THE SUPERCOMPUTER COMPANY

Obtaining Better Communication Performance and Scaling on Blue Waters: Topology Considerations

December 4, 2013

R. Fiedler
PRAC Applications Analyst

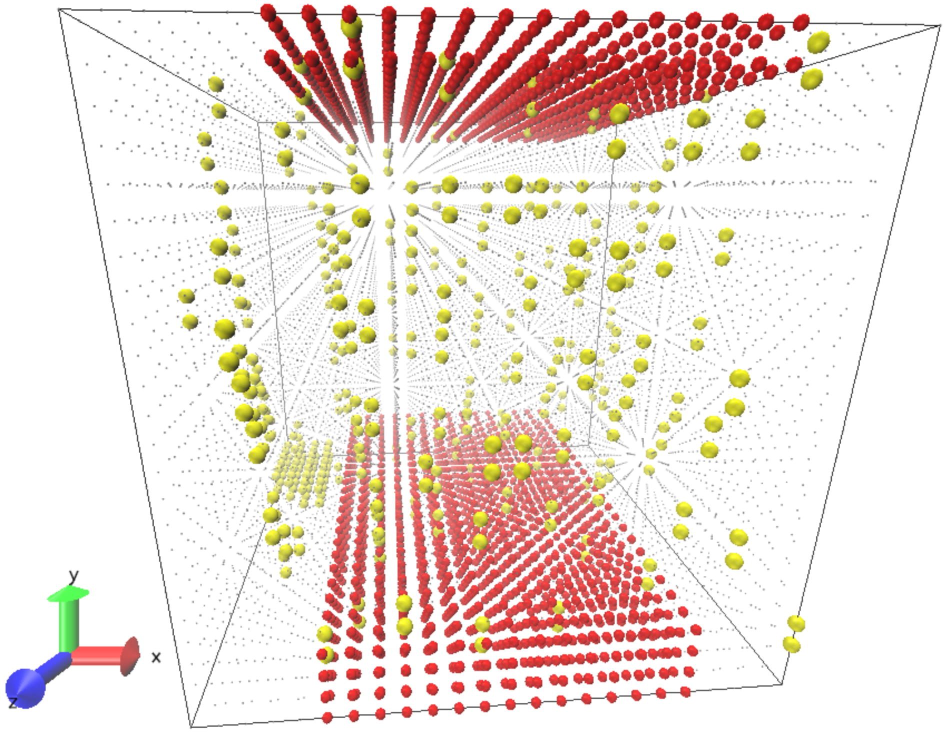
Overview

- Background on gemini interconnect
- Bisection bandwidth
- Job-job interference
- Node allocations
- Rank reordering
- Topology-aware applications
- Node selection and task placement

Background

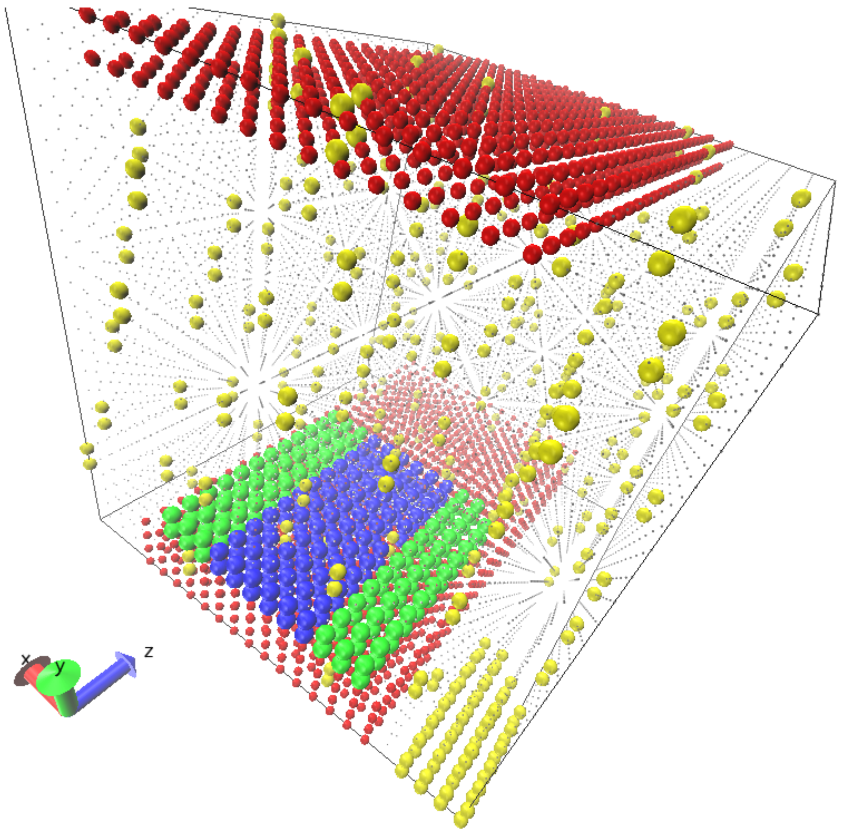
Blue Waters Interconnect

- Topology is 24x24x24 gemini routers
- 2 nodes per gemini, 2 geminis per blade
- 15x6x24 XK geminis (red)
- Service blades randomly distributed (yellow)
- Y-links between blades have 1/2 bandwidth of X- or Z-links
  - 2 nodes on same gemini don't use interconnect to exchange messages
- Routing algorithm is X, then Y, then Z



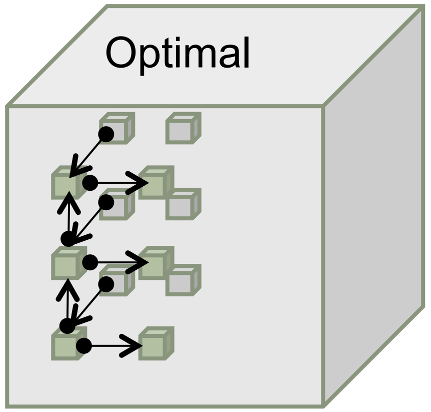
Background

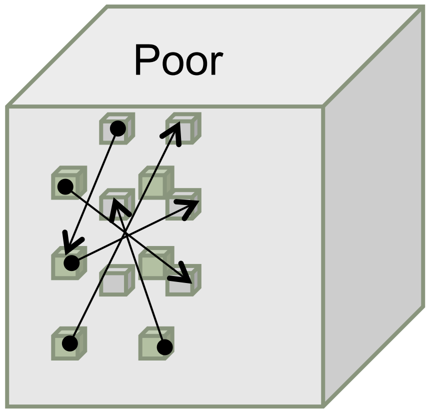
- Routing takes shortest path
- If path spans > 1/2 of nodes in given dimension, some communication may wrap around torus through nodes not assigned to job
- Jobs share interconnect for application communication, IO
- Run times affected by task placement, other running jobs
    - Figure: job on green geminis passes messages through blue geminis



Task Placement and Interference

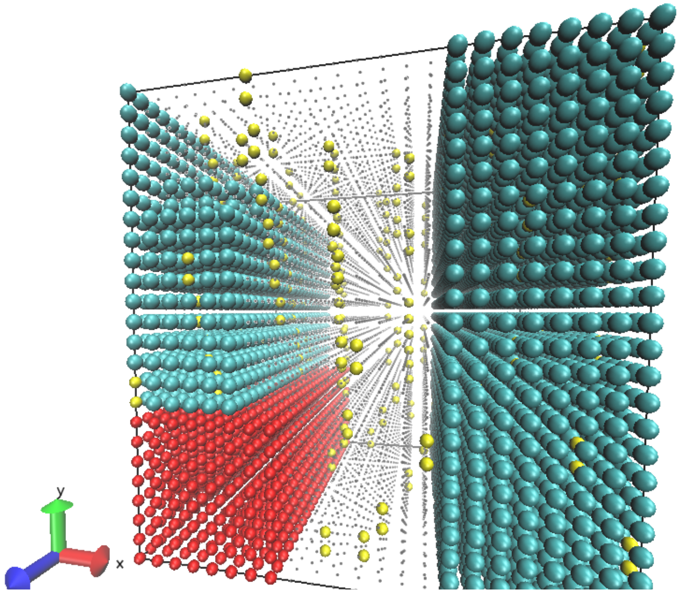
- **Applications that perform more communication are often more sensitive to placement and interference**
  - Applications with All-to-All communication patterns tend to compete more with other jobs
  - Such applications can benefit significantly from topology-aware node selection
- **Applications with only nearest-neighbor communication in their virtual topology, if poorly placed, actually perform pairwise communication between randomly located nodes**
  - "Random pairs" is like 1 stage of an All-to-All
  - Thus, analysis below of bisection bandwidth for All-to-All is relevant to many types of applications

