# Using Node Information to Implement MPI Cartesian Topologies

William Gropp
wgropp.cs.Illinois.edu

# MPI Virtual Process Topologies

- Lets user describe some common communication patterns
- Promises
  - Better performance (with “reorder” flag true)
  - Convenience in describing communication (at least with Cartesian process topologies)
- Reality
  - “Reorder” for performance rarely implemented
    - Few examples include NEC SX series and IBM BlueGene/L
  - Challenge to implement in general
    - Perfect mapping complex to achieve except in special cases
      - And perfect is only WRT the abstraction, not the real system
- How much is MPI_Cart_create used?

# MPI_Cart_create is rarely used by applications or optimized by implementations

| Benchmark | Codes | Uses Cart | With reorder |
|---|---|---|---|
| NAS PB 3.3.1 | | No | |
| HOMB 1.0 | | No | |
| POP 2.0 | | No | |
| HPCC 1.4.1 | | No | |
| CORAL | HACC | Yes | No |
| | nekbone | No | |
| | QMCPACK | No | |
| | Snap | Yes | Yes |
| SPP-2017 | PSDNS | Yes | No |
| | WRFV3 | Yes | No |
| LAMMPS 16 Mar 2018 | | Yes | Yes |

- Vendor MPI for tested systems does not remap in Cart_create
- Neither MPICH 3.2.1 nor Open MPI 3.1.0 remap in Cart_create
- Some systems have in the past
  - NEC SX
  - IBM BlueGene/L
- Focus has been on adapting to the network topology
  - Complex problem in general, even on mesh networks

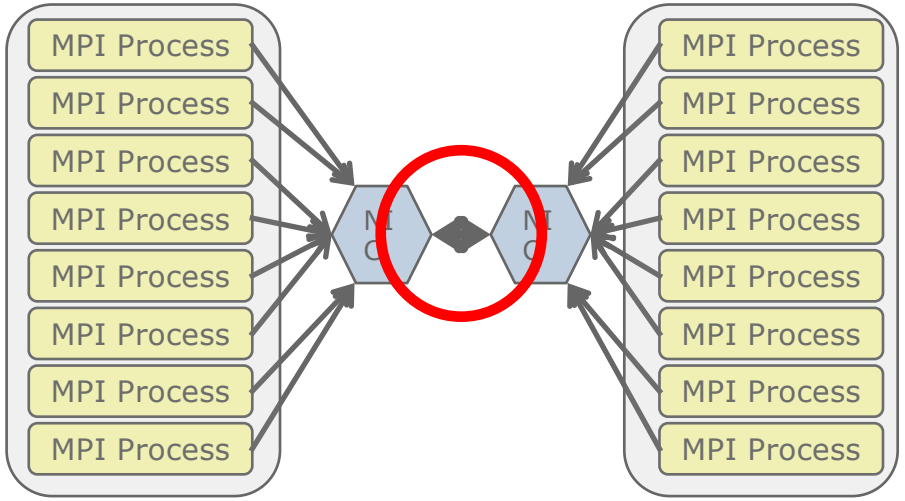
# Does Process Topology Matter?

- Example: comparison of halo exchanges for several systems and patterns
- Comparing Rates
  - Ratios of a single sender to all processes sending (in rate)
  - *Expect* a factor of roughly 2 (since processes must also receive). Larger factors are bad (using more processes takes more time for the same amount of data per process)
  - BG gives roughly double the halo rate. XTn and XE6 are much higher.
- Conventional explanation
  - Blue Gene has a simple interconnect topology with independent links in the Cartesian dimensions
  - Cray process mapping does not respect process topology; allocation of nodes not as clean or simple as on Blue Gene

| System | 4 Neighbors | | 8 Neighbors | |
|---|---|---|---|---|
| | | Periodic | | Periodic |
| BG/L | 2.24 | | 2.01 | |
| BG/P | 3.8 | | 2.2 | |
| BG/Q | | | 1.98 | |
| XT3 | 7.5 | 8.1 | 9.08 | 9.41 |
| XT4 | 10.7 | 10.7 | 13.0 | 13.7 |
| XE6 | | | 15.6 | 15.9 |

