Improving Performance of All-to-All, Random Pair, and Nearest-Neighbor Communication on Blue Waters

February 27, 2012

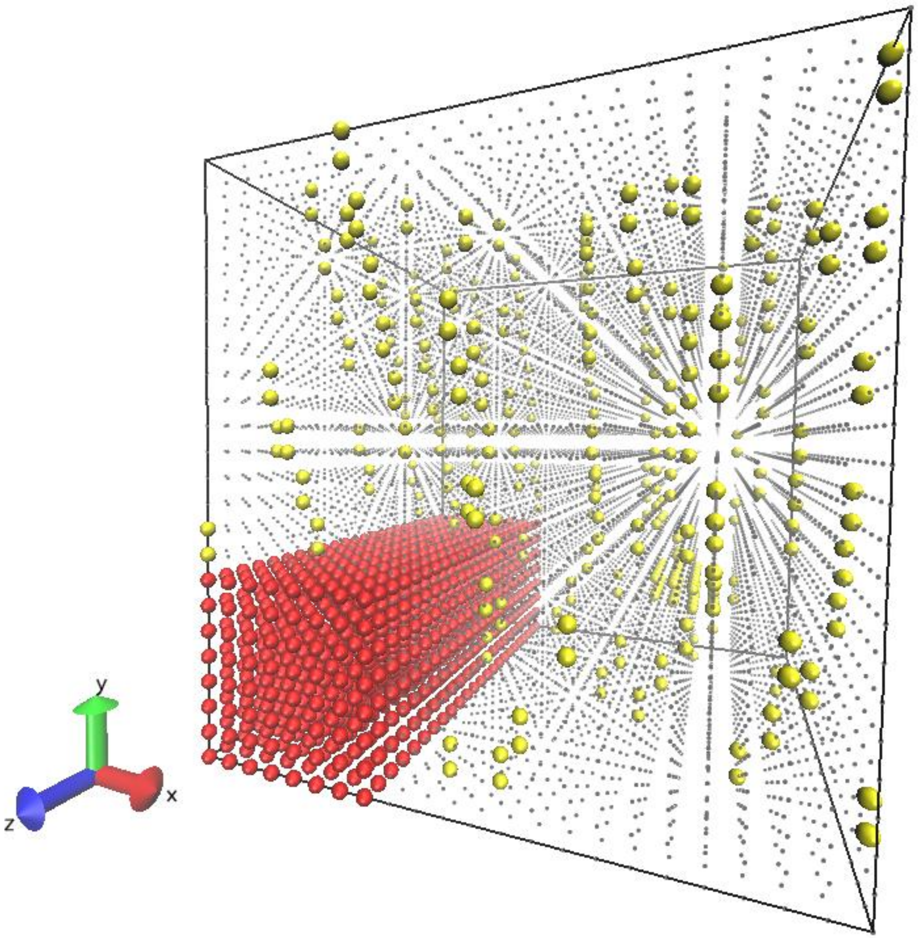
R. Fiedler
PRAC Applications Analyst

Part 1: All-to-All & Random Pair Communication

Background

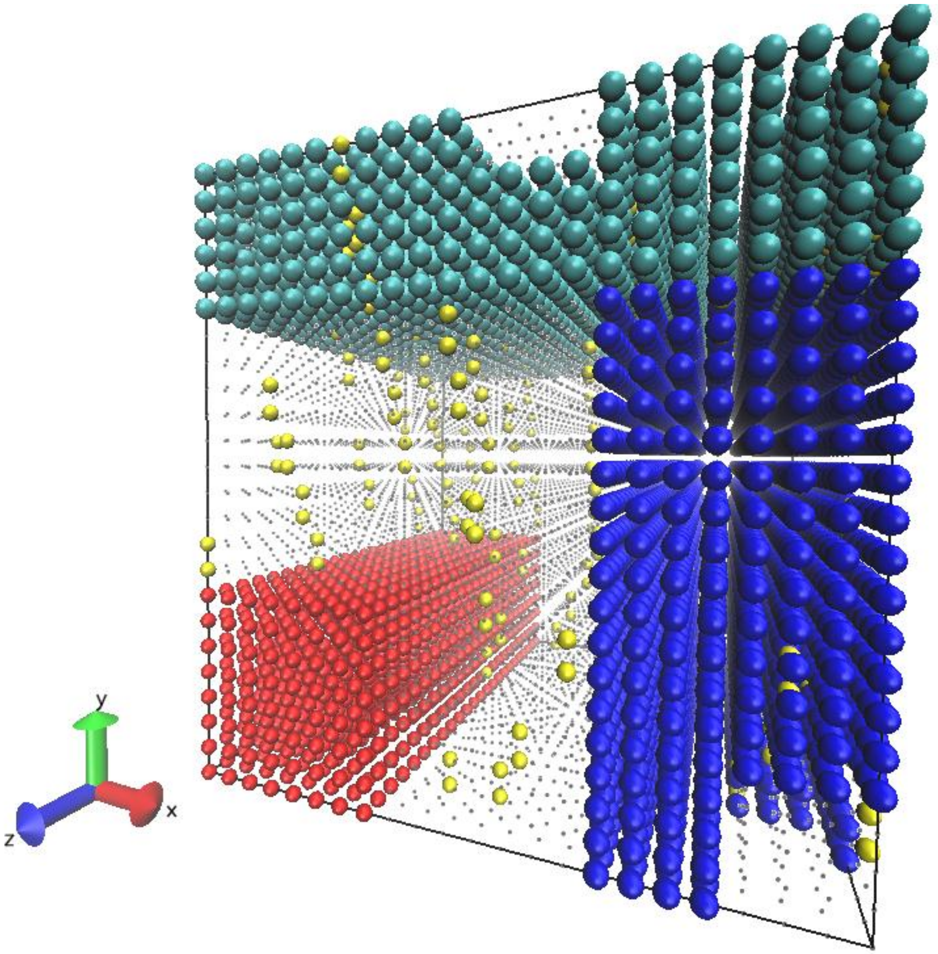
BW Interconnect

- Topology is 23x24x24 gemini hubs
- 2 nodes per gemini
- 8x8x24 XK geminis (red)
- Service nodes randomly distributed (yellow)
- Y-links between geminis have 1/2 bandwidth of X- or Z-links
  - 2 geminis on same board have 2X faster links in Y than Y-links between boards
  - 2 nodes on same gemini don't use interconnect to exchange messages
- Routing algorithm is X, then Y, then Z



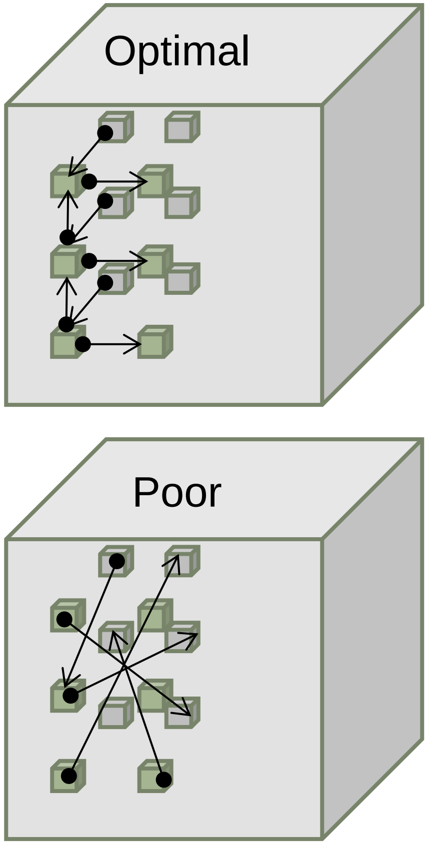
Background

- Routing takes shortest path
- If using > 1/2 of nodes in a given dimension, some communication may wrap around the torus through nodes not assigned to job
- Jobs share interconnect for application communication, IO
- Run times affected by task placement, other running jobs



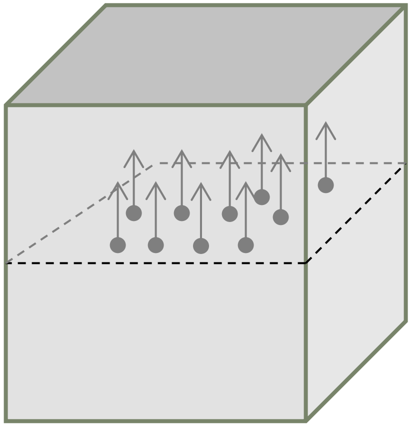
Task Placement and Interference

- **Applications that perform more communication are more sensitive to placement and interference**
- **Applications with All-to-All communication patterns compete more with other jobs**
- **Applications with only nearest-neighbor communication in their virtual topology, if poorly placed, actually perform pairwise communication between randomly located nodes**
  - Thus, analysis below of bisection bandwidth for All-to-All is relevant to many types of applications