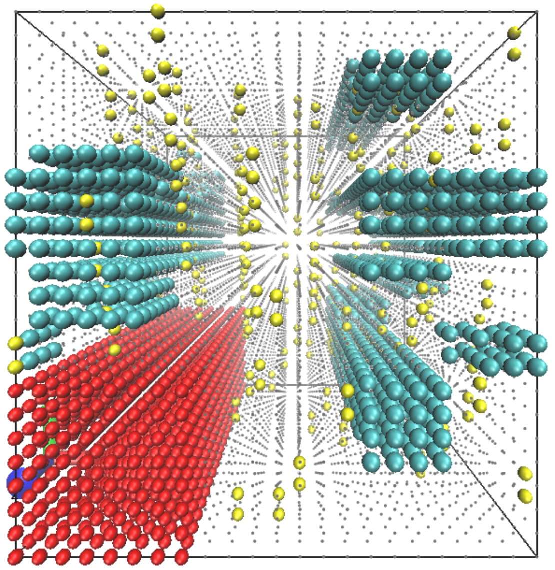




Node Allocations: ALPS & Job Scheduler

- **ALPS provides the scheduler (Torque/Moab) with list of compute nodes in specific order:**
  - 2x2x2 gemini blocks in space-fulling curve
  - Favors YZ slabs
- **Over time, after many jobs start & end, list of available nodes becomes increasingly fragmented**
  - Less compact allocations
  - Longer communication paths
  - More job-job interference
  - Less consistent, longer run times
- **Significant Cray/Adaptive/NCSA efforts underway to provide more compact allocations**
  - Favor XZ slabs & regular prisms
  - Request specified shapes

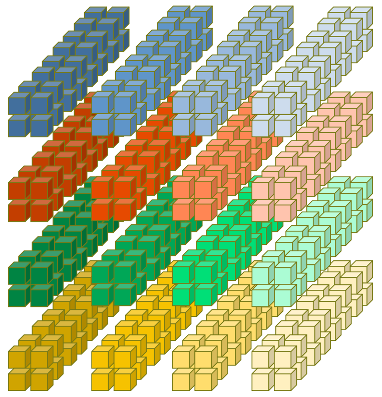


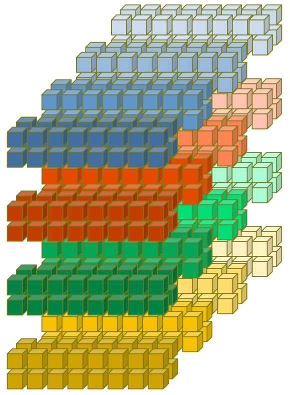


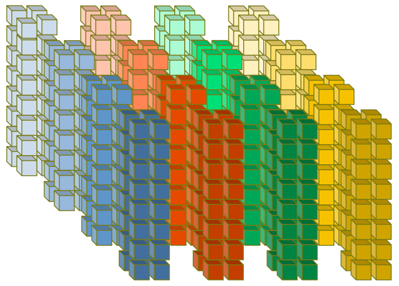
Example: PSDNS Turbulence Application

CFD Using Pseudo-Spectral Method

- Uses 3D FFTs of fluid variables to compute spatial derivatives
- Implementation uses 2D pencil decomposition
- For 3D FFT, must transpose full 3D arrays twice:
  - Begin with partitions spanning domain in x
  - 1D FFTs along x
  - Transpose within xy planes so each partition spans domain in y
  - 1D FFTs along y
  - Transpose within yz planes so each partition spans domain in z
  - 1D FFTs along z
- After some calculations requiring no communication, inverse 3D FFTs are performed in similar fashion
  - Dozens of forward and inverse 3D FFTs per time step
- Transposes comprise 50-75% of run time
  - Compute time includes local field variable updates, packing/unpacking communication buffers, 1D FFTs







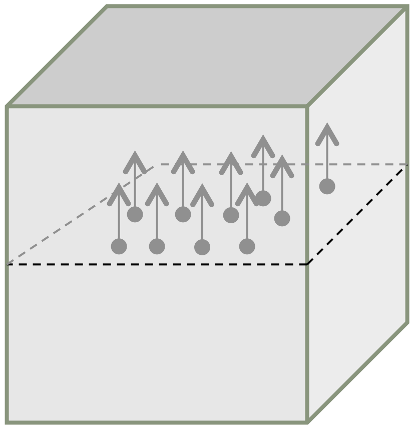
PSDNS Simple Performance Model

For N^3 grid points and P tasks

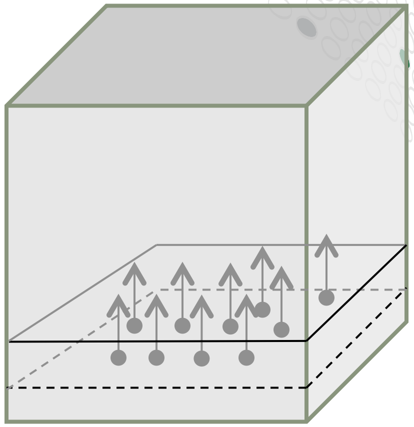
- **Computation time ~ N^3 * (const. + log N)**
- **Communication time ~ All-to-All time**
  - All-to-All time ~ Data volume/bisection bandwidth
    ~ N^3/bisection bandwidth
- **For naïve weak scaling experiments, N^3/P is held constant**
  - Computation time grows slowly with P
  - Communication time ~ P/bisection bandwidth
- **Thus, near-ideal weak scaling requires bisection bandwidth ~ P, or constant bisection bandwidth/ node**
- **Minimizing time to solution means maximizing bisection bandwidth per node**

Bisection Bandwidth: Full System

- **Bisection bandwidth of nodes in use determines run time for All-to-All**
- **Bisection bandwidth is defined as lowest bandwidth through any cross-sectional area**
  - BW topology is 24x24x24 geminis
  - Bisection bandwidth through cross section:
    - Normal to X: 24x24*X-link-bw*2 for torus
    - Normal to Y: 24x24*Y-link-bw*2 for torus
    - Normal to Z: 24x24*Z-link-bw*2 for tours
  - Y-link bandwidth ~ 1/2 X-link or Z-link bandwidth
  - Bisection bandwidth normal to Y ~ 24x24*Z-link-bw
    - Limits All-to-All



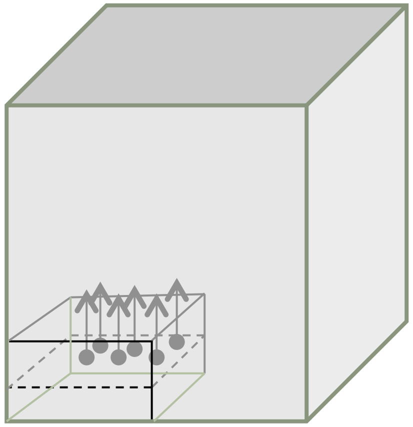
Bisection Bandwidth: Large Slab



- Consider subset of nodes: 24x6x24
- Contains ¼ of all nodes
- Bisection bandwidth through cross section:
  - Normal to X: 6*24*X-link-bw*2 for torus ~ 12x24*Z-link-bw
  - Normal to Y: 24x24*Y-link-bw ~ 24x12*Z-link-bw
  - Normal to Z: 24x6*Z-link-bw*2 for tours = 24x12 Z-link-bw
- Bisection bandwidth normal to Y ~ EQUALS that of other directions
- Bisection bandwidth for this subset is ~1/2 of bisection bandwidth for full system
- Gives highest bandwidth per node for All-to-All communication on ~ 2k nodes or more