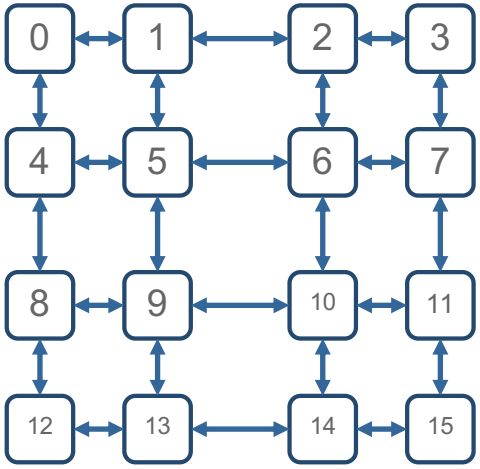
# Is There Another Explanation?

- Multicore nodes have high intranode bandwidth
  - Higher in aggregate than typical NIC
  - Replace the postal model with the max rate model
- Typical process mappings do not match the 2 level hierarchy of processes on nodes
  - This ignores further details, such as sockets on nodes

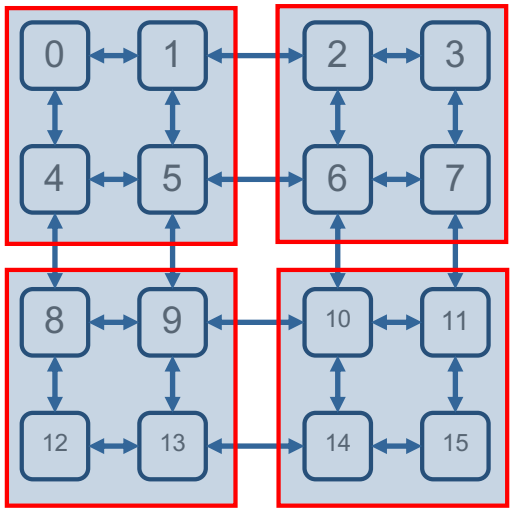



*Modeling MPI Communication Performance on SMP Nodes: Is it Time to Retire the Ping Pong Test*, W Gropp, L Olson, P Samfass, Proceedings of EuroMPI 16, https://doi.org/10.1145/2966884.2966919

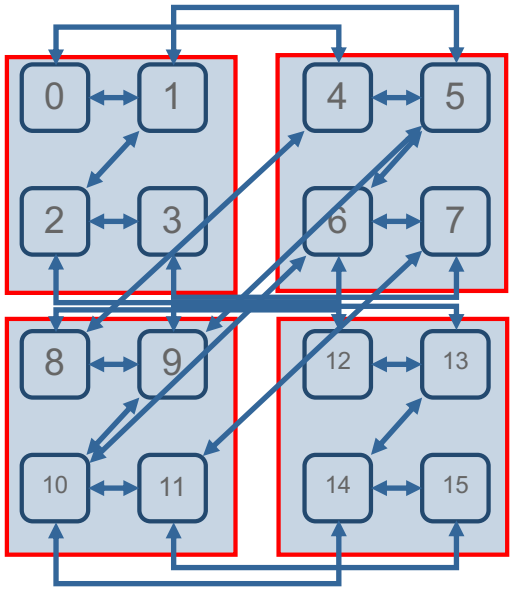
# Example Cartesian Process Mesh



# Example Cartesian Process Mesh – Four Nodes (Desired)



# Example Cartesian Process Mesh – Four Nodes (Typical process mapping)



# Can We Do Better?

- Hypothesis: A better process mapping **within** a node will provide significant benefits
  - **Ignore** the internode network topology
    - Vendors have argued that their network is fast enough that process mapping isn't necessary
    - They may be (almost) right – once data enters the network
- Idea for Cartesian Process Topologies
  - Identify nodes (see MPI_Comm_split_type)
  - Map processes *within* a node to minimize **inter**node communication
    - Trading **intra**node for **inter**node communication
  - Details follow…