Bisection Bandwidth

- **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 23x24x24 geminis
  - Bisection bandwidth through cross section:
    - Normal to X: 24x24*X-link-bw*2 for torus
    - Normal to Y: 23x24*Y-link-bw*2 for torus
    - Normal to Z: 23x24*Z-link-bw*2 for tours
  - Y-link bandwidth ~ 1/2 X-link or Z-link bandwidth
  - Bisection bandwidth normal to Y ~ 23x24*Z-link-bw, limits All-to-All



Bisection Bandwidth

1-D torus vs. 1-D mesh

```
0    1    2     3    4    5    6
*----*----*----*----*----*----*
|___________________________|
```

- **Suppose each node sends different messages to all other nodes**
- **Can send multiple messages simultaneously on each connected link**
- **Mesh: 1 path connects nodes 0 and 6 through other nodes.**
- **Torus: 1 path connects to 3 nodes on right, another path connects to 3 nodes on left**
  - Thus, torus has twice the bisection bandwidth of mesh
  - All-to-All is 2X faster for torus

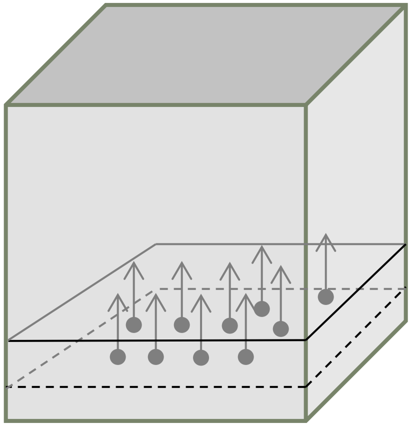
Bisection Bandwidth

1-D torus vs. 1-D mesh

```
0    1    2     3    4    5    6
.----.----*----*----*----*----*
|_____________________________|
```

- **If not all nodes participate in all-to-all, torus bandwidth < 2X mesh bandwidth**
- **E.g., nodes 0 and 1 not assigned to job but relaying messages**
  - Node 2 reaches node 6 in 3 hops through nodes 1 and 0 for torus
  - Messages to/from nodes 3, 4, 5 to any other node don’t benefit from torus
  - Only 1 of 4 messages sent by node 2 uses link between nodes 2 and 1
  - Torus All-to-All takes 3/4 of time for mesh All-to-All, not 1/2

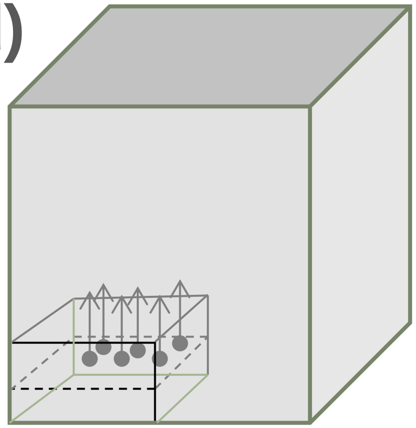
Bisection Bandwidth



- Consider subset of nodes: 23x6x24
- 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: 23x24*Y-link-bw ~ 23x12*Z-link-bw
  - Normal to Z: 23x6*Z-link-bw*2 for tours = 23x12 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 possible bandwidth per node for All-to-All communication

Bisection Bandwidth

- **23x6x24 gemini subsection best for ~ 6k nodes**
  - 23x4x24 best for ~ 4k nodes
- **Consider smaller node counts, e.g., 11x6x12 so no wrapping around torus (shortest route is used)**
  - 1584 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: 11*12*Y-link-bw ~ 11*6*Z-link-bw
  - Normal to Z: 11*6*Z-link-bw = 11*6 Z-link-bw

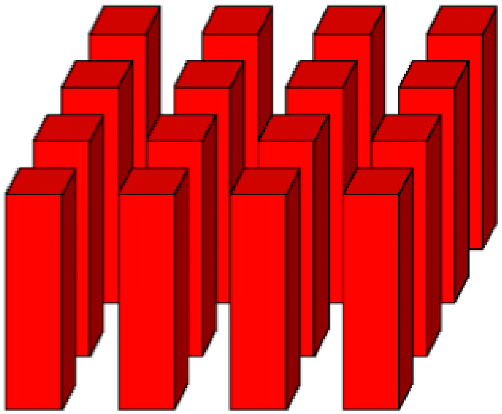


- **Bisection bandwidth normal to Y ~ EQUALS that of other directions**
- **Bisection bandwidth for subset ~ 1/8 of bisection bandwidth for full system**
  - Again gives maximum bandwidth per node for All-to-All communication

