Topology-Aware Job Scheduling Strategies for Torus Networks

J. Enos, G. Bauer, R. Brunner, S. Islam
NCSA, Blue Waters Project
University of Illinois
Urbana, IL USA
e-mail: jenos@illinois.edu

R. Fiedler
Cray, Inc.
Seattle, WA USA
e-mail: rfiedler@cray.com

M. Steed, D. Jackson
Adaptive Computing
Provo, UT USA
e-mail: jacksond@adaptivecomputing.com

***Abstract*—Multiple sites having Cray systems with a Gemini network in a 3D torus configuration have reported inconsistent application run times as a consequence of task placement and application interference on the torus. In 2013, a collaboration between Adaptive Computing, NCSA (Blue Waters project), and Cray was begun, which includes Adaptive's plan to incorporate topology awareness into the Moab scheduler product to mitigate this problem. In this paper, we describe the new scheduler features, tests and results that helped shape its design, and enhancements of the Topaware node selection and task placement tool that enable users to best exploit these new capabilities. We also discuss multiple alternative mitigation strategies implemented on Blue Waters that have shown success in improving application performance and consistency. These include predefined optimally-shaped groups of nodes that can be targeted by jobs, and custom modifications of the ALPS node order scheme.**

I. INTRODUCTION

Large, massively parallel supercomputers with multidimensional torus interconnection networks have been around for many years, including some of the most capable systems in production today (Cray XT/XE/XK, Blue Gene P/Q). The largest Cray XT/XE/XK systems including Blue Waters and Titan have 3-dimensional torus interconnects in which most of the nodes are devoted to running user jobs (i.e., compute nodes), but some nodes are service nodes performing various duties including IO, MOM, boot, internal login, etc. (see [1]). Fig. 1 shows the current Blue Waters interconnect topology, which consists of 24 by 24 by 24 gemini routers. The service nodes are depicted as yellow spheres, the XK nodes by red spheres, and the XE nodes by small gray spheres. Since the torus wraps around each dimension, the XK region is in reality a contiguous block of 15 by 6 by 24 geminis. It is well known that as the size of systems with such interconnects increases, if jobs are scheduled without regard to topology considerations, the run time for an application that performs a fixed amount of work on a specified number of nodes can vary significantly from one batch job to the next due to the locations of the nodes allocated to run the job, as well as contention for interconnect resources from other jobs on the system. Jobs that spend a greater fraction of their run time on communication tend to exhibit the most variability, especially at larger scales [2]. After startup on a large system whose scheduling policy allows non-contiguous node allocation strategies, as many different-sized jobs of different length run and complete, the sets of nodes allocated to new jobs tend to become more fragmented (i.e., each new large allocation consists of many non-contiguous groups of nodes scattered throughout the system, increasing contention for

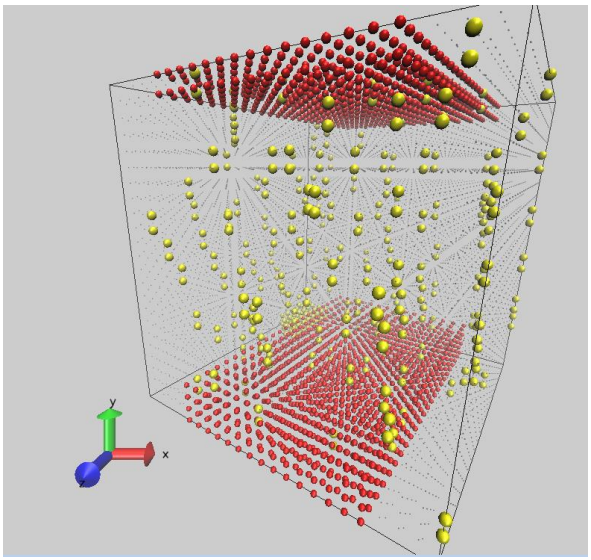

Figure 1. Blue Waters service geminis (yellow spheres) and XK geminis (red spheres).

bandwidth on links between nodes [3].

The Cray ALPS software provides the scheduler with a list of the compute nodes in an order designed to help improve bisection bandwidth while reducing hop counts compared to allocations selected from a node list with torus x,y,z ordering or random ordering [4]. While the ordering used with most large XE/XK systems today helps improve application performance somewhat, it does not do enough to ensure that allocations are contiguous, nor does it use information about an application's virtual process topology to provide an optimal layout on the torus. On the other hand, the current ALPS node ordering scheme has little impact on utilization, since its node order does not depend on whether the compute nodes are idle or busy running jobs.