Bisection Bandwidth: Small slab

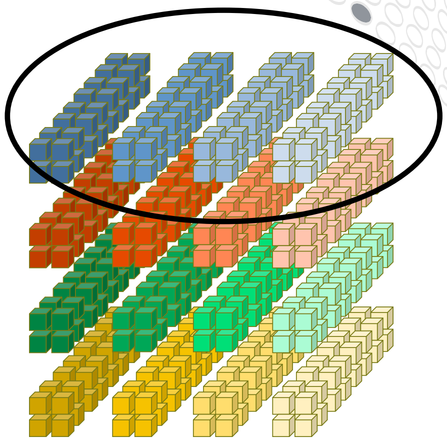
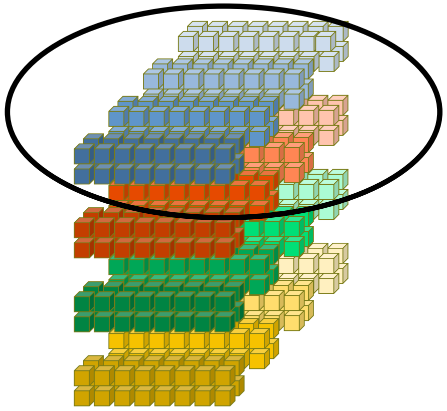
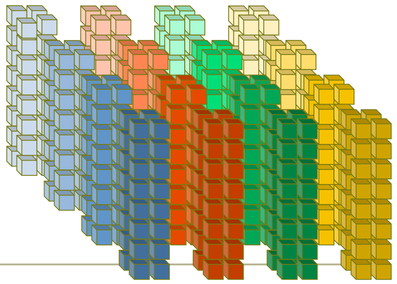
- **24x6x24 gemini subsection best for ~ 6k nodes**
  - 24x4x24 best for ~ 4k nodes
- **Consider smaller node counts, e.g., 12x6x12 so no wrapping occurs (shortest route is used)**
  - ~1700 nodes, ~1/16 of all nodes in system
- **Bisection bandwidth through cross section:**
  - Normal to X: 6*12*X-link-bw ~ 12*6*Z-link-bw
  - Normal to Y: 12*12*Y-link-bw ~ 12*6*Z-link-bw
  - Normal to Z: 12*6*Z-link-bw = 12*6 Z-link-bw
- **Bisection bandwidth normal to Y ~ EQUALS that of other directions**
- **Bisection bandwidth for subset ~ 1/8 of bisection bandwidth for full system**
  - Again gives maximum bisection bandwidth per node for All-to-All communication



PSDNS Optimizations

Minimize off-node communication

- **Transposes require All-to-All communication within each row (column) of pencils**
  - Multiple concurrent All-to-Alls on all rows (columns), not global All-to-All
- **Eliminate inter-nodal communication for xy transposes**
  - Place 1 or more full xy planes of domain per node
  - Each node has an entire row (16 or 32) of pencils
- **In benchmark runs with a 6k^3 grid on 3072 nodes, this strategy reduced the overall run time by up to 1.72X!**

COMPUTE | STORE | ANALYZE

PSDNS Optimizations: Maximize Bandwidth

Improving Transposes, II

- **YZ Transposes require off-node communication**
  - One process per node in each column communicator
  - Communication time depends on effective All-to-All bandwidth for nodes in job (plus any additional nodes relaying messages)
- **Two approaches for increasing effective All-to-all bandwidth via placement**
  1. Request node set with predefined shape (“features”)

  https://wiki.ncsa.illinois.edu/display/BWDOC/Moab+FEATURES+and+Shapes

```
#PBS -l nodes=6144:ppn=32
#PBS -nodeset=ONEOF:FEATURE:s1_6700n:s2_6700n:s3_6700n:...
```

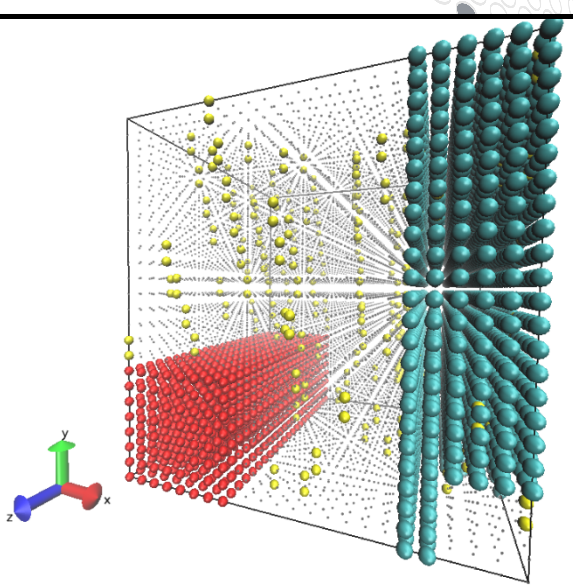
  - Bonus: job-job interference often reduced

  2. (Coming in 2014) Request node allocation with specified shape
     - X by Y by Z geminis

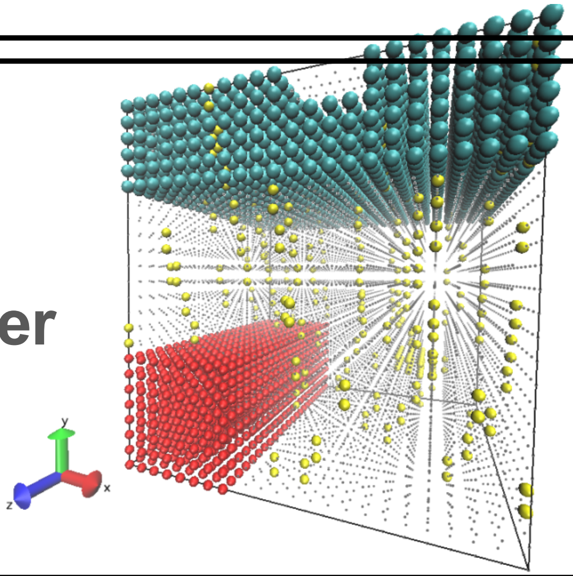
PSDNS: Effect of Slab Orientation

Allocation has fixed shape & number of nodes

- **6k XE node job**
- **6x24x24 XE gemini region**
  - Ave max time per step: 35.3 s



- **23x6x24 XE gemini region**
  - 2X more bisection bandwidth per node
  - Ave max time per step: 21.5 s
- **Job in slab normal to X takes 1.64X longer than job in slab normal to Y**



Virtual Topologies and Task Placement

- **Many applications define Cartesian grid virtual topologies**
  - MPI_CartCreate
  - Roll your own (i, j, …) virtual coordinates for each rank
- **Craypat rank placement**
  - Automatic generation of rank order based on detected grid topology
- **grid_order tool**
  - User specifies virtual topology to obtain rank order file
  - Node list by default is in whatever order ALPS/MOAB provide
- **These tools can be very helpful in reducing off-node communication, but they do not explicitly place neighboring groups of partitions in virtual topology onto neighboring nodes in torus**

