Improving Task Placement for Applications with 2D, 3D, and 4D Virtual Cartesian Topologies on 3D Torus Networks with Service Nodes

Robert A. Fiedler and Stephen Whalen
Cray, Inc.
St. Paul, MN, USA
rfiedler@cray.com

***Abstract*— We describe two new methods for mapping applications with multidimensional virtual Cartesian process topologies onto 3D torus networks with randomly distributed service nodes. The first method, "Adaptive Layout", works for any number of processes and distributes the MILC (lattice QCD, 4D topology) workload to ensure communicating processes are close together on the torus. This scheme reduces the run time by 2.7X compared to default placement. The second method, "Topaware", selects a prism of nodes slightly larger than the ideal prism one would select if there were no service nodes. The application's processes are ordered to group neighboring processes on the same node and to place groups of neighbors onto nodes which are no more than a few hops apart. Up to 40% run time reductions are obtained for 2D and 3D virtual topologies. In dedicated mode, using Topaware with MILC reduces the run time by 3.7X compared to default placement.**

Keywords—topology awareness, task placement, torus interconnect

I. INTRODUCTION

We consider the problem of mapping applications that define 2D, 3D, and 4D Cartesian grid virtual process topologies onto Cray systems that have 3D torus networks with service nodes (used for IO and other non-compute functions) randomly distributed among the compute nodes. Since finding a near-optimal mapping for such applications (which perform mainly nearest-neighbor communication) can be difficult even on a dedicated system, simple approaches that are often adopted prescribe rank orders that place separate groups of neighboring processes onto each node [1]. Although this strategy can help reduce off-node communication, it does not ensure that groups of processes that are neighbors in the application's virtual topology are assigned to nodes that are actually near each other on the torus. This can result in layouts with much longer than necessary communication paths that cause a high degree of contention for links between nodes, and can thereby significantly increase communication times compared to a near-optimal layout.

Node selection is very important for minimizing actual communication path lengths on the torus between processes (tasks) that are virtual nearest neighbors [2]. Unfortunately, there is a tradeoff between requiring a near-optimal set of nodes on which to run each application and the overall utilization of the system. In shared batch mode, users typically accept whichever sufficiently large set of nodes is selected by the resource manager to run their job as soon as possible. Moreover, as jobs of different sizes and durations execute and complete, the set of nodes allocated to any new large job is likely to be rather irregular in shape, and it may be fragmented into multiple non-contiguous groups, rather than comprising a regular prism [3]. The presence of service nodes (typically at random locations throughout the system) and the simple fact that there are many nodes in each torus dimension on the largest Cray XE/XK systems can exacerbate this effect. Irregular and (especially) non-contiguous node allocations increase the likelihood that messages may pass through gemini routers that are not attached to nodes in that job's reservation, which in turn makes more probable a significant performance degradation due to job-job interference.

In this paper we describe two new methodologies for placing groups of processes that are nearest neighbors in the virtual Cartesian grid topology onto nearby nodes in a 3D torus interconnect, and compare application performance when using these approaches to runs made using the default placement scheme as well as to runs made using Cray's grid_order tool [1]. We discuss in our concluding section several ways a typical user might obtain the kind of prism-shaped node allocations in shared batch mode that we were able to get easily on a dedicated system. The main objective of this work is to quantify the performance improvement that can be obtained for various applications with different virtual topologies when one can choose which nodes to use, or when one at least has access to a prescribed regular box-shaped section of the torus, and can place tasks on those nodes as desired.

A. Applications Studied

As part of the acceptance testing for Blue Waters, a wide variety of benchmarks representing realistic, complete science problems on 4k to 8k nodes were run to measure the sustained petascale performance (SPP) for the expected workload [4]. The WRF, VPIC, and MILC applications in this benchmark suite define virtual Cartesian mesh task topologies in 2D, 3D, and 4D, respectively, and are candidates for performance improvement through careful node selection and task placement.

In general, it is much more difficult to find a near-optimal placement for virtual topologies whose dimensionality is higher than that of the torus interconnect. When using default placement with applications with 4D virtual topologies on a 3D torus, even in dedicated mode many communication paths are

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. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

long and cross each other. From a network-traffic viewpoint, for any virtual topology, unless neighboring tasks are on nearby nodes in the torus, the communication pattern appears to be between randomly located-pairs rather than among nearest neighbors on the torus. With such poor placement, it would be beneficial to use a node allocation scheme that maximizes bisection bandwidth per node [5]. However, if one can control the node allocation to this extent (i.e., select a regular box-shaped torus section of desired dimensions), one should also be able to place tasks on the nodes so that virtual neighbors are close to each other on the torus.

1) WRF

WRF [6] is a weather prediction application used on Blue Waters to study phenomena such as tornadoes and hurricanes. WRF uses a hybrid MPI/OpenMP parallel programming model. Although the computational domain includes the vertical dimension, each task contains all grid points in that direction, and therefore the nearest-neighbor communication pattern is 2 dimensional. The stencil for approximating spatial derivatives in the evolution equations requires numerous layers of ghost cells around each task's portion of the full grid, which can result in halo exchanges that involve a considerable fraction of the working data set on each task. Therefore, the communication time typically comprises a significant fraction of the total run time.