PSDNS Turbulence Application

CFD Using Pseudo-Spectral Method

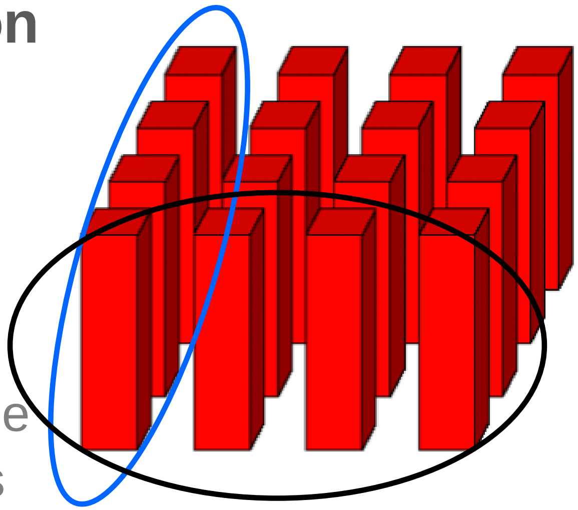
- **Uses 3D FFTs of fluid variables to compute spatial derivatives**
- **Implementation uses 2D pencil decomposition**
- **For 3D FFT, must transpose full 3D arrays twice:**
  - Begin with partitions spanning domain in X
  - 1D FFTs along X
  - Transpose within XY planes so each partition spans domain in Y
  - 1D FFTs along Y
  - Transpose within XZ 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**



PSDNS Turbulence Application

Improving Transposes, I

- **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
- **Optimization: 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 ~35%**
- **Possible to place 1 XY plane per gemini (node pair), but must ensure both nodes are up on all geminis used (later)**



PSDNS Turbulence Application

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
    - Can be << global, system-wide All-to-All bandwidth
- **Two approaches to increasing effective All-to-all bandwidth via placement**

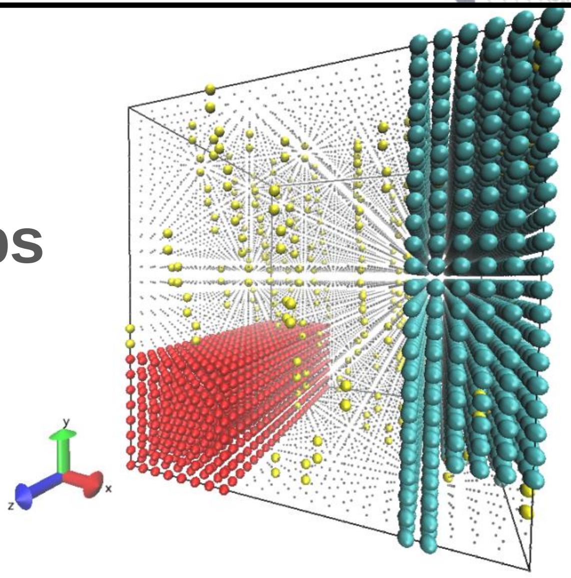
1. Request specific nodes & wait – works in shared mode (later)
   - qsub -l hostlist=`cat node_list | sed -e 's/-/+/g' | sed -e 's/,/+/g'` job_script
2. Run on a randomly distributed (spread out) set of nodes
   - Most useful on dedicated system (or reservation)
   - For a 6k^3 grid on 3072 nodes of ESS (~4k nodes total), this strategy reduced the overall run time by ~21%

PSDNS Turbulence Application

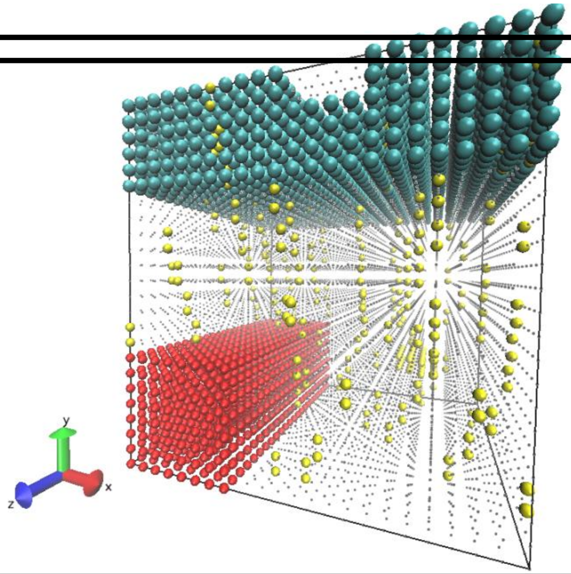
Sensitivity to Placement

- **6144 XE nodes, 8 non-IO steps, 2 IO steps**

- **6k-node job in 6x24x24 XE Region**
  - Ave max time per non-IO step: 35.3 s
  - Ave max time per IO step: 67.9 s



