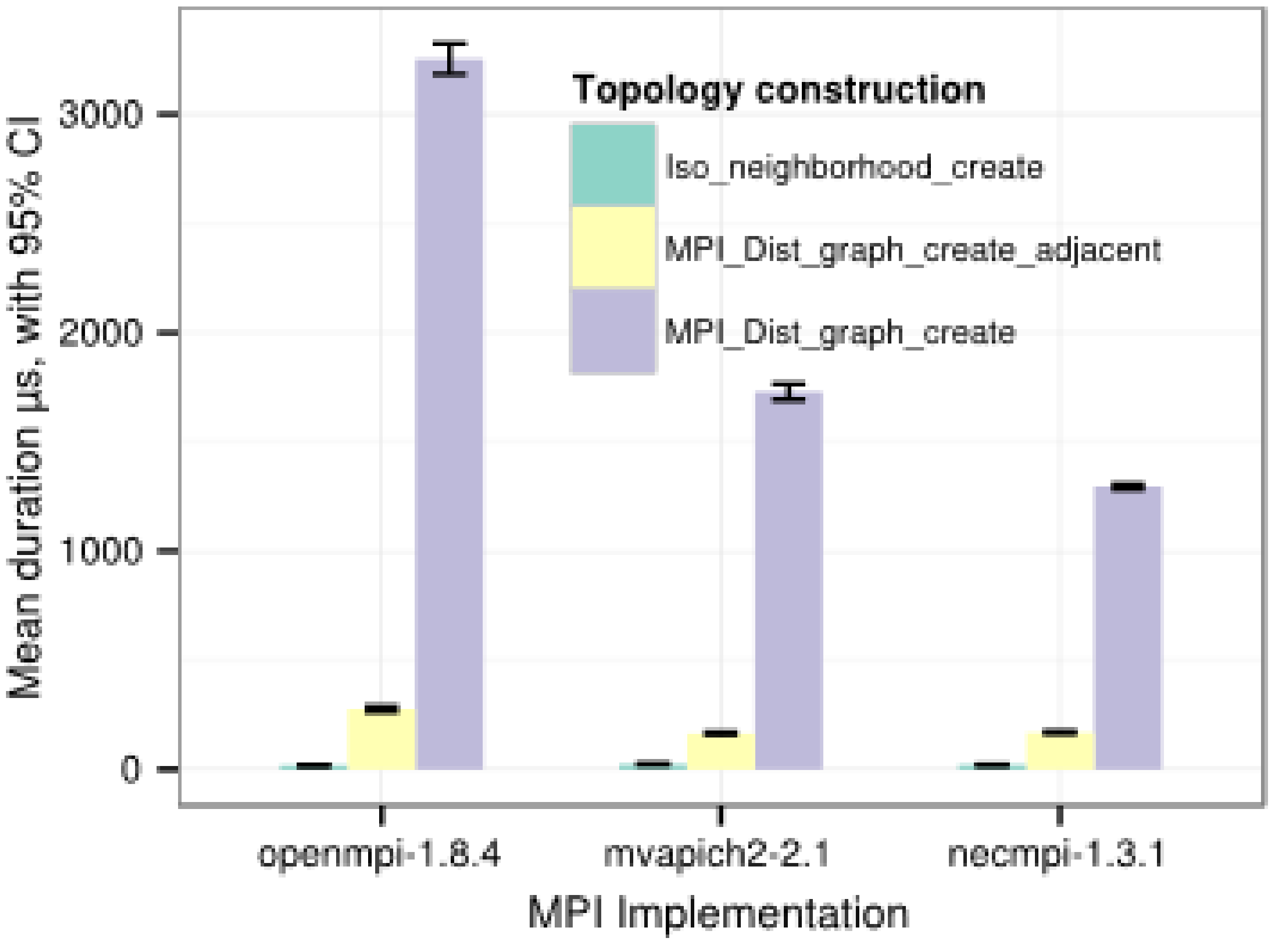
- Iso_neighborhood_create()

d=2, 24x20 virtual torus, r=1 von Neumann neighborhood



Note: reorder=true in MPI_Cart_create

d=2, 24x20 virtual torus, r=3 Moore neighborhood



Findings:

- MPI libraries differ

- Use MPI_Dist_graph_create_adjacent() where possible! Not MPI_Dist_graph_create()