2) VPIC

VPIC [7] is a space/plasma physics application that uses the particle-in-cell method. The virtual topology is 3-dimensional, and the amount of data involved in its halo exchanges is limited by the number of particles that move from one task's portion of the computational mesh to another task's portion in a single time step, which is typically only a small fraction of the total number of particles in the system. Using default node allocation and task placement on 2k nodes, the communication time is only ~8% of the total run time, and therefore communication times at such scales would have to improve significantly in order to have a noticeable impact on the total run time.

3) S3D

S3D [8], which uses a 3D virtual topology and simulates fluid flows with combustion, was not part of the NCSA acceptance test suite, but it was an acceptance benchmark for the Titan XK7 system at Oak Ridge National Laboratory. S3D was also run at larger scales on Titan than were most of the SPP benchmarks on Blue Waters. For most applications at larger scales, the benefits of near-optimal node selection are expected to become more apparent [9].

4) MILC

MILC [10] is a quantum chromodynamics application whose 4D lattice includes 3 spatial dimensions plus time. Our initial benchmark runs showed that the halo exchanges took much longer than expected from our performance model and dominated the run time. We and others [11] have observed that this application is extremely sensitive to placement and job-job interference on Cray systems with 3D torus interconnects, presumably due to the difficulty in keeping 4D virtual nearest-neighbors together, even on a regular prism of geminis in a 3D torus.

B. Gemini Interconnect

Implementing schemes for near-optimal node selection and task placement requires a basic understanding of the interconnect in question. Here we describe the key aspects of the Blue Waters Cray XE6/XK7 system [12].

The Blue Waters interconnect is a 3D torus with 23 gemini [13] routers in the x direction, 24 in y, and 24 in z, with two nodes attached to each gemini. There are 3072 XK7 compute nodes with 8x8x24 geminis embedded in this fabric. Each XK7 node has one 2.3 GHz AMD Interlagos processor with 8 "Bulldozer" compute units and one nVidia Kepler GPU. The rest of the ~22752 compute nodes are XE6, each with two Interlagos processors like the one in the XK7 nodes. There are also ~672 service nodes in various locations throughout the torus that perform functions such as IO, job launching, etc. The service nodes are not directly allocated to user jobs, but they do relay messages between compute nodes on behalf of user jobs.

User jobs are assigned on a per-node basis, i.e., different batch jobs do not share nodes. For optimum performance, it is best if a given job is running on both nodes attached to each gemini in the batch job reservation. If one of the nodes on a gemini is down or assigned to another job, a load imbalance is likely to result. A user can avoid this situation by requesting more nodes than are needed to run the job and using our placement tools or other means to avoid actually running on such nodes. On Blue Waters, another advantage of having a few extra nodes in the allocation is that they could be used to continue a run within the same batch job in the event of a node failure. When the run restarts, a new optimal layout of the same size can be obtained by using one or more of the previously idle nodes.

Although the two nodes attached to a given gemini use the router to exchange messages, that traffic does not traverse the links between geminis, and therefore is not considered to use the interconnect. We estimate that the application-realizable All-to-All bandwidth between same-gemini nodes ~ 12 GB/s [14], which is greater than the bandwidth of any of the individual links to neighboring nodes.

Links in the x direction of the torus consist of cables connecting different rows of cabinets. The z direction runs across the 24 boards in any given cabinet. Each cabinet has 3 cages containing 8 boards on a backplane with higher bandwidth than the cables connecting the backplanes together. These cables are the same capacity as those used for the x direction, and they determine the effective bandwidth across any set of nodes that spans more than one cage. In the y direction, the connections between different boards have only half the bandwidth of the cables used in the x and z directions. There are two geminis on each board, and the bandwidth between geminis on the same board is much higher than the bandwidth along y between boards. Good task placement strategies take advantage of the faster links in the x and z directions in order to reduce communication times for a given node count.

II. STRATEGIES FOR REDUCING COMMUNICATION TIMES

A. Tile Size Selection

Often the application user has some degree of control over the problem decomposition, namely, the per-node or per-gemini grid can be sized to take advantage of faster links along the x and z dimensions of the gemini interconnect. For example, consider an application that performs the same amount of communication per grid point on the surface of each per-task grid in halo exchanges along each dimension in a 3D virtual topology. In this case, the communication time for any dimension is proportional to the number of surface grid points divided by the link bandwidth. On a torus with identical link speeds in all directions, halo exchange communication times can be minimized by minimizing the surface-to-volume ratio of the group of partitions on each node pair, and so the most efficient configuration occurs for groups of partitions forming a perfect cube of grid points on each node pair.

The asymmetry of the gemini interconnect makes cube-shaped grids on each node pair less than optimal. Instead, the per-node-pair grid should have 2X less surface area normal to y than the surface areas normal to x and to z in order to make the communication times equal for each dimension. This is particularly important if the halo exchanges are performed in all dimensions at once, rather than in only one dimension at a time.