# Algorithm

Input: oldcomm : communicator of processes
dims : dimensions of virtual Cartesian mesh
numdims : number of dimensions
Output: nodecartcomm : communicator with processes reordered in a Cartesian mesh
Local Variables:
nodecomm : communicator of processes on the same node
leadercomm : communicator of leaders on each node
intradims : Processor mesh for processes on a node
interdims : Processor mesh of the nodes (1 entry per node)
intracoords: Coordinate of this process on node in mesh of intradims
intercoords: Coordinate of this node in mesh of interdims
coords : Coordinate of this process in mesh (as reordered)

**Step 1: Find nodes**

```
MPI_Comm_split_type(oldcomm,
                    MPI_COMM_TYPE_SHARED, rank,
                    MPI_INFO_NULL, &nodecomm)
```

Step 2: Assemble comm of node leaders

```
MPI_Comm_rank(nodecomm, &nrank)
color = MPI_UNDEFINED
if (nrank == 0) color = 0
MPI_Comm_split(oldcomm, color, oldrank,
               &leadercomm)
```

Step 3: Inform all processes of the number of nodes

```
if (color == 0) MPI_Comm_size(leadercomm, &nnodes)
MPI_Bcast(&nnodes, 1, MPI_INT, 0, nodecomm)
```

# Algorithm

Input: oldcomm : communicator of processes
dims : dimensions of virtual Cartesian mesh
numdims : number of dimensions
Output: nodecartcomm : communicator with processes reordered in a Cartesian mesh
Local Variables:
nodecomm : communicator of processes on the same node
leadercomm : communicator of leaders on each node
intradims : Processor mesh for processes on a node
interdims : Processor mesh of the nodes (1 entry per node)
intracoords: Coordinate of this process on node in mesh of intradims
intercoords: Coordinate of this node in mesh of interdims
coords : Coordinate of this process in mesh (as reordered)

Step 1: Find nodes

```
MPI_Comm_split_type(oldcomm,
                    MPI_COMM_TYPE_SHARED, rank,
                    MPI_INFO_NULL, &nodecomm)
```

**Step 2: Assemble comm of node leaders**

```
MPI_Comm_rank(nodecomm, &nrank)
color = MPI_UNDEFINED
if (nrank == 0) color = 0
MPI_Comm_split(oldcomm, color, oldrank,
               &leadercomm)
```

Step 3: Inform all processes of the number of nodes

```
if (color == 0) MPI_Comm_size(leadercomm, &nnodes)
MPI_Bcast(&nnodes, 1, MPI_INT, 0, nodecomm)
```

# Algorithm

Input: oldcomm : communicator of processes
dims : dimensions of virtual Cartesian mesh
numdims : number of dimensions
Output: nodecartcomm : communicator with processes reordered in a Cartesian mesh
Local Variables:
nodecomm : communicator of processes on the same node
leadercomm : communicator of leaders on each node
intradims : Processor mesh for processes on a node
interdims : Processor mesh of the nodes (1 entry per node)
intracoords: Coordinate of this process on node in mesh of intradims
intercoords: Coordinate of this node in mesh of interdims
coords : Coordinate of this process in mesh (as reordered)

Step 1: Find nodes

```
MPI_Comm_split_type(oldcomm,
                    MPI_COMM_TYPE_SHARED, rank,
                    MPI_INFO_NULL, &nodecomm)
```

Step 2: Assemble comm of node leaders

```
MPI_Comm_rank(nodecomm, &nrank)
color = MPI_UNDEFINED
if (nrank == 0) color = 0
MPI_Comm_split(oldcomm, color, oldrank,
               &leadercomm)
```

**Step 3: Inform all processes of the number of nodes**

```
if (color == 0) MPI_Comm_size(leadercomm, &nnodes)
MPI_Bcast(&nnodes, 1, MPI_INT, 0, nodecomm)
```

# Algorithm con't

**Step 4: Find 2-level decomposition of dimensions**

MPI_Comm_size(nodecomm, &nsize)

Confirm that nsize is the same for all processes in oldcomm (required for algorithm)