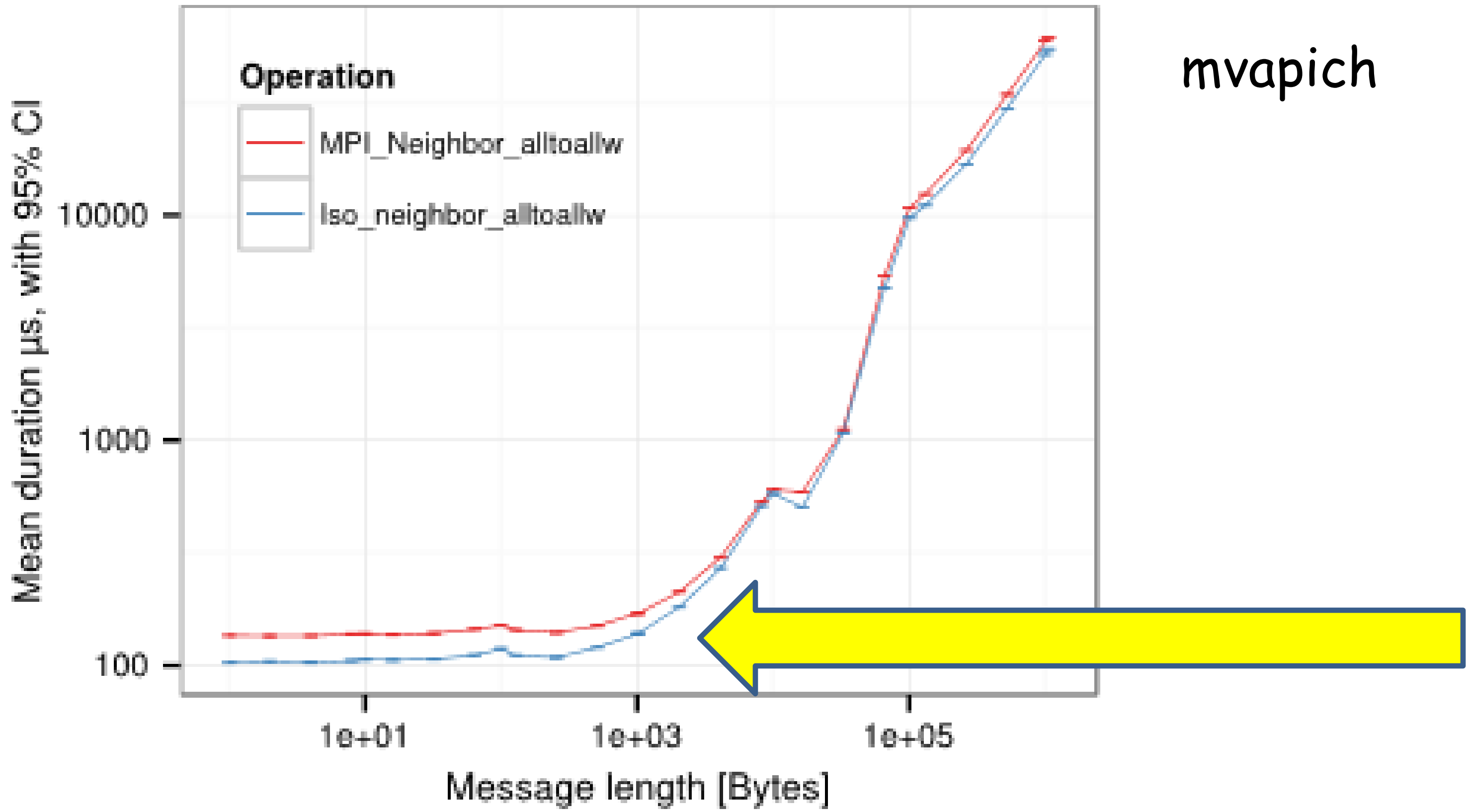
Sparse alltoall(w) communication times

Different neighborhoods:
•structure
•order

•Iso_neighbor_alltoallw() vs.
•MPI_Neighbor_alltoallw()