To quantify the above assertion, suppose the per-node-pair computational grid has M_x by M_y by M_z points. If the halo-exchange communication is performed in all three dimensions at once (and all messages are actually passed at the same time by the interconnect), then the total communication time T_{comm_tot} would be:

$$T_{comm\_tot} = \max(M_yM_z/B_x, M_xM_z/B_y, M_xM_y/B_z),$$

where B_x is the bandwidth of an x link, and so on for y and z. If the per-node-pair grid is cubic, so that it has M points in each dimension, then T_{comm_tot} equals the communication time for the y dimension:

$$T_{comm\_tot} = M^2/ B_y = 2M^2/B_x.$$

If we reshape the per-node-pair grid, keeping the total number of points equal to M^3, but setting $M_x = M_z = M_y/2$ so that the surface normal to y has half the area of the surfaces normal to x and to z, then $M_x = M/2^{(1/3)}$ and T_{comm_tot} is reduced by a factor of $2^{(2/3)} \sim 1.6$.

Again, the above analysis assumes that the interconnect perfectly overlaps the passing of messages in all three dimensions. This requires sufficiently large messages so that the non-blocking sends/receives actually do not block in practice (i.e., large enough for the Block Transfer Engine to be used), and the number of senders needs to be 8 or less (e.g., using the MPI/OpenMP programming model to limit the number of communicating tasks per node).

Now suppose the halo exchanges are performed one dimension at a time. For a cubic per-node-pair grid, since $B_y = B_x/2$ and $M_x = M_y = M_z = M$:

$$T_{comm\_tot} = M_yM_z/B_x + M_xM_z/B_y + M_xM_y/B_z = 4M^2/B_x.$$

For the reshaped per-node-pair grid, communication for all three dimensions takes the same amount of time, and therefore

$$T_{comm\_tot} = 3(2M_x^2/B_x) = [6/2^{(2/3)}]\, M^2/B_x.$$

Thus, when performing halo exchanges one dimension at a time, the total communication time for the reshaped per-node-pair grid is only ~ 1.06X shorter than it is for a cubic per-node-pair grid.

B. Placing Groups of Neighbors on the Same Node

The Cray Programming Environment offers two useful tools for placing groups of neighboring tasks on the same node: the Craypat performance tool suite and the grid_order tool [1]. When used to profile an MPI application, Craypat tries to detect Cartesian grid communication patterns. For detected grid patterns, Craypat creates a rank order file called MPICH_RANK_ORDER, which can be used at run time by setting the MPICH_RANK_REORDER_METHOD environment variable to 3 (for custom ordering). This rank order is suitable for running the application on the same number of nodes that were used when the profiling was done, provided the communication pattern and decomposition remain the same. Craypat also computes the fraction of on-node communication for the generated rank order as well as other rank order options and estimates how much of a difference the custom order will make in the communication time.

The grid_order tool generates custom rank orders based on user-provided specifications pertaining to the virtual topology and desired per-node task layout. This tool enables a user who knows the application virtual topology to generate a rank order that places neighboring tasks on the same node without first having to make a Craypat run for that node count. It also enables the user to more quickly try different per-node task layouts (other than the one suggested by Craypat, for example) in a sometimes fruitful effort to find a more optimal one.

We have successfully used both Craypat and grid_order to improve communication times, and highly recommend them to users as a first step in obtaining better scaling. No code changes are required to use the rank orders they generate, and performance is often somewhat better than it is when using the default placement (which puts groups of consecutive tasks on each node), even if the job's node allocation has a less than optimal geometry. Another advantage these tools offer is that they can be used with somewhat irregular topologies, such as a "cubed-sphere" grid (an unstructured mesh with quadrilateral elements used in the SPECFEM3D_GLOBE [15] seismology code and in some climate/weather applications). For the SPP SPECFEM benchmark, Craypat was able to detect the predominant underlying communication pattern even though it does not apply to all tasks [16].

C. Adaptive Layout

MILC can use a variety of domain-decomposition geometries, which the developers refer to as layouts, for the several applications supported by that software package. The default layout for the Blue Waters benchmark decomposes the 4D lattice into uniformly-sized hypercubes, and assigns one such hypercube to each task. However, decompositions with

partitions of varying size are allowed, which enables us to implement a task placement strategy specifically for MILC that we refer to as "Adaptive Layout" (AL). AL creates a layout that adapts itself automatically to the geometry of the nodes allocated to the job in a manner that balances the workload among tasks while keeping communicating partners nearby in the 3D torus.

AL begins by using the Cray RCA library interface to determine the maximum extent of the set of nodes allocated to the job in each dimension of the torus, and the maximum number of tasks per node. We label these extents S_x, S_y, S_z, and S_t, respectively. The layout then acts as though the tasks completely fill a grid of size S_x by S_y by S_z by S_t, and distributes each of the four lattice dimensions over these extents, whether or not this task grid evenly divides the lattice size. The result is a possibly non-uniform lattice decomposition, wherein each lattice block dimension may vary by 1 from block to block. The t dimension fits entirely on each node, and the remaining 3 spatial dimensions are treated as a 3D virtual Cartesian mesh topology.