Efforts to determine an optimal layout of an application's tasks onto a 3D torus network are stymied by poor allocation shape. The libTopoMap tool of Hoeffler, et al, is capable of reducing communication times by as much as 45% when mapping irregular communication graphs onto (idle) 3D torus networks [5], but much larger (1.9X) improvements in overall application run times have been demonstrated for 4D Cartesian mesh virtual topologies (e.g., for MILC [6], a lattice QCD application) when the Topaware node selection and task placement tool is utilized on a dedicated system [7]. Such large communication performance improvements require compact rectangular prism-shaped allocations, and therefore enhancements to the scheduler that provide prism-shaped node allocations are required in order to make using Topaware practical in a production environment.

Compact prism-shaped allocations help reduce contention, but inevitably impact system utilization to some degree, since jobs must wait longer for contiguous blocks of nodes to become available. An important goal in this work is to develop an interconnect-topology-aware scheduler that produces very favorable node allocation shapes, which improve application performance at least enough to compensate for any decrease in utilization and thereby increase overall system throughput. According to [8], for realistic workloads, if contiguous allocations are enforced, the applications need to run about 25-30% faster in order to make up for the reduction in system utilization.

The links along the x and z dimensions on gemini networks are at least 2X faster than the slowest links in the y dimension [9]. This has implications for which prism shapes are best in terms of bisection bandwidth per node. In particular, slabs of gemini hubs of aspect ratio 2:1:2 maximize bisection bandwidth per node and equalize bandwidth along each of the 3 torus dimensions, provided no slab dimension spans more than half the nodes in that direction on the torus.

Torus wrap-around also can play a role in determining the best prism shapes, given the routing scheme (shortest path in x, then y, then z) and the torus dimensions. If a prism of geminis spans more than half of the torus along a given dimension, messages between geminis on either end of the prism along that dimension will pass through nodes outside of the prism to reach their destinations. Thus, to avoid contending for bandwidth on links outside of a prism-shaped node allocation, the prism should span either less than or equal to half of the full torus dimension, or else it should span the full torus in any given dimension. The largest allocation size with optimal bisection bandwidth per node on Blue waters has 24 by 6 by 24 geminis. If a job requires more than half but less than all of the nodes on the system, some application communication is likely to traverse links outside the prism. For more discussion of considerations relating to the bisection bandwidth per node, see [10].

II. Run Time Consistency and Mitigation Strategies

A. *Application Run Time Variability*

Early in the production phase of the Blue Waters project, large run-to-run variations in performance were observed for a number of applications, particularly those whose communication times comprise a significant fraction (i.e., of order 50%) of their overall run time. One such application is PSDNS, a Computational Fluid Dynamics application that uses pseudo-spectral methods, requiring frequent global array transposes for 3-dimensional FFTs [11]. The most time-consuming communication pattern in PSDNS consists of 16 concurrent All-to-All operations, each of which involves sets of M tasks, where M is the number of nodes in the job. Each of the 16 tasks on a node participates in a different All-to-All operation.

PSDNS wall clock times for one time step have been observed to vary by more than a factor of two within a single batch job, and even larger variations (on the order of 4X) were noted from one batch job to the next. Experiments on a dedicated system indicated that PSDNS communication performance is sensitive to node allocation shape, favoring those with higher bisection bandwidth per node. On the initial Blue Waters configuration with 23 by 24 by 24 geminis, dedicated runs made in a node allocation with 6x24x24 geminis took 1.64X more time per step than a run made in a node allocation with 23x6x24 geminis [10].

A second application exhibiting large job to job performance variability is MILC [6], a Lattice Quantum Chromodynamics community code whose 4-dimensional lattice requires communication involving 4D halo exchanges as well as frequent reduction (All-reduce) operations. Although the halo exchanges in principle entail only nearest-neighbor communication in its 4D Cartesian grid virtual topology, in typical use the job's tasks are not carefully mapped onto the 3D torus to ensure

communicating tasks are placed onto nearby nodes. The current practice is to use Cray's grid_order utility to produce custom rank orders that minimize off-node traffic without attempting to place neighboring tasks on nearby nodes in the torus. Thus, for typical batch jobs that do not use special task placement, the effective communication pattern is between random pairs of nodes, resulting in much higher contention for inter-nodal bandwidth, even if no other jobs happen to be running on the system. We therefore anticipate better MILC performance in node allocation shapes having higher bisection bandwidth per node, for the same reason that such shapes improve All-to-All communication performance in PSDNS.