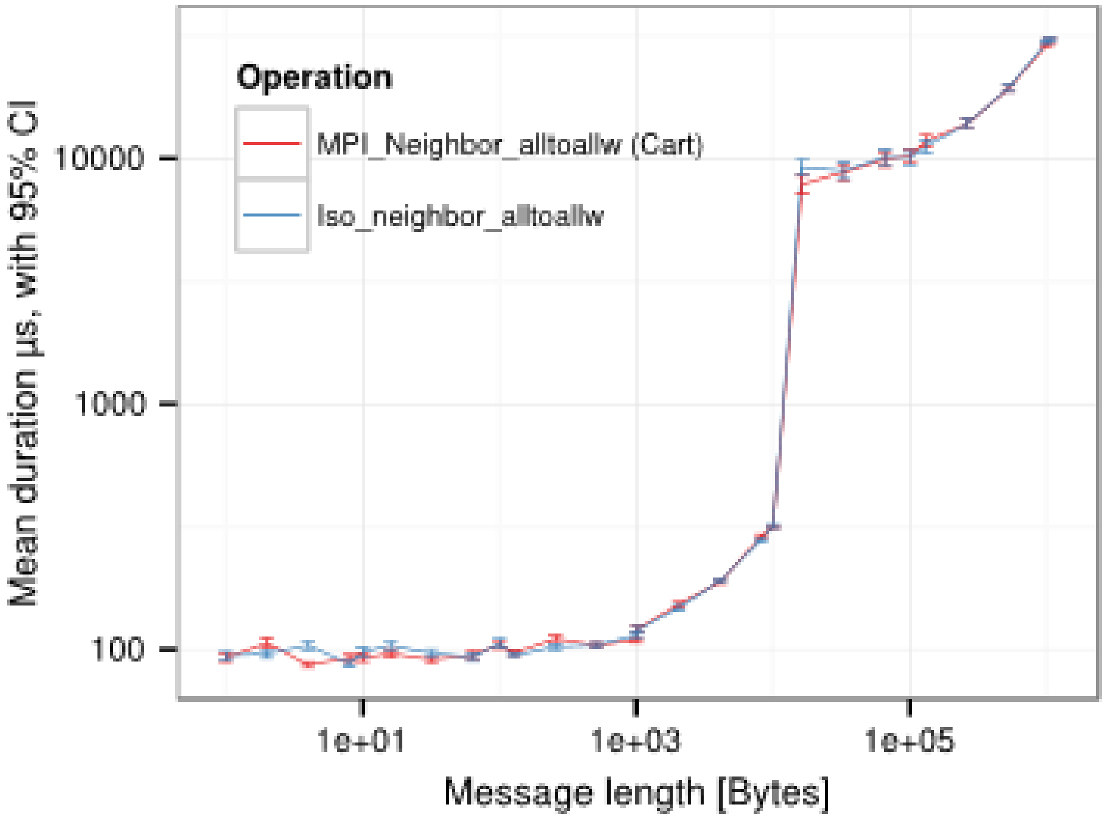
•Different MPI libraries

2=2, 6x5 virtual torus, r=3 Moore neighborhood



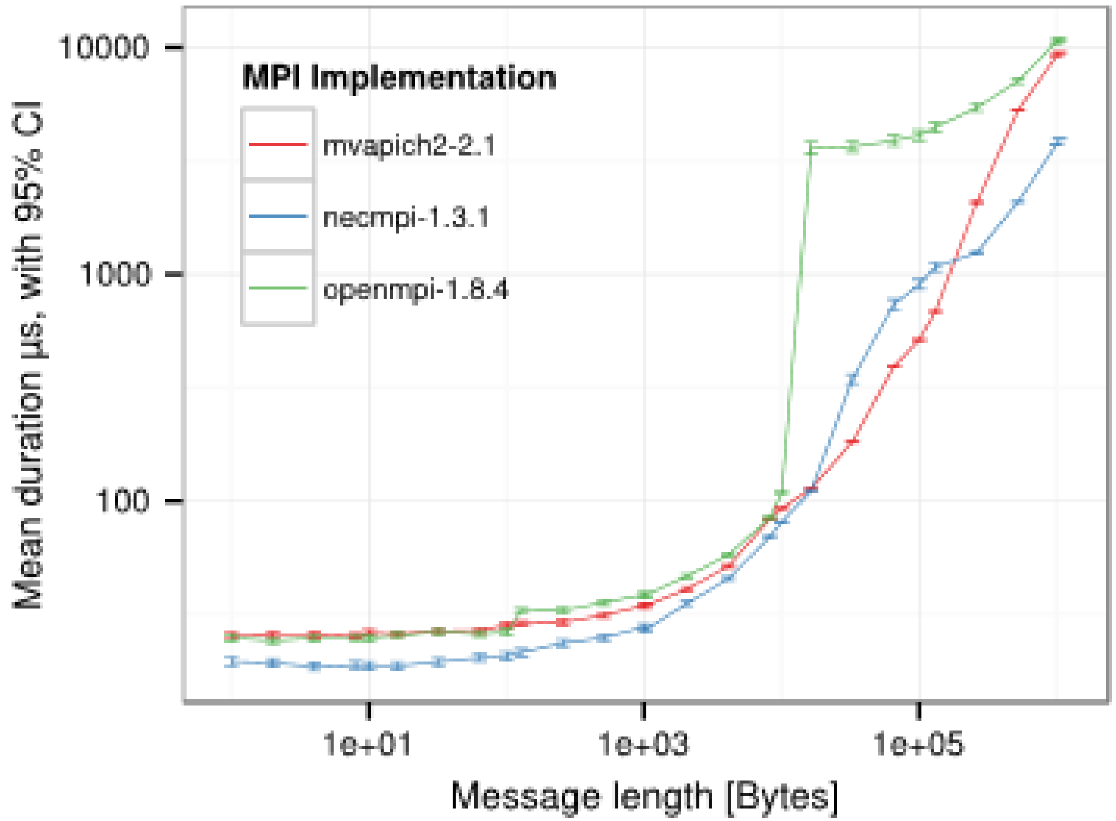
mvapich

d=2, 24x20 virtual torus, r=1 von Neumann neighborhood

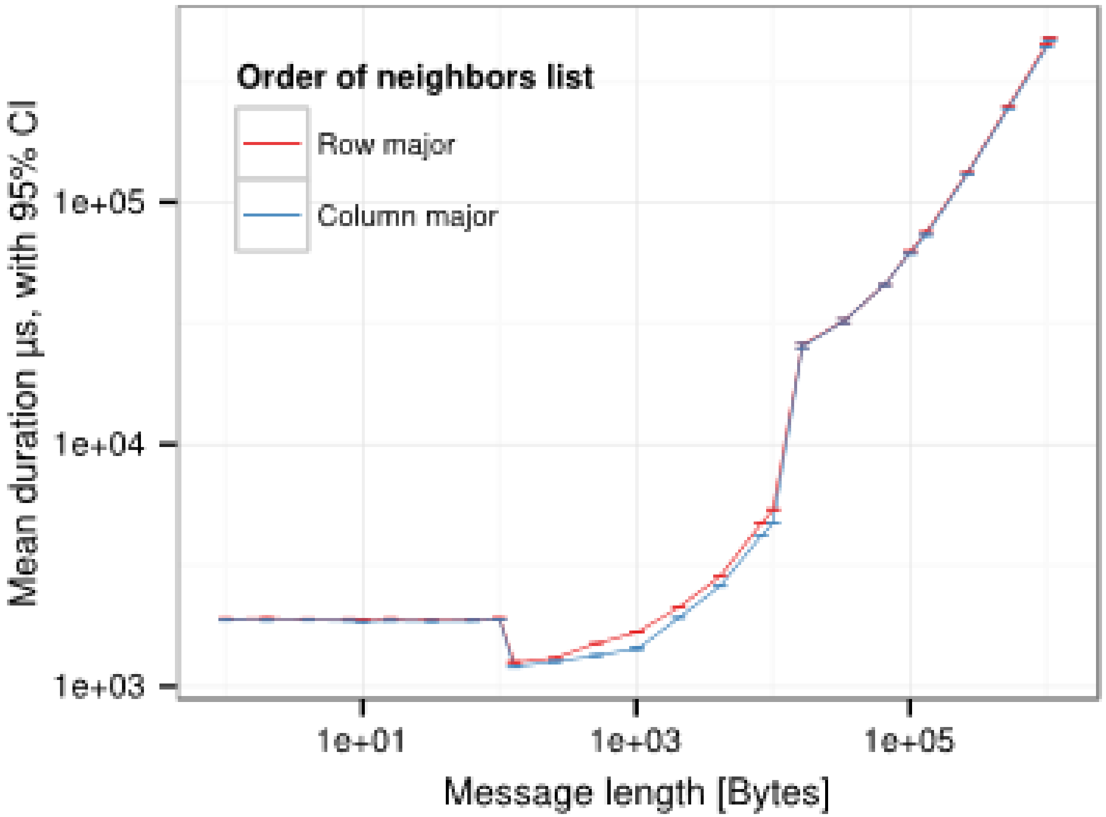


OpenMPI

d=2, 6x5 virtual torus, r=1 Moore neighborhood



d=2, 24x20 virtual mesh, r=3 Moore neighborhood



OpenMPI

Findings:

Preliminary: there are many, many combinations to try out. Automatic benchmark guided by performance guidelines could make more systematic assessment possible

- Basic communication performance between Iso_neighbor_alltoallw() and MPI_Neighbor_alltoallw() comparable. MPI libraries do not seem to have invested in non-trivial optimizations?

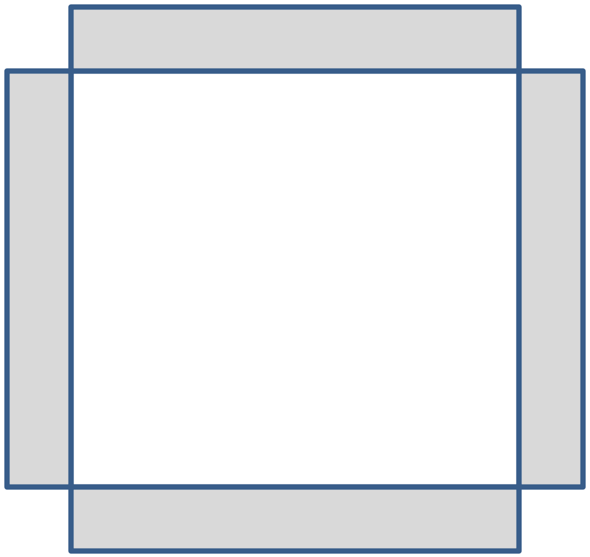
- Neighbor order does not seem to play a large role