- **6k-node job in 23x6x24 XE region**
  - Ave max time per non-IO step: 21.5 s
  - Ave max time per IO step: 48.0 s
  - Slab normal to X takes 1.64X (1.41X for IO) longer than slab normal to Y

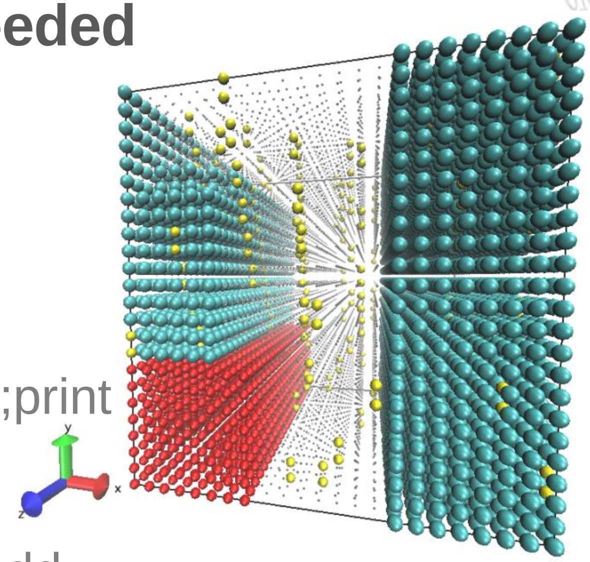


PSDNS Turbulence Application

Ensuring both nodes on each gemini are up

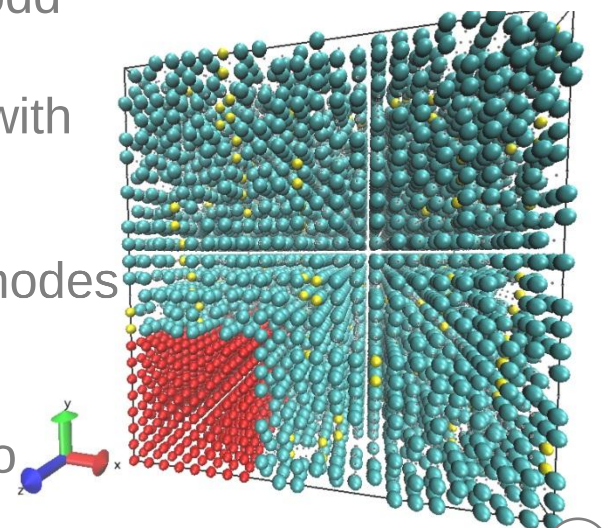
- **Request a few (~0.5%?) more nodes than needed by job**
- **At run time in batch script**
  - Get the list of nodes in reservation:

```
checkjob --xml $PBS_JOBID | perl -e 'while(<>){if
(/AllocNodeList=\"([0-9:,]*)/){$n=$1;$n =~ s/:\d+//g;print
"$n\n";}}‘ > node_list
```



  - Node IDs on same gemini are consecutive even-odd integers
  - Randomization script (later) can eliminate nodes with down partners:

```
cat node_list | randomize.pl --block=2 > random_nodes
aprun –l random_nodes …
```



  - Randomizing node list useful for random-pairs, too

PSDNS Turbulence Application

Improving Transposes, III

- **Replace calls to MPI_AlltoAll with library routine in co-array Fortran (CAF)**
  - CAF has one-sided communication, lower latency, smaller headers
  - Library routine copies messages to/from 4 MB statically allocated co-array “bucket” on each image
  - Breaks messages into 512 B chunks
  - Pulls chunks from other images in a different random order for each image
    - Reduces network congestion
    - Reduces length of time links are devoted to a given message
  - Tunable for specific application – source available
    - Saves source/target info and random orderings for the row and column communicators
- **Reduces the overall run time by ~33% on 4096 nodes**

PSDNS Turbulence Application

CAF Integration

```
 #ifdef CAF
 call compi_alltoall(sendbuf,recvbuf,items,mpi_comm_col)
#else
 call mpi_alltoall(sendbuf,items,mpi_byte,
    &                recvbuf,items,mpi_byte,mpi_comm_col,ierr)
#endif
```