At this point, more lattice blocks exist than there are actual tasks to claim them: some blocks may correspond to torus coordinates lying outside the allocation bounds, and some blocks may correspond to locations of service nodes. Any block whose position in the S_x by S_y by S_z by S_t grid corresponds to the location of an actual application task is assigned to that task. Unassigned blocks are then assigned to tasks that own neighboring blocks.

Searching for new task locations for unassigned blocks is done preferentially in the x and z directions. If a block location corresponds to a service node, it is likely that three adjacent blocks along the y direction will correspond to the other service nodes on that service blade. One is therefore more likely to find nearby neighbors in the x and z directions.

To reduce the severe load imbalance implied by this strategy, unclaimed blocks are divided, and the sub-blocks are assigned to different neighboring tasks. The specific strategy that was found to perform best for the Blue Waters benchmark subdivided each lattice block into four sub-blocks along their y dimensions only. The four sub-blocks were then dealt to the four nearest processes in the ±x and ±z directions. Such a subdivision results in tasks having fewer communication partners than would result from subdividing in either of the x or z directions. A final rebalancing step identifies those tasks holding the largest numbers of blocks, and reassigns a block from each of these overloaded tasks to another nearby process that has a lighter load.

The implementation of this layout strategy is written in UPC, using a hierarchy of shared arrays of S_x by S_y by S_z by S_t integers to store the MPI rank that owns each block. The layout algorithm executes fully in parallel, with the various application tasks searching for unclaimed sub-blocks in such a way that no two tasks will be able to claim the same sub-block simultaneously. Layout creation completes in just a few seconds for up to 68,528 tasks.

D. Topaware

The "Topaware" node selection and task placement tool provides a method for selecting a near-optimal set of nodes for jobs with a known virtual topology, taking into account the presence of unavailable nodes (IO nodes, MOM nodes, failed nodes, nodes in use by other jobs, and compute nodes not of the desired type) in the torus. Topaware generates a list of nodes on which the job is to run. The set of node pairs that Topaware selects is roughly a regular prism, but with a somewhat bumpy surface normal to z and possibly to x due to unavailable nodes in the interior. Fig. 1 shows the set of 2058 node pairs selected to run the MILC benchmark, for which the maximum distance between virtual nearest-neighbors as actually placed on the torus is just 3 hops.

The Topaware user specifies the number of node pairs to use along each torus dimension (N_x, N_y, N_z), as well as the number of partitions in each virtual dimension to place on each node pair. Topaware finds nodes to use by first selecting starting values for the x, y, and z coordinates in the torus. As depicted schematically in Fig. 2, in the starting XZ plane, beginning at the starting x coordinate, Topaware counts available compute node pairs along the z direction. If there are at least N_z compute node pairs along z for that value of x, then that x value is a candidate for inclusion in the node pair list. Topaware advances to the next value of x and again counts available node pairs along z. This process is repeated until either every x value in that XZ plane has been examined or the desired number of nodes (N_x N_z) has been found in that XZ plane. If Topaware finds sufficiently many available compute nodes, then the nodes in that XZ plane are to be included in the node pair list. Topaware then advances to the next XZ plane and repeats the compute node counts. If it finds N_y XZ planes to be included in the node pair list, it declares success and generates the node list.

By construction, Topaware selects a set of node pairs with the desired numbers of compute nodes along z at every x value, and the desired number of node pairs in each XZ plane.

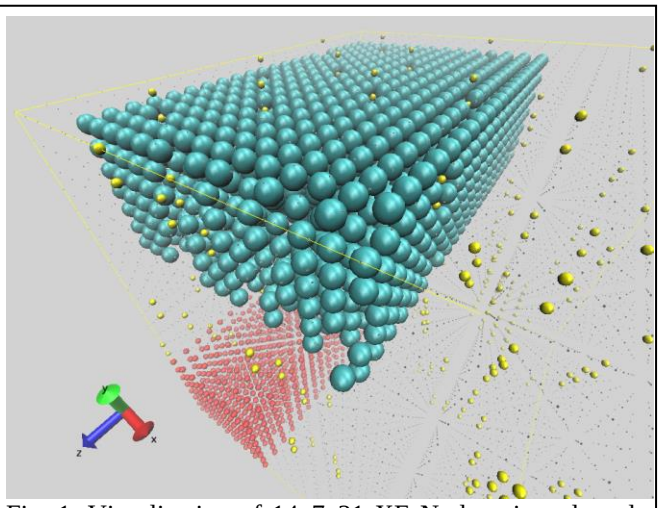

Fig. 1. Visualization of 14x7x21 XE Node pairs selected by Topaware for MILC (cyan spheres) on the Blue Waters interconnect. The red spheres represent XK hubs and the yellow spheres represent service nodes.