- Sometimes notable differences between MPI libraries

Stencil computations

- 5-point (von Neumann) and 9-point (Moore) d=2 stencil computation with halo of depth k, k≥1
- Only communication here, no actual stencil update, 100 iterations of communication loop
- Matrix order n (per process)
- Total time, including time to set up neighborhood
- Iso_neighbor_alltoallw() vs. MPI_Neighbor_alltoallw()

5-point stencil, k=1 halo

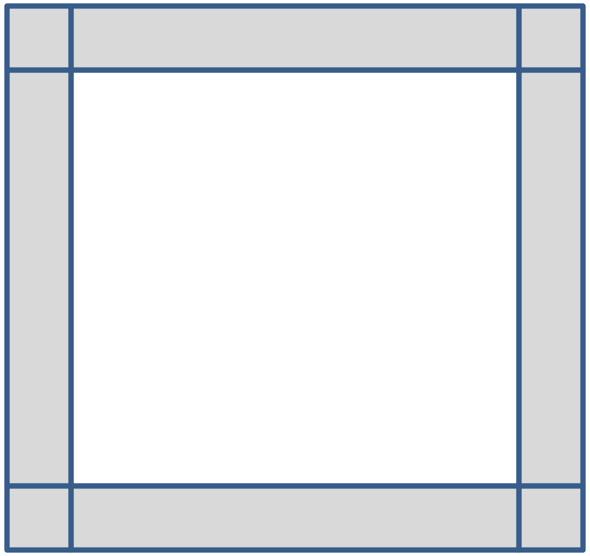


•Halo has same structure as stencil

•Different data layout in the two dimensions

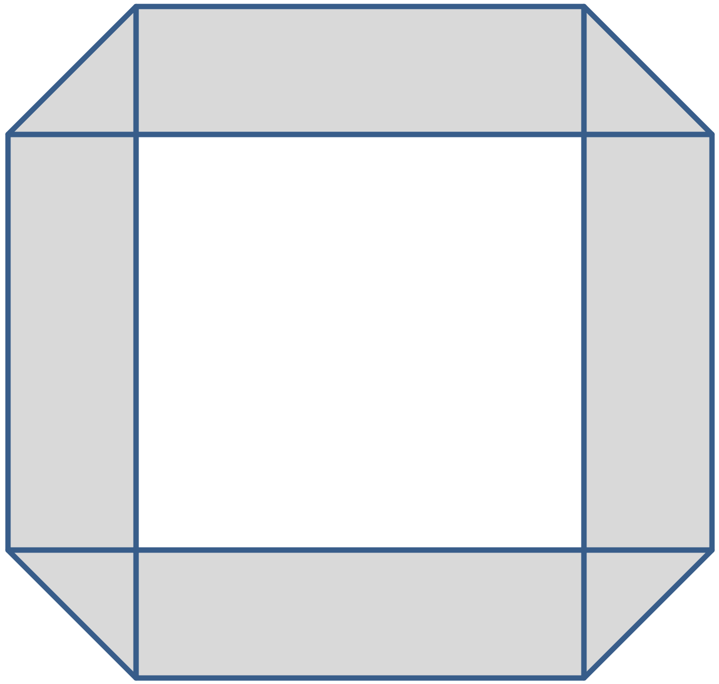
•Need for alltoallw-functionality: different datatypes for different dimensions

9-point stencil, k=1 halo



- Halo has same structure as stencil
- Different data layout in the two dimensions
- Need for alltoallw-functionality: different datatypes for different dimensions

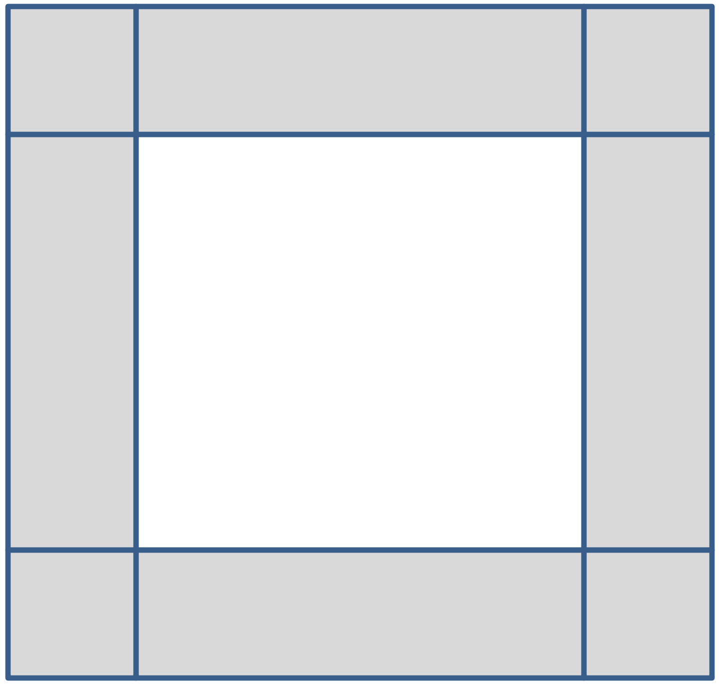
5-point stencil, k>1 halo



- Halo has different structure from stencil: Moore neighborhood needed, all 8 neighbors
- Different data layout in the two dimensions
- Need for alltoallw-functionality: different datatypes for different dimensions