- compi_alltoallv also available, nearly as efficient

PSDNS Turbulence Application

Improving "Compute" Time

- PSDNS allocates/deallocates buffer arrays for communication every time it performs All-to-All operations
- For PGI (maybe GNU) compiler, a 10-20% improvement in run time was obtained by setting environment variables:
  - MALLOC_MMAP_MAX_=0
  - MALLOC_TRIM_THRESHOLD_=512MiB
- Cray compiler by default manages memory better, so setting these variables does not help
- Avoiding repeated allocation/deallocation of the same arrays may reduce overhead for many applications

Part 2: Nearest-Neighbor Communication

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 (column major, like Fortran)
- 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
- 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

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
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
etc															

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
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
etc															

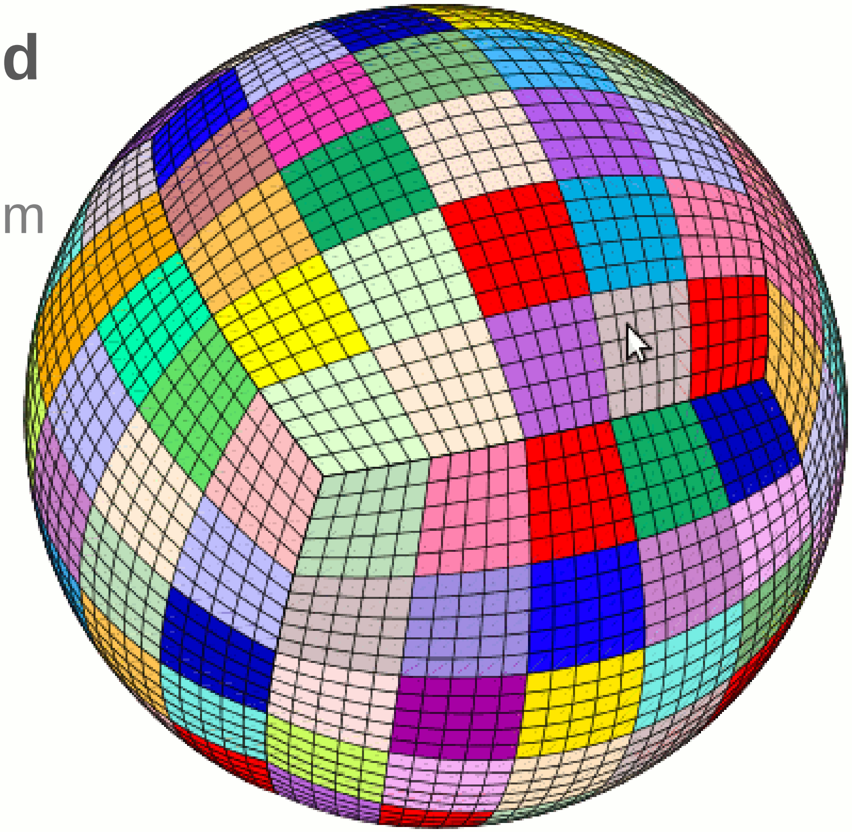
Examples: 3D Cubed Sphere

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

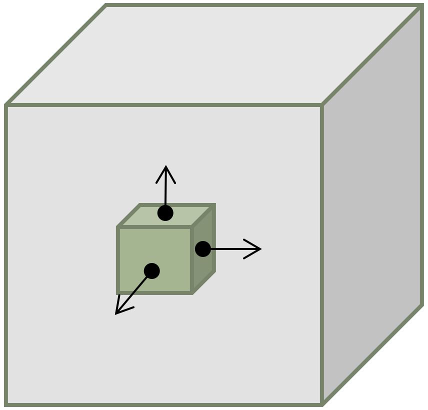
- **4D Lattice, 84x84x84x144**
- **4116 nodes, 16 tasks per node, 65856 tasks**
- **6x6x6x6 lattice points per task**
- **Found best performance with**