Although the selected set of node pairs has some internal "dislocations" due to the presence of unavailable nodes, and these dislocations add a small number of extra hops for many of the communication paths, the longest communication path in the selected node set is never more than a small number of hops (e.g., in Fig. 2, 3 hops between pair 41 and 34). Moreover, as we request larger numbers of node pairs for larger jobs, the longest path length increases little, if at all, and therefore the increase in the communication time due to contention grows very slowly with the number of nodes. As a result, jobs run on nodes selected by Topaware should scale nearly as well as if there were no service nodes in the system.

Topaware may occasionally have to skip an x value in an XZ plane if N_z available compute node pairs are not found along z for that x value, as in the 6th row of Fig. 2. This could leave dozens of nodes idle in the interior of the allocation. Rarely, Topaware may have to skip one or two entire XZ planes, which could leave hundreds to thousands of nodes idle in the interior of a large allocation. Doing so in shared batch mode would invite job-job interference if other jobs are placed on these idle nodes. This problem can usually be avoided either by changing the starting value of y (via an environment variable), or by reducing the number of requested compute node pairs along one or more dimensions of the torus.

After the node list is generated, Topaware creates a custom rank order file to place the job tasks onto the nodes in the list as directed by the user in the Topaware command line arguments. Some experimentation may be required to obtain the best possible performance for a particular application and input deck. For example, if the virtual topology is 3D, one specifies that each node pair gets n_x by n_y by n_z partitions, and also which dimension is to be divided between the two nodes on the same gemini. This can allow a more optimal per-node-pair task placement than the grid_order tool provides, but that requires Topaware to avoid using compute nodes whose partner on the same gemini is unavailable.

The rank order generated by the placement tools must be consistent with the virtual topology in the application. In particular, Topaware and grid_order need to know (through environment variables or command line arguments) which virtual topology dimension changes fastest with increasing rank. If the first (leftmost) dimension changes fastest, the ordering is called "Row major", and if the last (rightmost) dimension changes fastest, the ordering is called "Column major" (as with Fortran and C language array memory layouts, respectively).

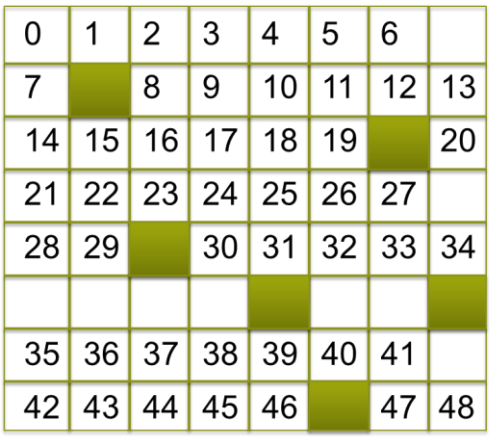

Fig. 2. Topaware hub selection for 7x7 hubs in an 8x8-hub XZ plane. The z axis points to the right. Green squares represent service nodes, and blank white squares represent available compute nodes that are not selected by Topaware for the specified hub layout.

If the nearest-neighbor communication is periodic along any given dimension, it is best to use all geminis in that dimension. Otherwise, if the node allocation spans up to half of the geminis in a periodic dimension, traffic from the geminis on opposite surfaces must pass through all geminis in the interior, sharing bandwidth with the rest of the nearest-neighbor pattern. If the node allocation spans more than half of the geminis along that dimension, then communication between geminis on opposite surfaces will wrap around the torus through geminis that are not assigned to the job, potentially impacting (and being impacted by) other jobs using those nodes. The worst case occurs when a periodic dimension in the virtual topology aligns with the y dimension of the torus, since those links have half the bandwidth of the links in the other two torus dimensions and are therefore more often driven at full capacity by the interior traffic of each application.

1) 2D Virtual Topologies

Topaware maps 2D virtual topologies onto a 3D torus by dividing the virtual domain into N_1 by N_2 "super-tiles" (as specified by the user) and placing each super-tile onto a different plane of the torus. The resulting prism has $N = N_1 N_2$ planes, each of which has just enough node pairs for one super-tile. In order to keep communication paths short, it is important that the 2D domain is placed onto the torus without "tearing" it along any of the super-tile boundaries. Instead, the 2D domain is folded like a sheet of paper [2] accordion style, first across the dimension with the fewest super-tiles, and then in the other dimension, as depicted in Fig. 3. The rank order Topaware generates reverses direction at each fold to keep neighbors together.