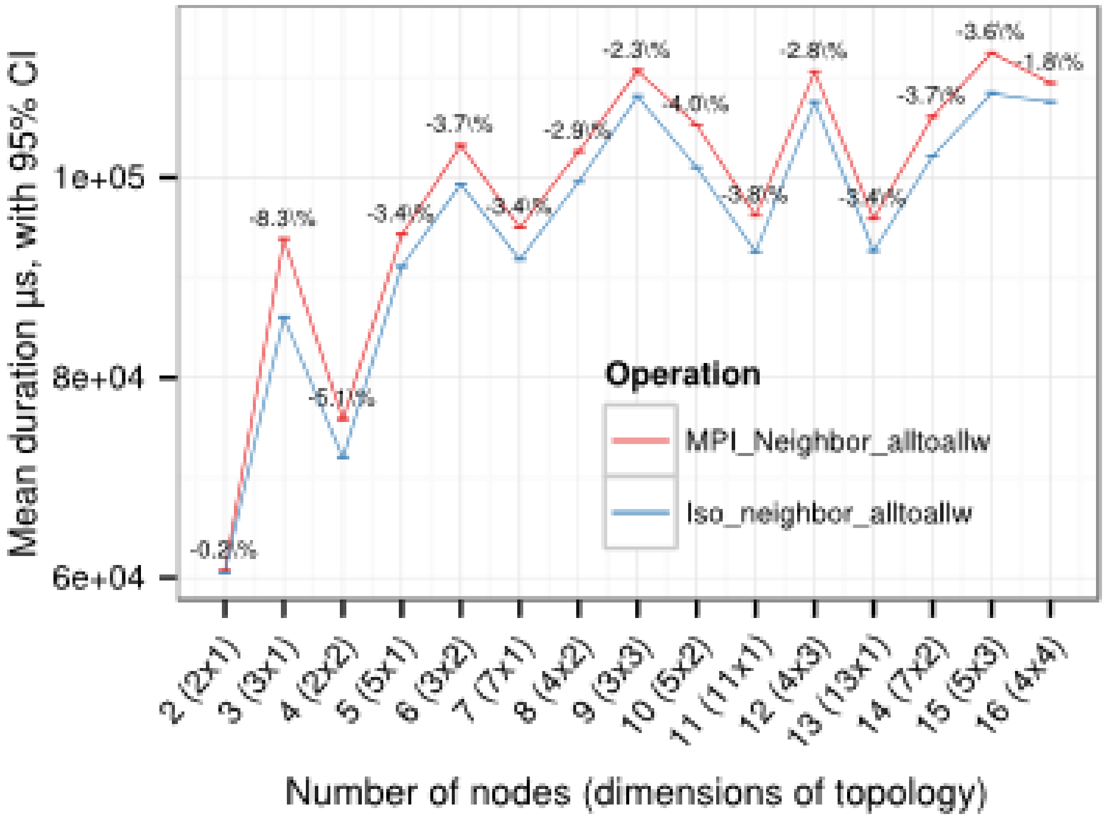
MPI Cartesian neighborhood does not suffice even for simple 5-point stencil with deeper halo

9-point stencil, k>1 halo



- Halo has same structure as stencil
- Different data layout in the two dimensions
- Need for alltoallw-functionality: different datatypes for different dimensions