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

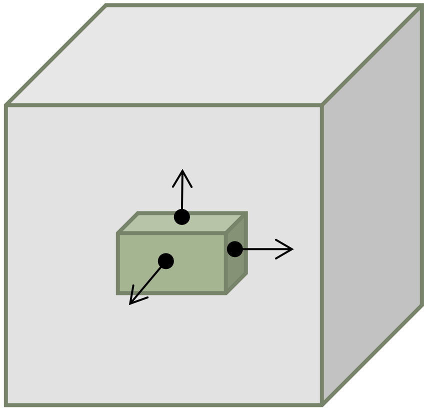
- **Consider applications that perform nearest-neighbor communication in a 3D virtual Cartesian grid**
  - Assume same amount of communication in each direction
- **Communication time for halo exchange ~ tile_face_area / link_bandwidth**
- **Cubic tile: same area normal to all 3 directions**
  - T_comm_cubic_x ~ tile_face_area / X-link-bw
  - T_comm_cubic_y ~ tile_face_area / Y-link-bw
  - T_comm_cubic_z ~ tile_face_area / 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 volume as cubic tile, but different face areas in different directions**
  - T_comm_x ~ X_face_area / X-link_bw
  - T_comm_y ~ Y_face_area / Y-link_bw
  - T_comm_z ~ Z_face_area / Z-link_bw
- **These three times are equal if**
  - X_face_area = Z_face_area = 2*Y_face_area
  - 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**

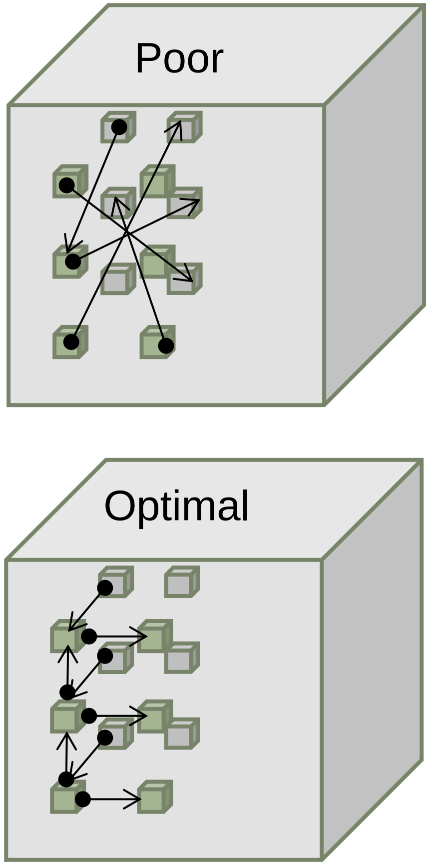
Choosing Tile Sizes

Example: tile size for cubic grid

- **Global mesh with 1024^3 zones, 32x32x32 partitions**
- **Each node has 16 compute units, 32 integer cores**
- **To get cubic tiles**
  - Could have 2x2x2 partitions per node (w/ 2 or 4 OpenMP threads)
  - Could have 4x4x4 partitions per node pair, single threaded
  - But neither of these 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, single threaded
  - Could have 4x2x4 partitions per node pair (both partners up)
    - 2x2x4, 4x2x2, or 4x1x4 partitions per node (different rank orders)
  - Nearly 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

Selecting Nodes to Use

- **Very desirable to place tiles on any given set of nodes so that virtual neighbors are nearby on torus**
  - Difficult problem for arbitrary node lists
  - If application uses most nodes in a reservation with a specified node list, then can apply existing Topaware tool (later)
    - Ensures neighboring tiles are placed on nearby nodes in torus
    - Takes into account presence of service nodes
  - Enabling Topaware to place tiles that should be neighbors close together on the torus in shared batch mode is under investigation



Motivation for Developing Topaware

- Applications that perform mainly nearest-neighbor communication on a 3D mesh should weak scale linearly on a 3D torus interconnect.
- Such apps should map nicely to a 3D torus, but service nodes scattered throughout the system impede finding a good mapping even on a dedicated system.
- As a result, halo exchanges can take considerably more time than models predict.