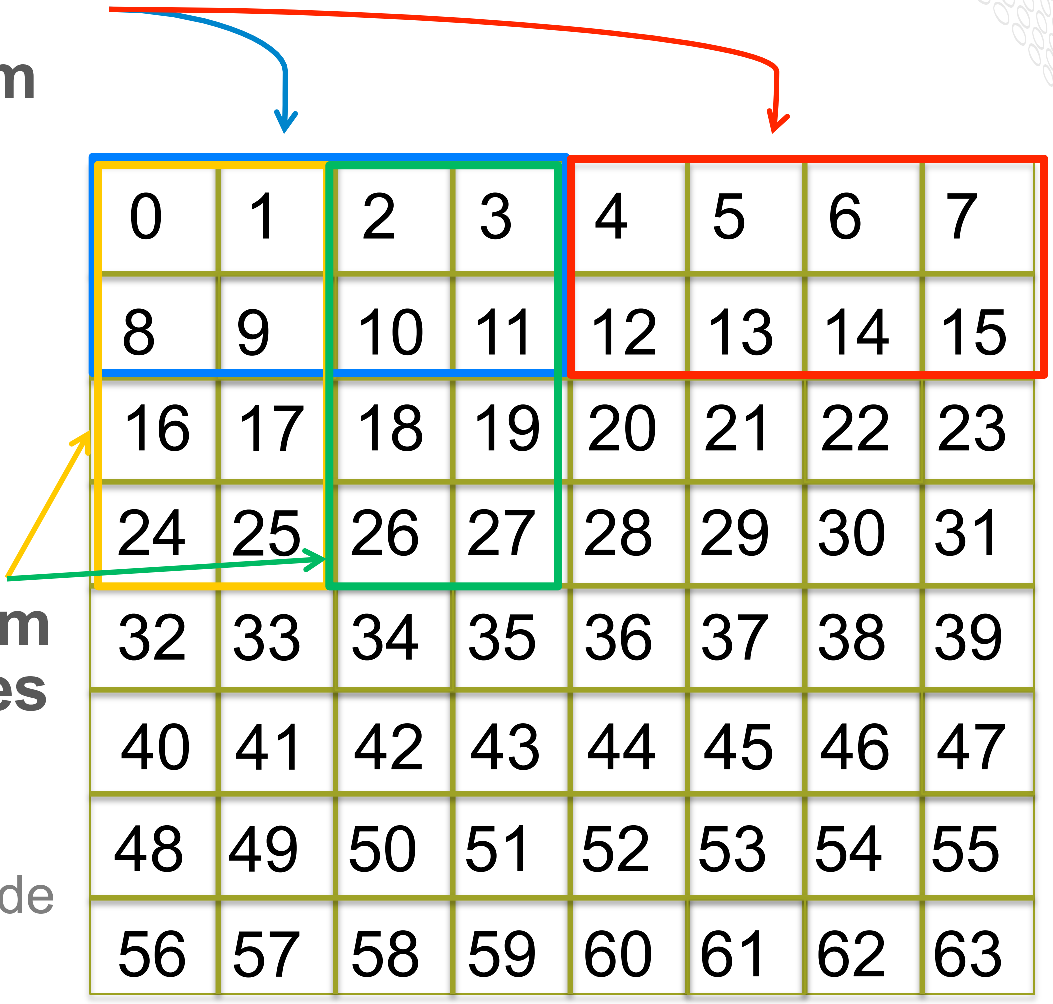
Examples: 2D Virtual topology

grid_order –C –c 4,2 –g 8,8

- **Ranks ordered with 1st dim changing fastest**
- **Nodes get 4x2 partitions**
- **Rank order is**
  - 0,1,2,3,8,9,10,11 on 1st node
  - 4,5,6,7,12,13,14,15 on 2nd
  - Node pair is 8x2

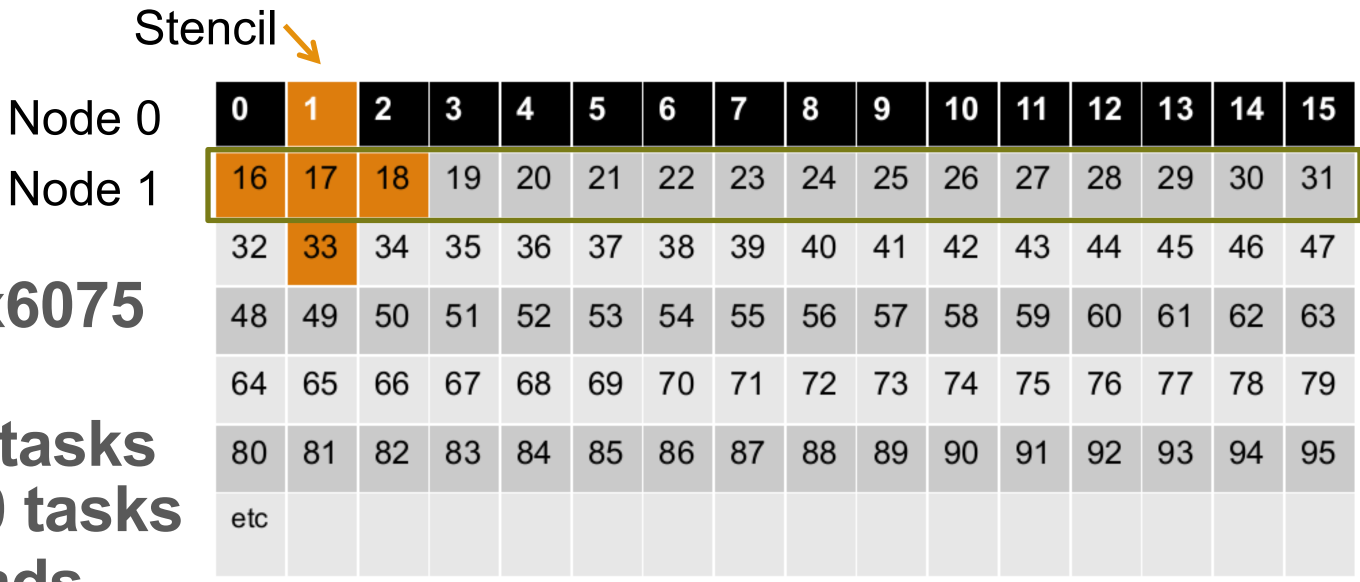
grid_order –R –c 4,2 –g 8,8

- **Ranks ordered with 2nd dim changing fastest (MPI does it this way)**
- **Rank order is**
  - 0,1,8,9,16,17,24,25 on 1st node
  - 2,3,10,11,18,19,26,27 on 2nd
  - Node pair is 4x4



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Examples: 2D Virtual Topology

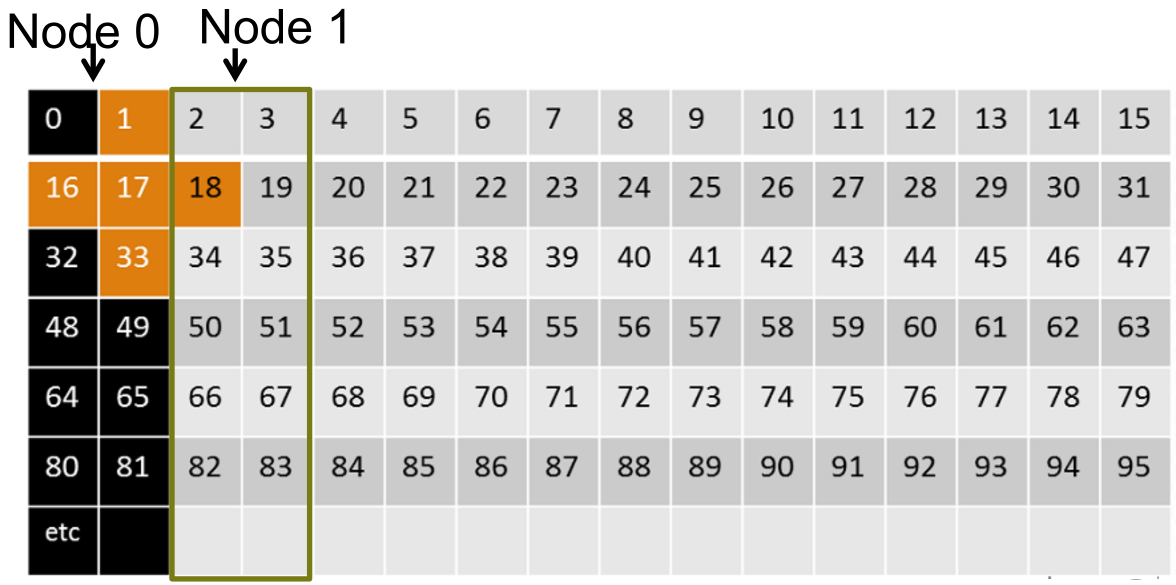


WRF

- **2D mesh, 6075x6075 cells**
- **4560 nodes, 16 tasks per node, 72960 tasks**
- **2 OpenMP threads**
- **Found best performance with**