B. Mitigation Strategies

1) Node sets ("features"): In order to improve node allocation shapes for the science teams in the Blue Waters production environment, we used the ability of Adaptive Computing's Moab scheduler to assign groups of selected nodes to node sets, which we refer to here as "node features". For example, one series of node features configured on Blue Waters is comprised of sets of nodes with 24x6x24 geminis, each spanning a different set of 6 geminis in the torus y dimension. Users can target any one of these features using the following directive

```
#PBS -l nodeset=ONEOF:FEATURE:s1_6700n:s2_6700n:
s3_6700n:s4_6700n:s5_6700n:s6_6700n:s7_6700n:
s8_6700n:s9_6700n:s10_6700n:s11_6700n:s12_6700n:s13_
6700n
```

The scheduler will run the job in the first node feature in the list to have sufficiently many idle compute nodes. Note that, as currently configured on Blue Waters, it is quite possible for jobs that do not explicitly target a particular node feature to run on some nodes in that feature anyway, if they are available. Thus, to avoid long queue wait times, the "high" priority queue (which consumes service units at a higher rate) is often used for jobs that require nodes in a node feature.

These features contain a total of from 6724 to 6760 compute nodes when all nodes are up, since the number of service nodes varies somewhat with location in the torus. Users are advised to allow for several nodes to be down when submitting jobs, so they should request no more than 6720 nodes if they want to allow the job to run in any of these features.

We have observed significant performance improvements or at least substantial reductions in run time variation for most applications when run in node features. We obtained 30-50% performance improvements when using node features with PSDNS on up to 8192 nodes. Users of NWChem (a computational chemistry code based on the Global Arrays library with an irregular communication pattern, see [12]) always target node features in production runs. Among the more communication-intensive applications, an NWChem benchmark requiring 1000 nodes ran between 32% and 38% faster in node features with 12 by 4 by 12 geminis than it did when submitted without any restriction on node allocation. A 6700 node NAMD (molecular dynamics code, see [13]) benchmark ran 17% faster in a 24 by 6 by 24 gemini feature than it did in a run without restricting the allocation to any features, in which the allocation bounding box was 8 by 24 by 24 geminis.

When an application runs inside a node feature whose shape is such that it either spans the entire torus in a given dimension, or it spans up to half of that dimension, no application communication ever leaves the feature it is using, which prevents that communication from interfering with other jobs using links between geminis outside of the feature. However, for any dimension not spanned by the feature, it is possible for the scheduler to assign another job to groups of nodes on either end of the feature, allowing job-job interference due to other jobs using links inside the prism, although they are running on nodes outside of the prism.

2) Node Ordering Schemes: On Blue Waters, we observed that with the Cray ALPS standard node ordering scheme "-O2" described in [4], node allocations for large jobs on a dedicated system were full yz slabs whose extent in the x torus dimension was four geminis. Since xz slabs have higher bisection bandwidth per node than yz slabs of the same aspect ratio, We tried the existing "-OY" option, which favors xz planes of geminis, but found that this strategy has the disadvantage of assigning small allocations in a single y plane. Consequently, small jobs were given an allocation with relatively low bisection bandwith per node and a relatively large maximum hop count.

NCSA and C. Albing collaborated on modifying the "-OY" node ID ordering scheme to order the nodes in 4 by 2 by 8 gemini blocks, first filling the torus along z, then along x, and then along y. This change ensures that the xz slabs assigned to jobs are at least 2 geminis thick along y (one blade). A refinement added in this work reverses the direction (c.f., Peano curves) in which the blocks fill the torus after jumping to the next set of x (or y) values to help provide a more contiguous node list in which neighboring nodes are closer together on the torus. Further changes help the scheduler deal with the fact that the XK region on Blue Waters does not span the full torus in the x direction.

We used a synthetic workload that included a suite of seven real applications to evaluate the impact of the node ordering scheme on run times. The applications in this suite are: MILC, NWChem, PSDNS, Changa (collisionless N-body solver for astrophysics, [14]), NAMD, WRF (Weather Research and Forecasting, [15]), CESM (climate modeling,

[16]), and DNS_distuf (Computational Fluid Dynamics, spectral method, [17]).

Three different node ordering schemes where compared using this application suite. Allocations for the default or baseline scheme ("-O2") are depicted in Fig. 2. The geminis in each allocation are represented by a set of spheres of a unique color. It is evident that this scheme favors yz slabs that are 4 geminis thick along the x direction of the torus (y is up, z is toward the viewer). Fig. 3 shows allocations for "Nov.11_2Y", a scheme that favors xz slabs that are 2 geminis thick along y, while Fig. 4 shows allocations for "Nov.11_4Y", a scheme that favors xz slabs that are 4 geminis thick along y.