Folding the 2D domain along both dimensions often leads to super-tiles (and therefore per-node grids) that are more nearly square than those resulting from folding along only one dimension. The main disadvantage of folding along both dimensions is that some super-tiles that should be neighbors have 2 or more other super-tiles between them. Not only are the communication paths longer, but some links are used for

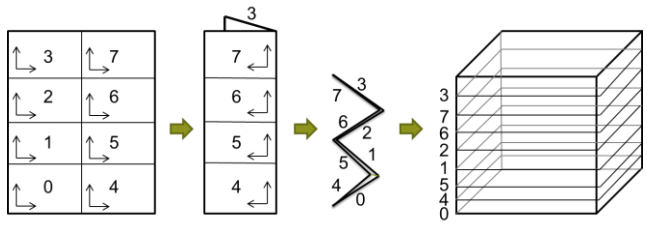

Fig. 3. Mapping a 2D virtual topology to a 3D torus by dividing the domain into 8 "super-tiles" and folding once in z and then 3 times in x. The ordering of the super-tiles as stacked onto the 3D torus is indicated (far right).

communication between multiple pairs of super-tiles. Since only the links on super-tile edges are used at all, it should be advantageous to have Topaware displace pairs of super-tiles sharing the same links along their surfaces by one gemini to avoid contention. For example, in Fig. 4 we displaced super-tiles 4 & 5 (and 6 & 7) with respect to the remaining super-tiles, so that the communication between super-tiles 0 & 1 (and 2 & 3) uses a different set of y links than do the displaced super-tile pairs. However, this displacement technique was not yet implemented for the runs described below for WRF.

2) 4D Virtual Topologies

Topaware places 4D virtual topologies onto a 3D torus by mapping one of the 4 dimensions entirely onto each node-pair. While this does not minimize the amount of off-node traffic, it does enable us to treat the remaining 3 dimensions in the manner described above for 3D virtual topologies. As a result, the longest communication paths are much shorter than they are when one does not specify a node list (e.g., when using grid_order), and the reduction in congestion significantly outweighs the cost of the extra off-node traffic.

III. RESULTS

A. MILC

Off-node communication for MILC can be reduced by a factor > 2 using the grid_order tool to generate a rank order that puts 2x2x2x2 blocks of neighboring tasks onto each node. The benchmark on 4116 nodes with an 84x84x84x144 global lattice results in 6x6x6x6 lattice points per task. Using this rank order improves the run time by 1.9X compared to using default placement on a dedicated system. Using grid_order in shared batch mode is as easy and often as effective as using it in dedicated mode, although widely varying run times have been observed due to the shape of the node allocation and interference from other jobs on the system.

Given a dedicated 23x4x24 set of geminis (in a node list or in a set of reserved nodes in a list generated by the cnselect command, for example), using the Adaptive Layout scheme reduces the overall execution time by 2.7X compared to default placement. Some of the speedup of this run compared to the run with the grid_order rank order and default node allocation undoubtedly derives from the shape of this node allocation, since it has optimal bisection bandwidth per node (up to 2X higher than a typical default node allocation, even in dedicated mode [5]). We did not measure separately the impact of this node allocation for the grid_order rank order, however.

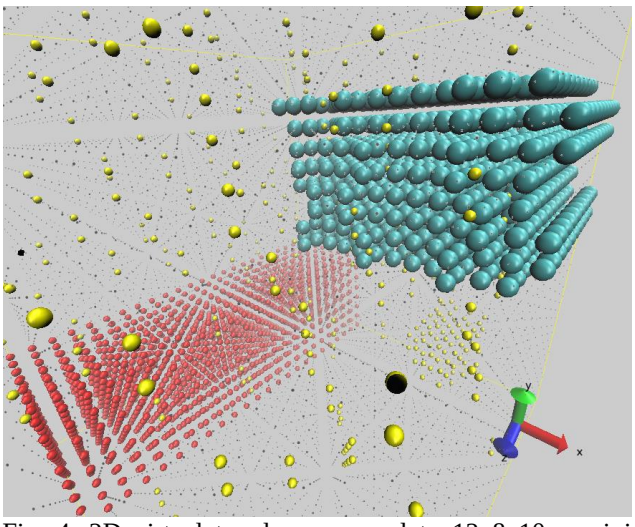

Fig. 4. 2D virtual topology mapped to 12x8x10 gemini section of torus with 8 super-tiles staggered in the x direction to avoid overloading y links at super-tile edges.

Using Topaware with MILC, we specified that it should run on 14x7x21 geminis (node pairs) with 1x2x1x16 tasks per node pair. We assigned 1x2x1x8 tasks to each node, and the per-task lattice was 6x6x4x9. Note that there are 144 lattice points in the fourth (time) dimension on each node, so that no off-node communication needs to be done for this dimension. We rely on having both nodes on each gemini available to us for running MILC. Each node pair has only 1 partition in x and z, and 2 in y, compared to 2 in each dimension for grid_order, and therefore the Topaware layout has more off-node communication. The benefit from the near-optimal layout for the 3 spatial dimensions far outweighs the extra communication, since using Topaware results in an overall run time reduction of 3.7X compared to default placement (i.e., 1.9X shorter total run time than using grid_order with a default node allocation in dedicated mode).

B. VPIC

The VPIC SPP benchmark ran on 4608 nodes in dedicated mode. The computational grid was 1536^3, with 48x48x32 partitions in the 3D virtual topology. We used Topaware to place the 73,728 MPI tasks onto 12x12x16 geminis, which made the per task grid 32x32x48. Each node pair had 4x4x2 partitions (2x4x2 partitions per node). The overall run time was reduced by 5% compared to the default placement in dedicated mode. This improvement is relatively small because the communication time for default placement was only a modest fraction of the total run time.