grid_order -C -c 2,8 -g 190,384

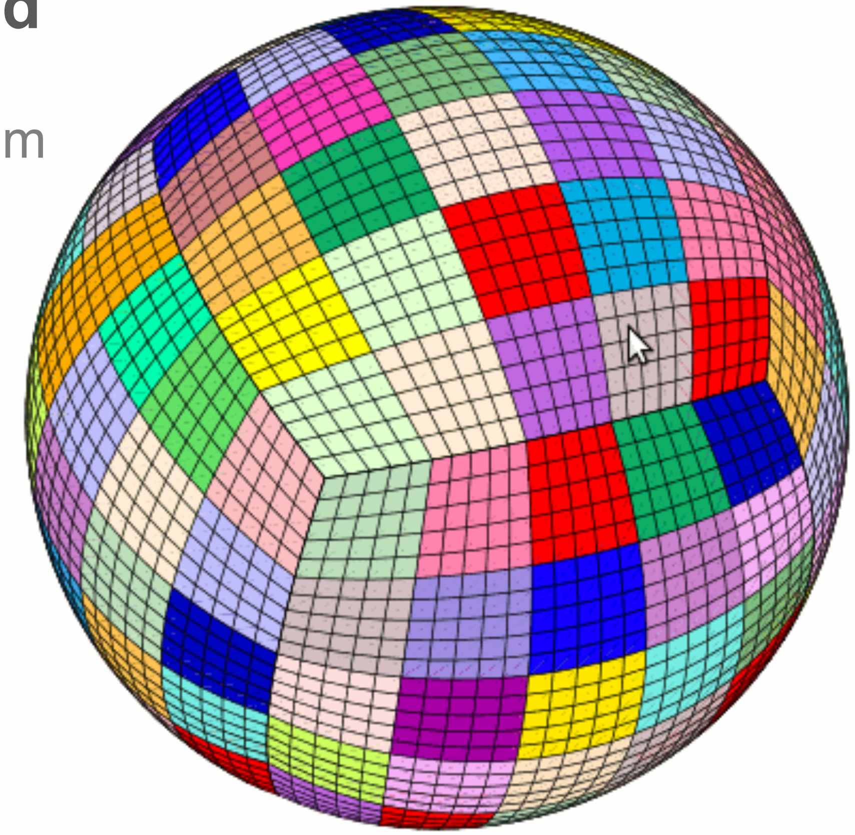
- Node pair is 4x8
- ~18% speedup over SMP ordering



Examples: 3D Cubed Sphere

SPECFEM3D_GLOBE

- **Quad element unstructured grid**
- **5419 nodes, 32 tasks per node**
- **Craypat detected a 1020x170 grid pattern (8 less than # tasks)**
  - On-node 81% of total B/task w/Custom
  - On-node 48% of total B/task w/SMP
- **Found best performance with grid_order –R -c 4,1 -g 1020,170**
  - Each node gets eight 4x1 patches
  - Also tried –c 8,2, etc.
  - 16% speedup over SMP ordering



Examples: 4D Virtual Topology

MILC

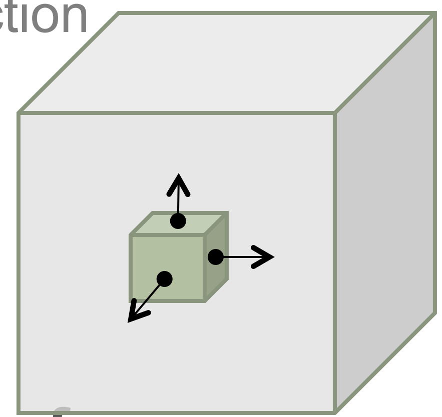
- **4D Lattice, 84x84x84x144**
- **4116 nodes, 16 tasks per node, 65856 tasks**
- **6x6x6x6 lattice points per task**
- **Found best performance with**

grid_order –R -c 2,2,2,2 -g 14,14,14,24

  - 1.9X speedup over SMP ordering!
  - Difficult to map 4D virtual topology onto 3D torus using 2x2x2x2
  - Possible to improve performance further by selecting which nodes to use (later)

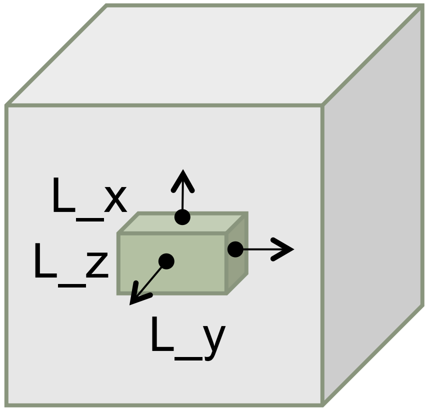
Choosing Tile Sizes

- **Consider applications that perform nearest-neighbor communication in a 3D virtual Cartesian grid**
  - Assume same amount of communication in each direction
- **Communication time for halo exchange ~ tile_face_points / link_bandwidth**
- **Cubic tile: same face points in all 3 directions**
  - T_comm_cubic_x ~ tile_face_points / X-link-bw
  - T_comm_cubic_y ~ tile_face_points / Y-link-bw
  - T_comm_cubic_z ~ tile_face_points / Z-link_bw
- **Longest time is T_comm_cubic_y, by a factor of ~ 2**
- **Limits performance if 3 directions done concurrently:**
  - T_comm_cubic = L^2/Y-link-bw = 2 * T_comm_cubic_x
- **If directions must be done in sequence**
  - T_comm_cubic ~ 4* T_comm_cubic_x



Choosing Tile Sizes

- **Elongated tile: assume same # points as cubic tile, but different # of face points in different directions**
  - T_comm_x ~ X_face_points / X-link_bw
  - T_comm_y ~ Y_face_points / Y-link_bw
  - T_comm_z ~ Z_face_points / Z-link_bw
- **These three times are equal if**
  - X_face_points = Z_face_points = 2*Y_face_points
  - L_y = 2 * L_x
  - V = L^3 from cubic case ➔ L_x = L / 2^(1/3)
  - T_comm_x = 2^(1/3) T_comm_cubic_x
- **If communication for all 3 directions concurrent**
  - T_comm = T_comm_cubic * 2^(1/3) / 2 = 0.63 * T_comm_cubic
- **If 3 directions done in sequence**
  - T_comm_seq = T_comm_cubic_seq * 2^(1/3) * (3/4)
    = 0.945 * T_comm_cubic_seq
- **Bottom line: If possible, do all 3 directions concurrently and use tiles with 2X more cells along Y**



Choosing Tile Sizes

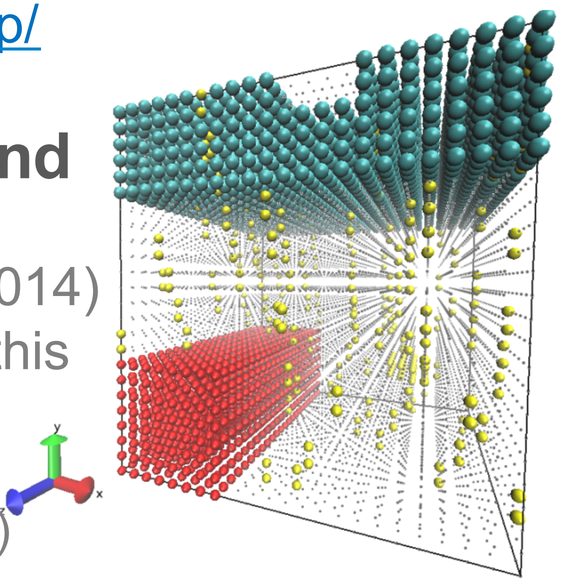
Example: tile size for cubic grid

- **Global mesh with 1024^3 zones, 32x32x32 partitions**
- **To get cubic tiles**
  - Could have 4x4x2 partitions per node
  - Does not take slower y-links into account
- **To get 2X more points along y → 1/2 as many y-partitions**
  - Partition global mesh with 1000^3 zones as 40x20x40
  - Each partition has 25x50x25 mesh zones
  - Could have 4x2x4 partitions per node
  - Up to 1.6X faster halo exchanges than 32^3 partition case, provided communication is done over all 3 dimensions at once
  - Only 6% improvement if exchanges are done 1 dimension at a time