Distribute factors of nodesize to create intradims, attempting to make intradims "square", with

intradims[i] * interdims[i] == dims[i],

and such that sum(interdims) is small

**(see later slide for details of this step)**

Step 5: Find coordinates in virtual mesh

From rank in leadercomm, compute coordinates of node in mesh of size interdims

From rank in nodecomm, compute coordinates of process in mesh of size intradims

```
coords[i] = intracoords[i] +
            intercoords[i] * intradims[i]
```

Step 6: Find rank of process in output communicator based on coordinates in row-major order, and create communicator

```
rr = coords[0]
for (i=1; i<numdims; i++)
    rr = rr * dims[i] + coords[i]
MPI_Comm_split(oldcomm, 0, rr,
               &nodecartcomm)
```

# Algorithm con't

Step 4: Find 2-level decomposition of dimensions

MPI_Comm_size(nodecomm, &nsize)

Confirm that nsize is the same for all processes in oldcomm (required for algorithm)

Distribute factors of nodesize to create intradims, attempting to make intradims "square", with

intradims[i] * interdims[i] == dims[i],

and such that sum(interdims) is small

**Step 5: Find coordinates in virtual mesh**

**From rank in leadercomm, compute coordinates of node in mesh of size interdims**

**From rank in nodecomm, compute coordinates of process in mesh of size intradims**

**coords[i] = intracoords[i] +**

**intercoords[i] * intradims[i]**

Step 6: Find rank of process in output communicator based on coordinates in row-major order, and create communicator

```
rr = coords[0]
for (i=1; i<numdims; i++)
    rr = rr * dims[i] + coords[i]
MPI_Comm_split(oldcomm, 0, rr,
        &nodecartcomm)
```

# Algorithm con't

Step 4: Find 2-level decomposition of dimensions

MPI_Comm_size(nodecomm, &nsize)

Confirm that nsize is the same for all processes in oldcomm (required for algorithm)

Distribute factors of nodesize to create intradims, attempting to make intradims "square", with

intradims[i] * interdims[i] == dims[i],

and such that sum(interdims) is small

Step 5: Find coordinates in virtual mesh

From rank in leadercomm, compute coordinates of node in mesh of size interdims

From rank in nodecomm, compute coordinates of process in mesh of size intradims

coords[i] = intracoords[i] +

intercoords[i] * intradims[i]

**Step 6: Find rank of process in output communicator based on coordinates in row-major order, and create communicator**

```
rr = coords[0]
for (i=1; i<numdims; i++)
    rr = rr * dims[i] + coords[i]
MPI_Comm_split(oldcomm, 0, rr,
        &nodecartcomm)
```

# Finding a “good” intradims

- Try to make the intradims “square”, subject to the constraints established by dims[]
- Note that dims can make a good choice for intradims impossible
  - MPI_Dims_create is not, in fact, helpful
  - Replacements are needed that take into account the specific communicator

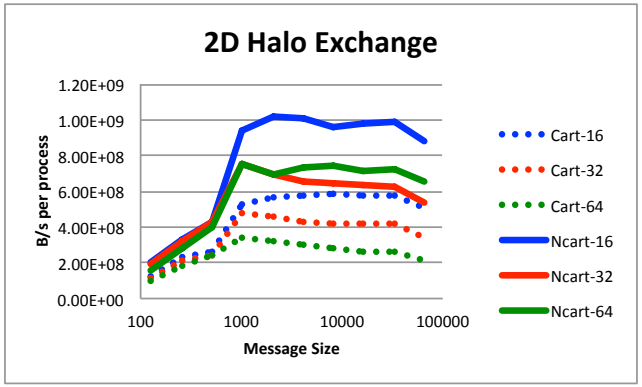
Step 1: Factor node size into primes
Step 2: Set intradims[i] to 1 and
    remaindims[i] to dims[i]
Step 3: Take the largest prime factor fac
    find j such that
        fac divides remaindims[j] and
        remaindims[j] is larger than
        remaindims[k] for any k such
        that fac also divides remaindims[k]
Step 4: Set intradims[j] to intradims[j]*fac and
    remainsdims[j] to remaindims[j]/fac
Step 5: Repeat from Step 3 until all factors are
    distributed