Further reduction in the communication time could be expected for this input deck by orienting the z dimension of the virtual topology along the y dimension of the torus, so that the dimension with the least amount of communication would use the y links. However, this capability was not yet implemented in the Topaware 3D rank order generation routine when the benchmark results were due.

C. S3D

Craypat-style rank reordering was used successfully with S3D on hopper at NERSC, but the reported performance gains were a modest 4% [17]. S3D was run on Titan in weak scaling studies (fixed work per task) using from 2000 to 12900 nodes [18]. For default node selection, run times increased significantly with the number of nodes. When Topaware was used, near-linear weak scaling was obtained on up to 12900 nodes. On 2000 nodes, the run with Topaware placement completed in 1.32X less time than the run with default placement, and on 6000 nodes, the Topaware run was 1.61X

faster. A comparable run with default placement on 12900 nodes was not made.

D. WRF

For WRF, one benchmark problem has a 6075x6075 cell grid in longitude and latitude. For this application on 4560 nodes with 16 MPI tasks per node, Craypat indicates that with the default placement, 40% of the communication goes off node. If Craypat's custom rank order with 2x8 partitions per node is used, only 20% of the communication goes off node. The total run time using the custom rank order is 1.18X shorter than a run using the default rank order [19]. Since the communication time should be reduced by 2X using the custom rank order, the communication time for the custom rank order must be about 18% of the total run time.

A near-optimal layout for this benchmark on 4864 nodes was obtained using Topaware. The 2D virtual topology was divided into 8 super-tiles, and each super-tile was placed on 16x19 geminis in an XZ plane of the torus. The domain was folded in half along z, and then 3 times in accordion fashion in x to arrange the 8 super-tiles along the y torus dimension as in Fig. 3. There were 6x5 partitions on each node pair (3x5 per node), leaving one idle compute module per Interlagos processor. The Cray "core specialization" feature [20] was used to assign OS-related tasks to the idle compute modules, which would reduce any load imbalance caused by system interrupts occurring on cores running WRF. Since this run used more nodes than the Craypat rank order run, we compare performance in terms of the number of sustained GFLOPS/node, which was 3% higher for the run using Topaware placement. The use of core specialization helped reduce communication times, possibly by reducing the impact of Lustre pings [21], while the squarer aspect ratio of the per task grid (compared to the Craypat rank order run) appears to increase the computational work time due to WRF's loop structure and our use of 2 OpenMP threads per MPI task. We believe that better efficiency for the Topaware placement could be obtained by displacing pairs of super-tiles that share links, as described earlier.

IV. Conclusions and Future Work

We presented several strategies for reducing the communication time on 3D torus interconnects with asymmetrical link speeds and randomly distributed service nodes for a number of applications that perform nearest-neighbor communication within multidimensional Cartesian grid virtual topologies. If the users have no control over the set of nodes allocated by the system for their jobs other than the number and their type (e.g., XE vs. XK), they can reduce off-node communication by using Craypat or the grid_order tool to group tasks that are virtual neighbors onto the same node. We obtained significant reductions in off-node communication for several Blue Waters SPP applications without making any changes to the codes. In addition, users can take advantage of the faster links in the x and z dimensions of Cray genimi networks by carefully choosing the aspect ratio of a per-node or per-node-pair chunk of the domain such that the communication time for each dimension is nearly equal, in order to take advantage of the faster x and z links. While these strategies often improve communication performance somewhat in practice, they are incapable of ensuring that neighboring groups of tasks are placed onto nearby nodes in the torus, and therefore often fail to improve performance to the level expected from theoretical models based on link speeds and the number and sizes of messages on each link

If users are able to specify the nodes on which their jobs are to run and the communication pattern in their application has a 2D, 3D, or 4D Cartesian virtual topology, then they may be able to place tasks that should be neighbors onto nodes that are close together in the torus, either by developing an application-specific topology-aware module in the spirit of Adaptive Layout for MILC, or by using Topaware (without any changes to the application). While Topaware provides the best possible performance for MILC, Adaptive Layout can be used with fairly regular prism-shaped batch job node allocations that do not conform well to the virtual topology, and it should enable significantly better MILC performance than Craypat/grid_order on such allocations.

Topaware can also be used in principle in shared batch mode. The most straightforward way of doing so would be to submit the job with the restriction that it must run on the specific nodes selected by Topaware, and then let it wait in the queue until all of the required nodes become available. One could make this strategy more palatable by requesting a reservation restricted to the nodes in the list at a particular time in the future, so that one can plan to be ready to monitor the job when it runs. Another approach would be for the site to create two or more separate node pools with desirable shapes, so that "right-sized" jobs requiring placement by Topaware could request all nodes in the appropriate pool and select a prism of node pairs from that pool. Any resulting reduction in system utilization under those circumstances could be mitigated to some extent by the improved performance of those jobs requiring careful placement.