10.000x10.000 matrix, 5-point stencil, halo depth k=10

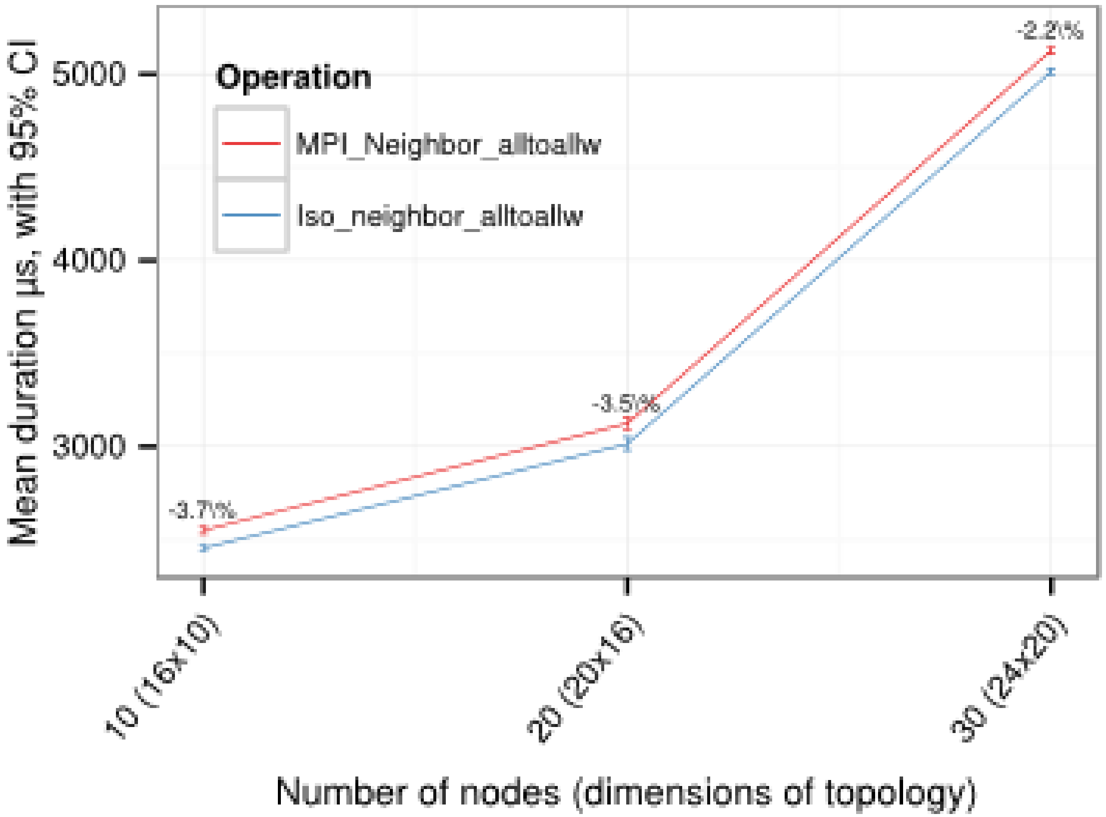


mvapich

First attempt: MPI_Dims_create() to factor p into “best” 2 dimension sizes.

MPI libraries differ badly, and sometimes fail miserably. See POSTER

100x100 matrix, 9-point stencil, halo depth k=2



necMPI

Findings:

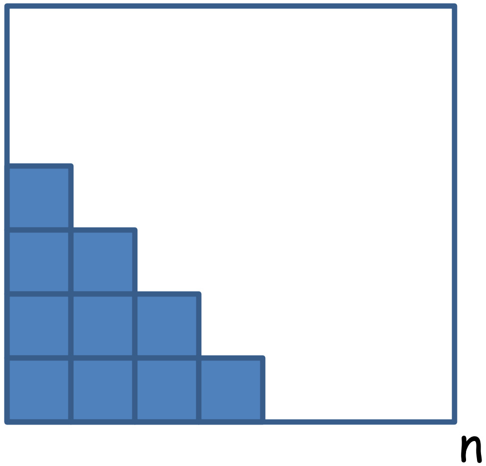
- Due to smaller set up time, Iso_neighbor_alltoallw() can be faster

- Problems with MPI_Dims_create()