# Testing the Hypothesis: The Systems

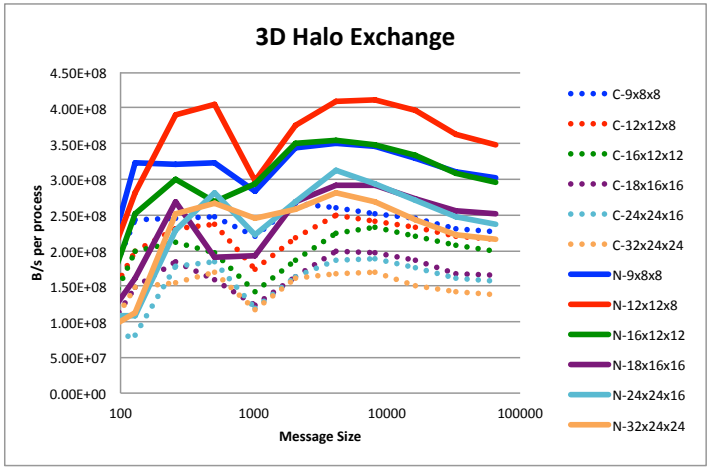
- Blue Waters at Illinois
  - Cray XE6/XK7
  - 3D mesh (Gemini); service nodes embedded in mesh
  - 22,636 XE6 nodes, each with 2 AMD Interlagos (and 4228 XK7 nodes)
- Theta at Argonne
  - Cray XC40
  - Dragonfly (Aires) interconnect
  - 4392 Intel KNL nodes
- Piz Daint at Swiss National Supercomputing Center
  - Cray XC50/XC40
  - Dragonfly (Aires) interconnect
  - 5320 XC50 and 1813 XC40 nodes