Going Beyond Simple Rank Reordering

Significant improvement possible

- **Can we place tasks on a given set of nodes so that virtual neighbors are nearby on torus?**
  - Difficult problem for arbitrary node lists
  - Possibly helpful library: Hoefler's LibTopoMap http://htor.inf.ethz.ch/research/mpitopo/libtopomap/
  - Not widely used
- **Can we specify size of prism of geminis and directly map virtual topology to torus?**
  - Use predefined node sets, or request via qsub (2014)
  - Presence of service & down nodes complicates this
  - Two ways to do mapping:
    1. Write a topology-aware application (hard)
    2. Use Topaware tool (easy, no source code changes)



How To Get Node IDs and Torus Coordinates

MPI

- **Use MPI_Cart_Create, MPI_Cart_coords, MPI_Cart_Shift, etc. to get ranks of neighboring tasks**
  - Suppose array rnks(0:nbrs) contains my rank and ranks of nbrs neighbors
- **Use Cray libs (next slide) to get node IDs and coords.**

my_prog.f90

```
integer, parameter :: size=10  ! match torus_coords.c
integer isize,rnks(0:size-1),nid(0:size-1),  &
   tx(0:size-1),ty(0:size-1), tz(0:size-1)
common /cblock/ isize,rnks,nid,tx,ty,tz

isize = 1 + nbrs
call torus_coords ()
```

torus_coords.c

```
#include <mpi.h>
#include "pmi.h"
#include "rca_lib.h“

extern struct {
int isize, rnks[10], nid[10], tx[10], ty[10], tz[10];
} cblock_;

void torus_coords_() {
   int irank, rank, *nidlist;
   rca_mesh_coord_t rca_coords;  // struct of unsigned short ints
   uint16_t nidu;

   PMI_Get_nidlist_ptr((void **)&nidlist);
   // nidlist now points to a list of nid numbers in rank order:
   // nidlist[p] is the nid number of rank p in this job

   for (irank=0; irank<cblock_.isize; irank++)
   {
      rank = cblock_.rnks[irank];
      cblock_.nid[irank] = nidlist[rank];
      nidu = (uint16_t) nidlist[rank];
      rca_get_meshcoord(nidu, &rca_coords);
      cblock_.tx[irank] = (int)rca_coords.mesh_x;
      cblock_.ty[irank] = (int)rca_coords.mesh_y;
      cblock_.tz[irank]  = (int)rca_coords.mesh_z;
   }
   return;
}
```

Topaware: Node Selection and Task Placement

Purpose

- Given application w/2-, 3-, or 4-D grid communication graph
- Given particular input deck and decomposition
- Find near-optimal layout on given Cray XE/XK system
- Explore best possible performance and scaling

Limitations

- Presence of service nodes limits max node count
- Not all decompositions can be placed ideally
  - Number of usable nodes along each torus direction
  - Number of partitions per node
- Leaves some idle nodes in prism of geminis

Topware: New Compiled Language Version

- **Helps you choose problem size/node count that will map well to torus**
- **Can be used with node sets**
  1. Choose target node count that fits in a series of predefined “features” https://wiki.ncsa.illinois.edu/display/BWDOC/Moab+FEATURES+and+Shapes
  2. Choose problem size that fits in “features” with help from Topaware
  3. Submit batch job targeting those features
  4. Run Topaware within batch job to select subset of nodes application will actually use
  5. Specify Topaware-generated node list and rank order on aprun command line

Topaware Node Selection Scheme

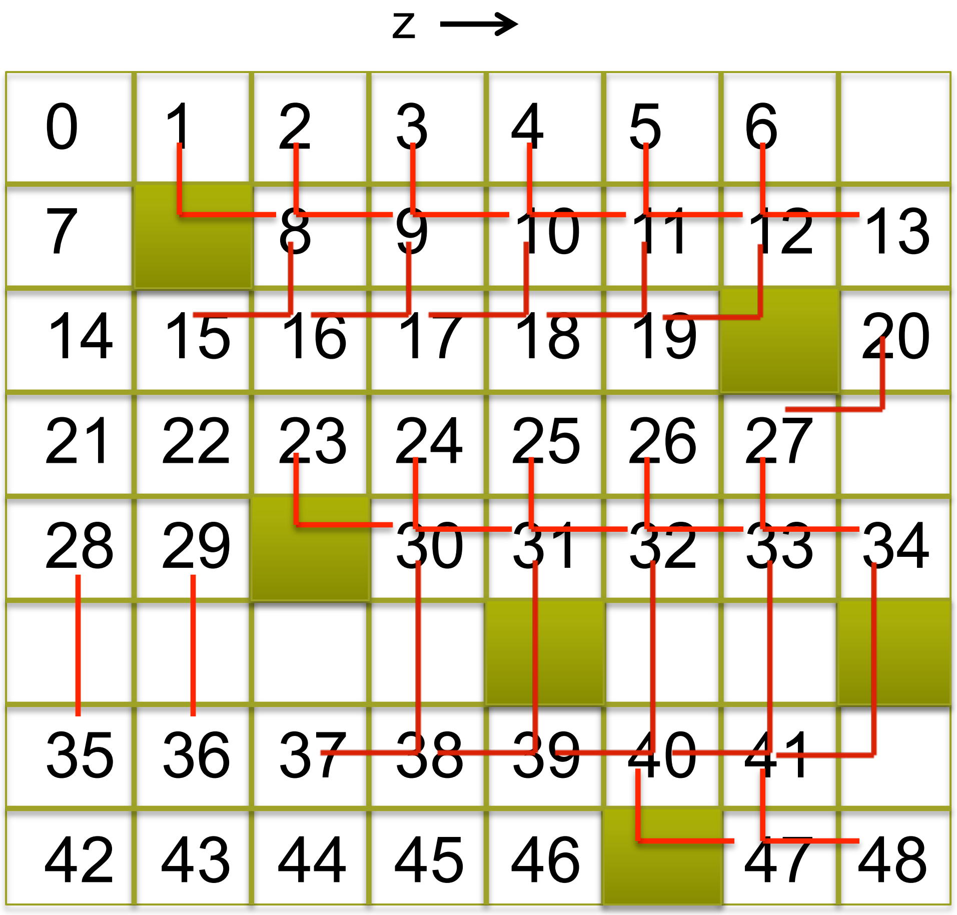
- One XZ plane shown
- Most rows and columns have 0 or 1 service node (green)
- Can fit up to a 7x7 gemini layout onto this 8x8 torus cross section
  - Selects 7 geminis in same rows they would have w/o service nodes
  - All selected geminis are also in same plane as w/o service nodes
- Scan in Y to find enough usable XZ planes
- Skipping an x value rarely occurs in practice

z $\longrightarrow$

0	1	2	3	4	5	6	
7	🟩	8	9	10	11	12	13
14	15	16	17	18	19	🟩	20
21	22	23	24	25	26	27	
28	29	🟩	30	31	32	33	34
				🟩			🟩
35	36	37	38	39	40	41	
42	43	44	45	46	🟩	47	48

Extra hops for North/South exchange

- Many hubs require second hop to reach some neighbors
- Density of multiple hops does not increase with scale, nor does # hops
- Should enable nearly ideal weak scaling, despite extra hops