Not really MPI function (no communicator): deprecate

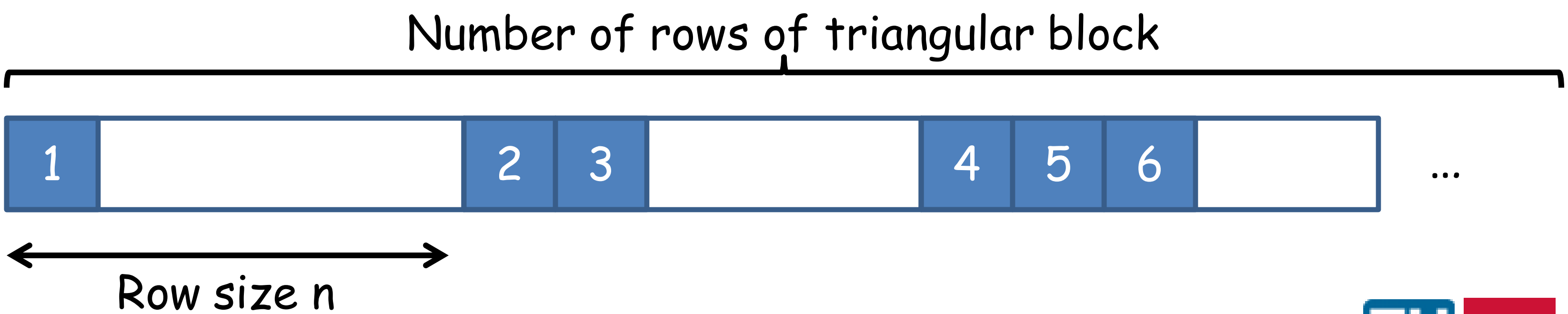
A useful datatype

Here:
Corners for 5-point stencil with halo k>1

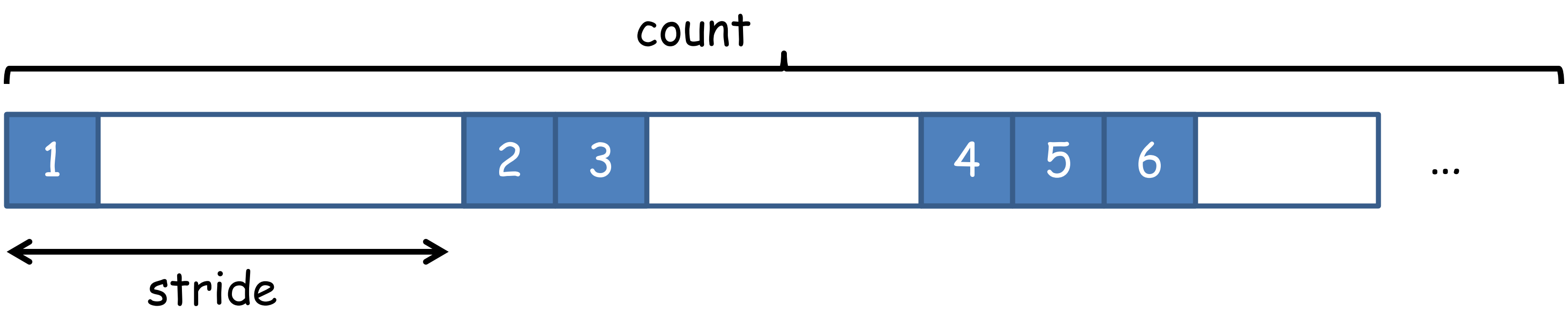


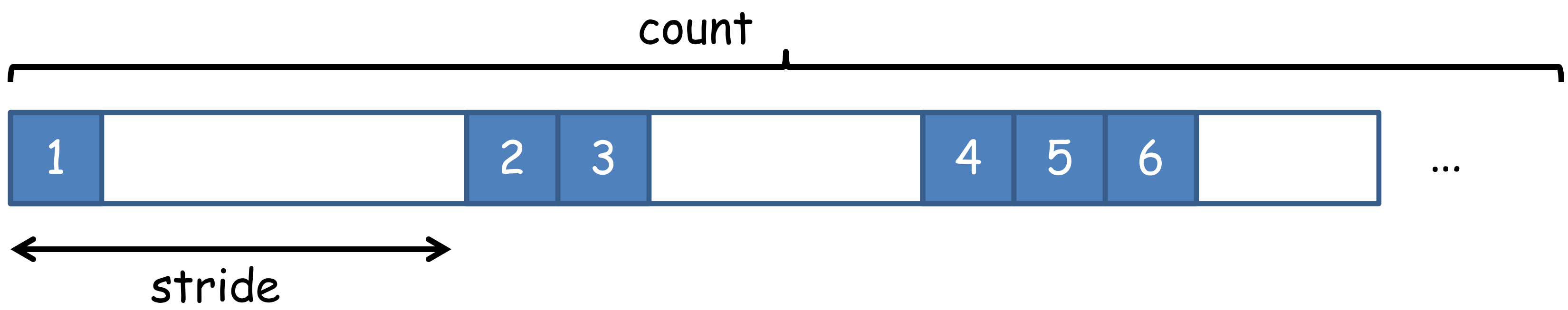
Common structure:

Triangular block of d=2 matrix



```
Create_triangular(count,
                  firstblock,blockincrement,
                  stride,strideincrement,
                  oldtype,*newtype);
```





Can trivially be implemented with MPI_Type_indexed() constructor, but

- Indexed requires storing and processing count, non-O(1) offsets

BUT:

Pattern is regular, can be represented with O(1) information, and presumably processed efficiently

Yesterday's talk

Strong (enough) case for considering additional datatype constructors for MPI?

Summary

- A more structured interface for stencil-like, sparse collective communication, easy to implement on top of MPI

- Isomorphic communication provides many possibilities for local schedule computations for efficient collective communication

Analogy with regular collectives: what can be asserted about global structure that will assist algorithms development?

- Some steps toward systematic benchmarking of neighborhood collectives

This work was supported by

EPiGRAM

FWF

Der Wissenschaftsfonds.