# Comparing 2D Halo Exchanges

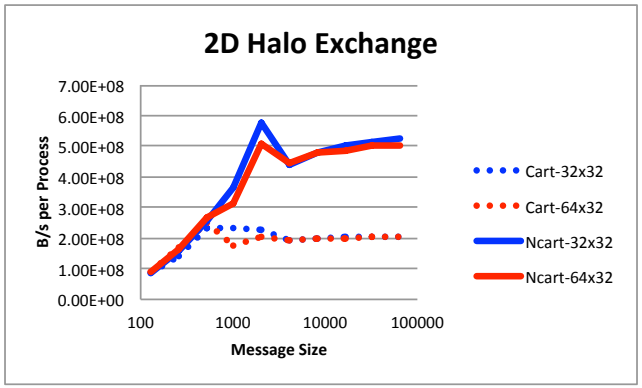
Blue Waters




Piz Daint

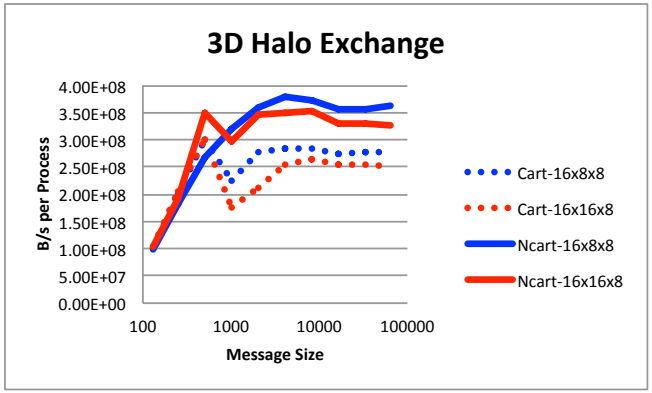



Theta

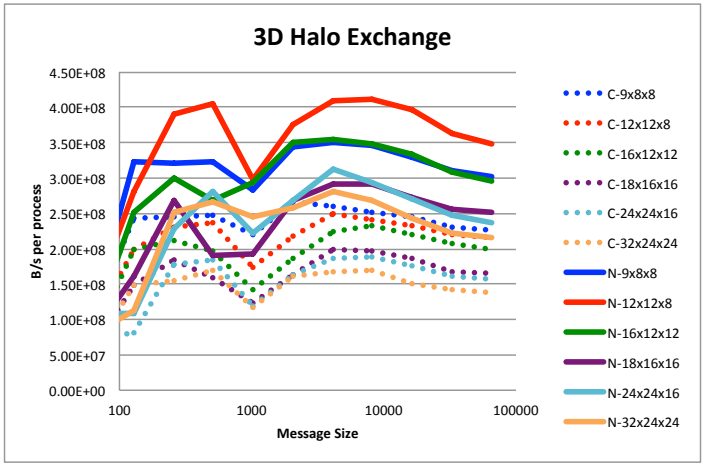


# Comparing 3D Halo Exchanges

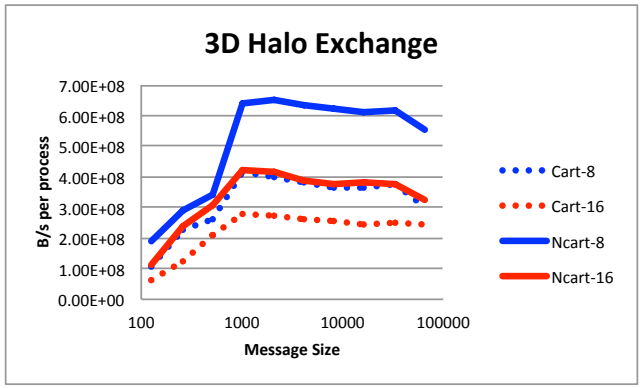
Blue Waters




Piz Daint




Theta



# Comparing On- and Off-node Communication

- Number of intranode (on-node) and internode (off-node) communication partners *per process*
- 16 processes per node
- Size is Cartesian virtual process topology
- “Nodecart” mapping is significantly better
  - In terms of reducing off-node communication in favor of on-node communication

| | | | On-node | | | Off-node | | |
|---|---|---|---|---|---|---|---|---|
| Type | Dim | Size | Min | Max | Avg | Min | Max | Avg |
| Cart | 2 | 128x128 | 1 | 2 | 1.88 | 2 | 3 | 2.12 |
| Nodecart | 2 | 128x128 | 2 | 4 | 3 | 0 | 2 | 1 |
| Cart | 3 | 32x32x16 | 1 | 2 | 1.88 | 4 | 5 | 4.12 |
| Nodecart | 3 | 32x32x16 | 3 | 4 | 3.5 | 2 | 3 | 2.5 |

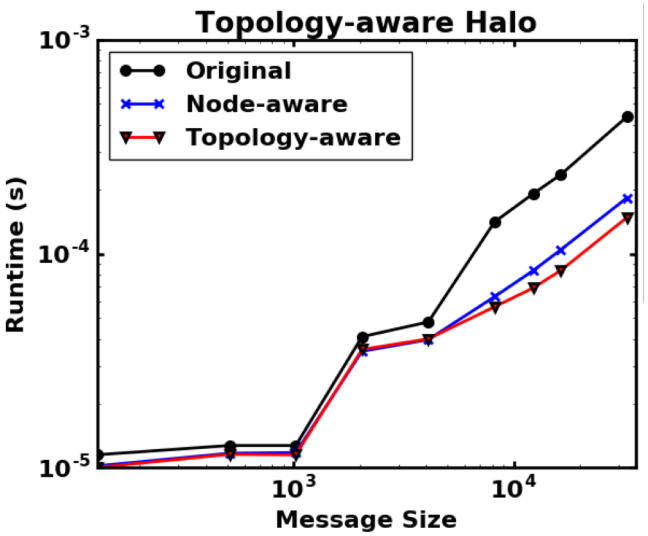
# Comparing Performance in A Stencil Sweep