In the near future, we plan to extend Topaware to place neighboring groups of tasks onto nearby nodes in the torus for the set of nodes assigned to the job by the resource manager. Given the virtual topology, Topaware would for example place ranks with larger virtual y values onto nodes with larger y values in the torus, and then use a space-filling curve to ensure a reasonably good layout for the x and z dimensions, at least for a reasonably regular prism of nodes. This approach would reduce the lengths of the longest communication paths (especially in the y torus dimension) and therefore reduce the contention for certain links, but it would not be able to minimize contention to the extent possible for a list of nodes generated by Topaware for a specified prism of geminis. This approach would be less general than that of Hoefler and Snir [9], but it would not be limited like the current Topaware implementation to Cartesian virtual topologies.

References

[1] Cray, Inc., "Using Cray performance measurement and analysis tools", docs.**cray**.com/books/S-2376-60//S-2376-60.pdf, 2012.

[2] H. Yu, I.-H. Chung, and J. Moreira, "Topology mapping for Blue Gene/L supercomputer", in SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, page 116, New York, NY, USA.

[3] Carl Albing, Norm Troullier, Stephen Whalen, Ryan Olson, Joe Glenski, Howard Pritchard and Hugo Mills, "Scalable Node Allocation for Improved Performance in Regular and Anisotropic 3D Torus Supercomputers", Lecture Notes in Computer Science, Volume 6960, Recent Advances in the Message Passing Interface, Pages 61-70, 2011.

[4] G. Bauer, T. Hoefler, W. Kramer, and R. Fiedler, "Analyses and Modeling of Applications Used to Demonstrate Sustained Petascale Performance on Blue Waters", CUG 2012, April 29 - May 3, 2012, Stuttgart, Germany,

https://cug.org/proceedings/attendee_program_cug2012/includes/files/pap168.pdf.

[5] R. Fiedler, N. Wichmann, S. Whalen, and D. Pekurovsky, "Improving the performance of the PSDNS pseudo-spectral turbulence application on Blue Waters using coarray Fortran and task placement", CUG 2013, May 6-9, 20123 Napa, CA, USA.

[6] W. C. Skamarock, et al, "A description of the Advanced Research WRF version 3", NCAR Technical Note TN-475+STR, June 2008, http://www.mmm.ucar.edu/wrf/users/docs/arw_v3.pdf.

[7] K. J. Bowers, B. J. Albright, L. Yin, W. Daughton, V. Roytershteyn, B. Bergen, and T. J T. Kwan, "Advances in petascale kinetic plasma simulation with *VPIC* and Roadrunner", *J. Phys.: Conference. Series* **180** 012055, 2009.

[8] J. M. Levesque, R. Sankaran, and R. Grout, "Hybridizing S3D into an exascale application using OpenACC", SC12, November 10-16, 2012, Salt Lake City, UT, USA,

http://conferences.computer.org/sc/2012/papers/1000a040.pdf.

[9] T. Hoefler and M. Snir, "Generic Topology Mapping Strategies for Large-scale Parallel Architectures",CS'11, May 31 – June 4, 2011, Tuscon, Arizona, USA,

http://www.unixer.de/publications//img/hoefler_snir_topology_mapping.pdf.

[10] MIMD Lattice Computation (MILC) Collaboration code page, *http://www.physics.utah.edu/~detar/mil*

[11] D. Wang, "Application performance variability on Hopper", NERSC user Web page, 2012, http://www.nersc.gov/users/computational-systems/hopper/performance-and-optimization/application-performance-variability-on-hopper.

[12] Blue Waters main Web page, http://www.ncsa.illinois.edu/BlueWaters/.

[13] R. Alverson, D. Roweth, and L. Kaplan, "The gemini system interconnect," in International Symposium on High Performance Interconnects, Aug. 2010, pp. 83 -87.

[14] Nathan Wichmann, Cray, Inc., private communication, 2012

[15] L. Carrington, et al,. "High-Frequency Simulations of Global Seismic Wave Propagation Using SPECFEM3D_GLOBE on 62K Processors", SC08 Nov. 15-20, 2008, Austin, TX, USA, http://www.sdsc.edu/~allans/specfem3D_161TF.updated.pdf.

[16] Sarah Anderson, Cray, Inc., 2012, private communication.

[17] Y. He and K. Antypas, "Running large scale jobs on a Cray XE6 system, CUG 2012, April 29 - May 3, 2012, Stuttgart, Germany, https://cug.org/proceedings/attendee_program_cug2012/includes/files/pap162.pdf

[18] R. Sankaran, Oak Ridge National Laboratory, 2012, private communication.

[19] Peter J. Johnson, Cray, Inc., 2012, provate communication.

[20] H. Pritchard, D. Roweth, D. Henseler, and P. Cassella, "Leveraging the Cray Linux environment core specialization to realize MPI asynchronous progress on Cray XE systems", CUG 2012, April 29 - May 3, 2012, Stuttgart, Germany,

https://cug.org/proceedings/attendee_program_cug2012/includes/files/pap115.pdf.

[21] C. Sptiz, N. Henke, D. Petesch, and J. Glenski, "Minimizing Lustre ping effects at scale on Cray systems",

https://cug.org/proceedings/attendee_program_cug2012/includes/files/pap166.pdf.