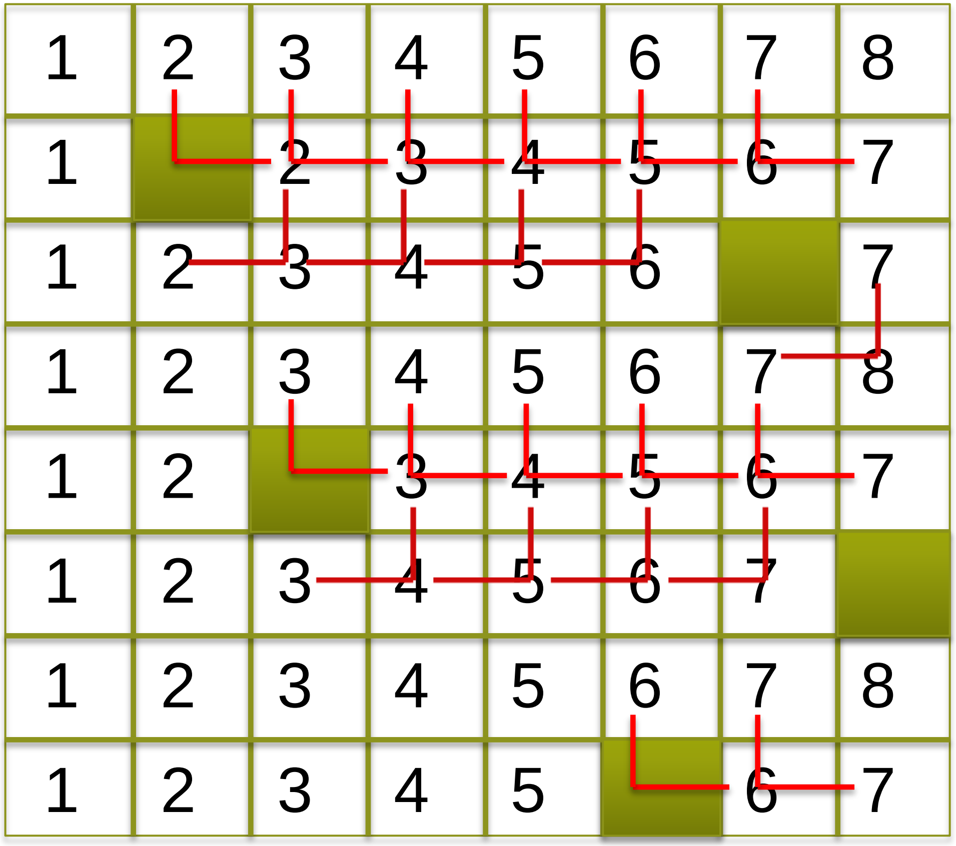
Topaware Node Selection Scheme

- Most rows and columns have 0 or 1 service node (green)
- Can fit up to a 7x7 gemini plane onto this 8x8 section of torus
- This mapping selects 7 geminis in the same rows they would have w/o service nodes
- All selected geminis are also in the same plane as w/o service nodes

1	2	3	4	5	6	7	8
1		2	3	4	5	6	7
1	2	3	4	5	6		7
1	2	3	4	5	6	7	8
1	2		3	4	5	6	7
1	2	3	4	5	6	7	
1	2	3	4	5	6	7	8
1	2	3	4	5		6	7

Extra hops for up/down exchange

- About half of the hubs require a second hop to reach North neighbor
- Density of double hops does not increase with scale, nor does # hops
- Should enable nearly ideal weak scaling, despite extra hops



Results on Blue Waters for MILC

- **Lattice Quantum Chromodynamics**
- **4D Lattice, 128x128x128x192**
- **8192 nodes, 32 tasks per node, 262144 tasks**
- **1x1x1x32 lattice points per task**
  - Placed entire 4th dimension on each node, mapped remaining 3 dimensions like a 3D virtual topology
- **3.7X faster than default placement**
  - 1.9X faster than when using grid_order –c 2x2x2x2 …

Results on Titan for S3D

- Fluid dynamics w/ combustion
- 3D Virtual topology
- Ran on ~12900 nodes in dedicated mode
- Up to 40% faster than default placement

Results on Blue Waters for VPIC SPP test

- Plasma physics
- 3D virtual topology
- On 2k nodes, this code spends 8% of run time on communication
- Ran on 4608 nodes in dedicated mode
- Best results: 5% faster than default placement

Results on Blue Waters for WRF SPP test

- **Weather forecasting**
- **2D virtual topology**
- **2D domain is folded like a sheet of paper**
  - No tearing – keeps neighbors together – complicates rank ordering
  - Folded in half along one dimension, then 3 times in the other (accordion style) to map 8 super-tiles onto 8 planes of 3D torus
- **Ran on 4864 nodes in dedicated mode**
- **Best results: 3% faster than grid_order placement**

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
  - Input node lists (used by job, etc.) with torus coordinates
- **How do I know which nodes my job ran on?**
  - Use checkjob, as described above
- **How can the program tell which ranks are on which nodes?**
  - I have an example program that does this
  - Makes use of “rca” system library
- **How can I get the torus coordinates from the node IDs?**
  - I have scripts and executables that you can use
  - Makes use of xtdb2proc command
- **What is the best way to contact me?**
  - Email rfiedler@cray.com

CRAY

THE SUPERCOMPUTER COMPANY