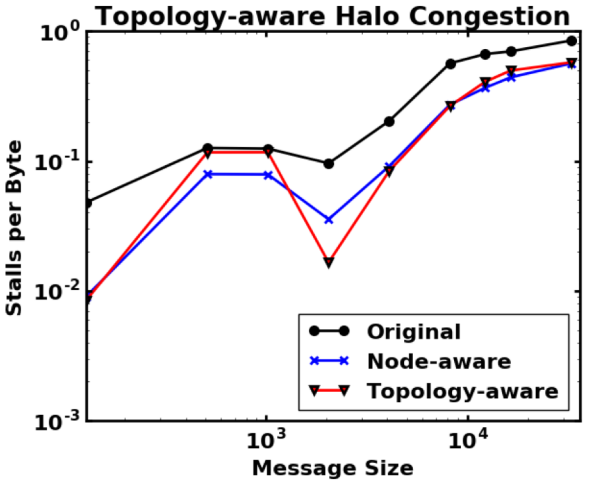
- 2D Stencil (5-point operator) sweep on 4096 processors
  - 256 nodes of 16 processes
  - Code used in “Advanced MPI” tutorial at SC
  - Based on code from Torsten Hoefler
- Rates in GFLOP/s
- NxN global mesh
- “Nodecart” shows modest but consistent advantage

| N | Cart | Nodecart | Ratio |
|---|---|---|---|
| 256 | 30.1 | 32.4 | 1.08 |
| 512 | 118 | 121 | 1.03 |
| 1024 | 432 | 491 | 1.14 |
| 2048 | 1550 | 1670 | 1.08 |
| 4096 | 3170 | 3370 | 1.11 |
| 8192 | 4400 | 4710 | 1.07 |
| 16384 | 5960 | 6170 | 1.04 |

# How Important is Network Topology?

- No answer yet, but…
- 432 nodes, 3D halo exchange on Blue Waters
  - Requested a cube of nodes, used non-standard routines to implement mapping for network topology
- Part of study into scalable Krylov methods (looking to avoid the blocking MPI_Allreduce)
- Nodecart version provides most of the benefit with no need for network topology information
- Some (nontrivial) further benefit possible by taking network topology into account
- But the largest contribution comes from node-awareness
- Thanks to Paul Eller for these results



# Comparison with Process Mapping

- An alternative is to use a tool to change the mapping of processes in MPI_COMM_WORLD to processors
- 3D halo exchange on 1024 processes and 64 nodes of Blue Waters
- Results are MB/s per process
- Used Cray pat_build and pat_report tools, with MPICH_RANK_REORDER_METHOD=3
- Columns are correct – default is better than rank mapped
  - Investigation showed rank mapped had few intranode and many internode communications

| | Rank Mapped | | Default | |
|---|---|---|---|---|
| N | World | Nodecart | World | Nodecart |
| 128 | 45 | 136 | 98 | 138 |
| 256 | 132 | 242 | 224 | 266 |
| 512 | 152 | 294 | 260 | 318 |
| 1024 | 205 | 487 | 489 | 616 |
| 2048 | 193 | 486 | 483 | 616 |
| 4096 | 185 | 483 | 477 | 606 |
| 16384 | 172 | 449 | 456 | 579 |
| 32768 | 168 | 442 | 461 | 580 |
| 65536 | 148 | 378 | 398 | 508 |

# Conclusion

- Using only node information provides a simple, fast, and effective implementation of MPI_Cart_create
- All MPI implementations should do at least this
- Need a better MPI_Dims_create
- Future Work
  - Implementation can be improved, e.g., add socket info (this work is in progress)
  - Implementation can be extended to MPI_Dist_graph_create, but harder (MPI _Cart_create exploits implicit information about location)
    - Could (should?) use existing graph partitioners without worrying about network topology
- Best is the enemy of (the) good
  - To allow the demand, desire, or insistence for perfection decreases the chances of obtaining a good or favorable result in the end. (https://idioms.thefreedictionary.com/best+is+the+enemy+of+the+good )
- Fortune cookie (Sept 14, 2018): “Many receive advice, only the wise profit by it”

# Thanks!


- This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07–25070) and the state of Illinois.
- Rajeev Thakur for runs on Theta
- Torsten Hoefler and Timo Schneider for runs on the Piz Daint
- Reviewers for their careful reading and helpful suggestions