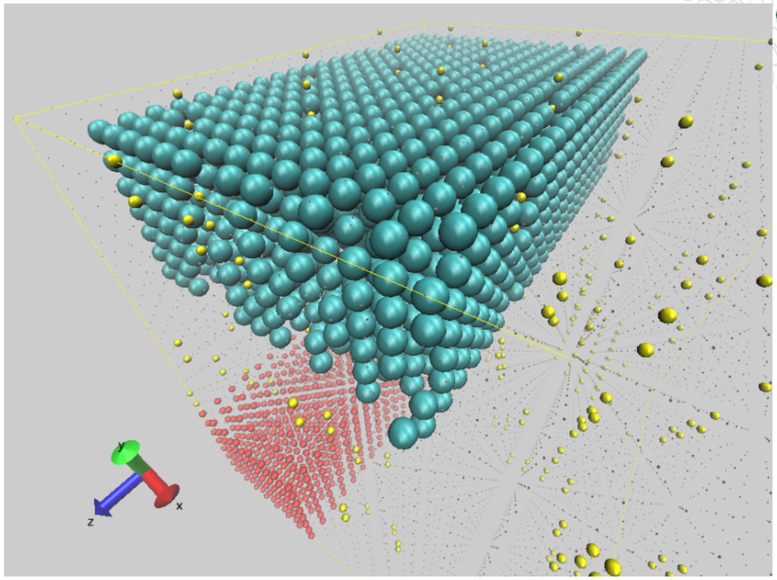
Examples: 4D Halo Exchanges

Compare default ordering, grid_order, and Topaware on <u>same set of nodes</u> (selected by Topaware).

- **4D Lattice, 144x144x144x288 points**
- **12x16x16x16 partitions**
- **1536 nodes, 32 tasks per node, 49152 tasks**
- **12x9x9x18 lattice points per task**
- **Periodic BCs**
- **Topaware: each node gets 1x2x2x8 tasks**

- **Run times**
  - Default placement (SMP): 0.0240 s
  - grid_order -C -g 12,16,16,16 -c 2,2,2,4: 0.0245 s (worse than default!)
  - Topaware: 0.0127 s (1.9X < default!!)

Results on Blue Waters



MILC

- **4D Lattice, 84x84x84x144**
- **4116 nodes, 16 tasks per node**
- **6x6x4x9 lattice points per task**
- **Entire 4th dimension on each node pair**
  - Remaining 3 dimensions mapped like any 3D virtual topology
  - 14x7x21 geminis
  - 1x2x1x16 partitions per node pair
- **3.7X faster than default SMP placement**
  - 1.9X faster than when using grid_order –c 2x2x2x2 …

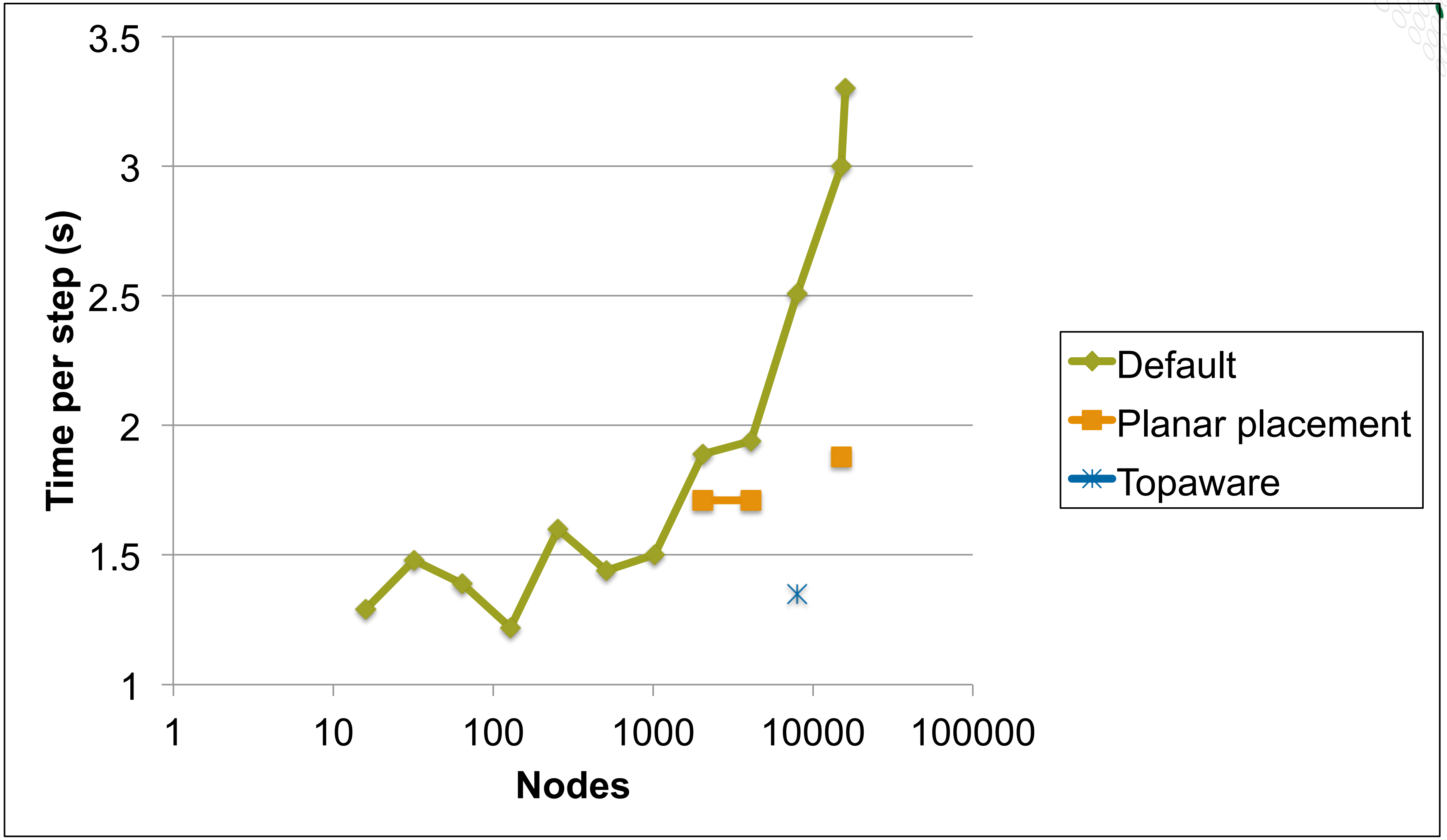
Results on Blue Waters for VPIC

- **Plasma physics**
- **3D virtual topology**
- **On 2k nodes, this code spends 8% of total run time on communication**
- **Ran on 4608 nodes in dedicated mode**
  - 12x12x16 geminis
  - 4x4x2 partitions per node pair
- **Best results: 5% faster total run time than default placement**

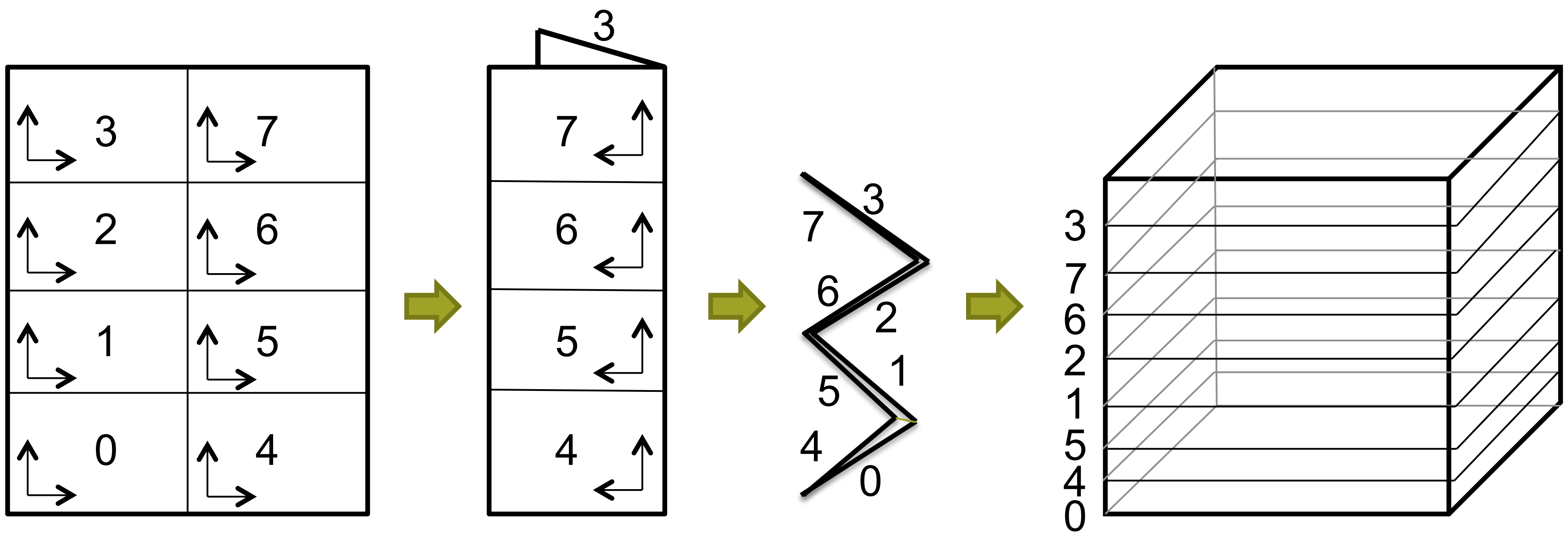
Results on Titan for S3D (R. Sankaran@ONRL)

- **Fluid dynamics w/ combustion**
- **3D Virtual topology**
- **Ran on up to ~12900 nodes in dedicated mode**
  - Near linear weak scaling (unlike default placement; see next slide)
- **Topaware placement → faster run times than default**
  - 2000 nodes: 1.32X
  - 6000 nodes: 1.61X

Results on Titan for S3D (R. Sankaran@ONRL)



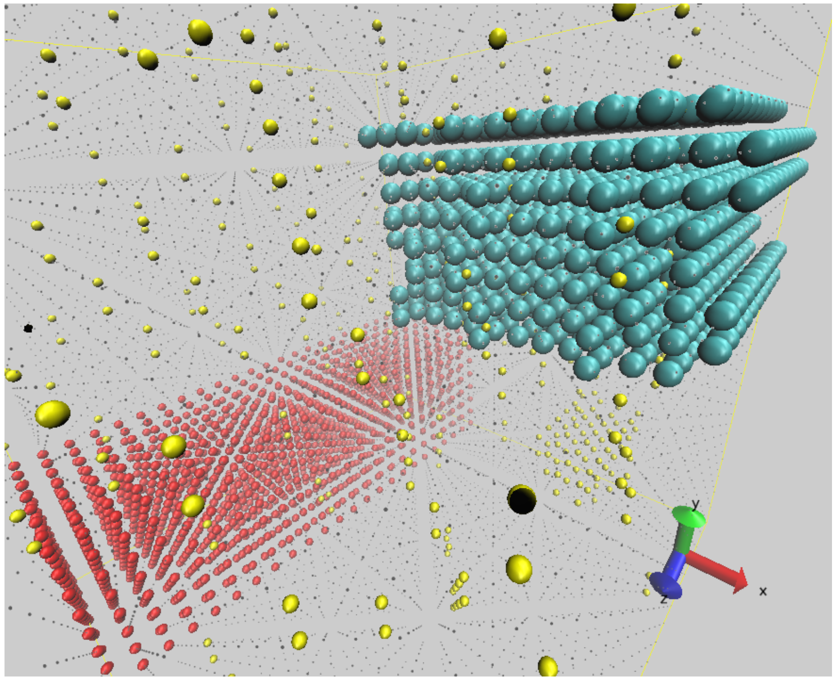
Mapping 2D Virtual Topology to 3D Torus

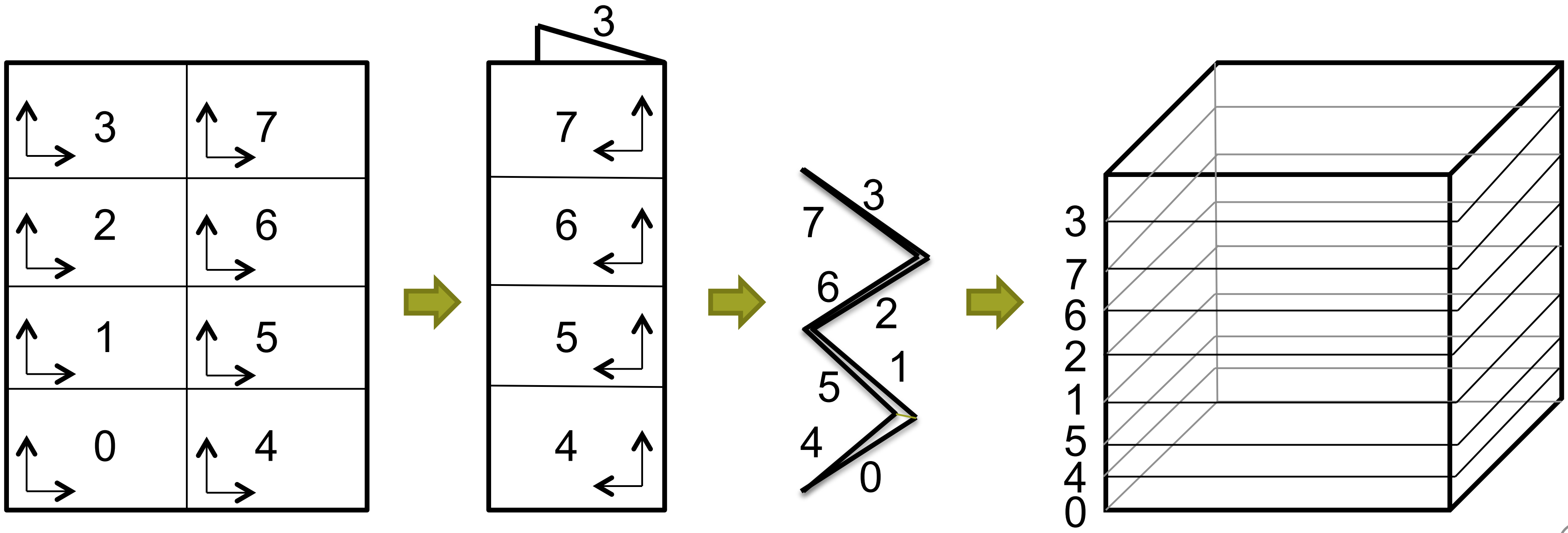


- **2D domain is folded like a sheet of paper into 8 supertiles**
  - Fold in half along one dimension, then 3 times in the other
  - No tearing – keeps neighbors close together
  - Communication between tiles is confined to super-tile edges
  - Folding in both dimensions overloads links shared by 4 supertiles
  - Optimal when folding along just one dimension
    - But results in long, thin tiles that increase “surface to volume” ratio

Staggered Supertiles

- **12x8x10 geminis**
  - 8 XZ planes
  - Stacked along Y
  - 4&5 and 6&7 staggered in X to avoid sharing links
  - Max hops = 4





Remarks on Topaware

- **NO application modifications are required for Topaware**
  - Set MPICH_RANK_REORDER_METHOD to 3
  - aprun –L`cat node_list` …
- **This goes beyond Craypat/grid_order rank reordering:**
  - We pick which nodes to use
  - We make sure that neighboring tiles (all processes on a node) in the MPI Cartesian topology are placed on near-neighbor hubs on the torus
  - We control more precisely how ranks are placed on nodes

FAQ

- **How am I able to make these plots of nodes on BW?**
  - VMD, a visualization package for molecules (NCSA has tool)
  - Input node lists (used by job, etc.) with torus coordinates
    - “ver_sim_new” program in Topaware suite
- **How do I know which nodes my job ran on?**
  - Place this line in your batch job script:
    aprun -B -D0x10000 /bin/true | head -1 > node_list.$PBS_JOBID
- **What is the best way to contact me?**
  - Email rfiedler@cray.com

THE SUPERCOMPUTER COMPANY