Table I gives the speedups for each application for which we obtained timing information when using the "2Y" and "4Y" node ordering schemes Two separate test runs were performed on two different dates, corresponding to the "Nov.11" and "Nov.4" prefixes in the table. Although some applications running in certain allocations performed better in the allocation provided by the older "-O2" scheme, on average the speedups were between 1.14 and 1.25. If we had been able to collect data for Changa, WRF, and CESM for all tests and they had exhibited little or no benefit from either new ordering scheme, the averages would be reduced to a range from 1.12 to 1.19, which is still a very substantial improvement over the baseline "-O2" scheme. The tests indicate that the 4Y scheme gives slightly better overall performance than the 2Y scheme. However, 2Y was ultimately selected to be the default because the applications showing the strongest 4Y benefit (skewing the averages to 4Y favor) were typically being run inside a pre-defined node feature anyway, and would therefore not suffer from 2Y as the default.

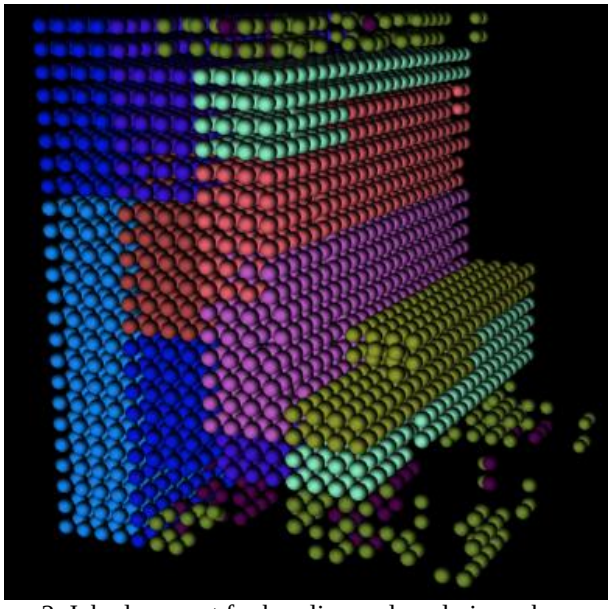

Figure 2. Job placement for baseline node ordering scheme.

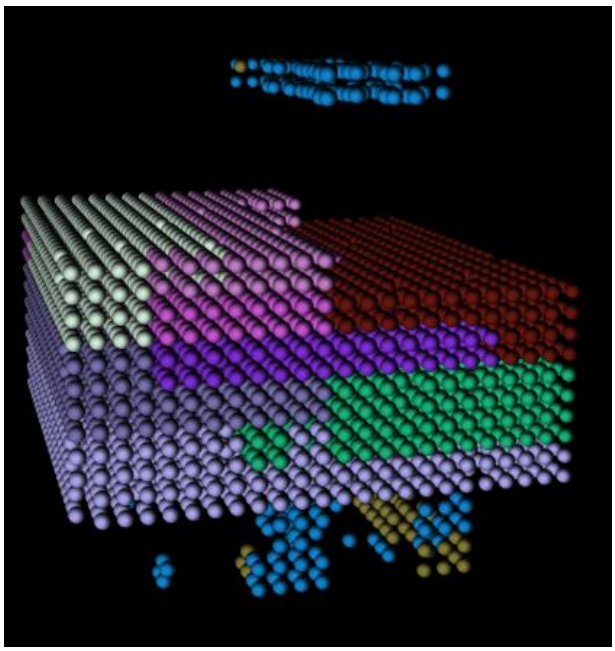

Figure 3. Job placement for Nov. 11 2Y node ordering.

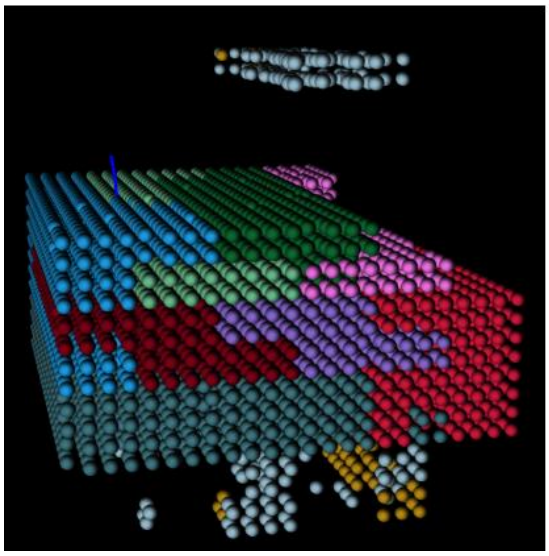

Figure 4. Job placement for Nov. 11 4Y node ordering.

C. I/O and Job-Job Interference

As the Lustre file system is currently designed, I/O operations can use any of the IO service nodes scattered throughout the system, and therefore IO traffic from other jobs would remain a source of job-job interference even if all jobs were running in favorable prism-shaped allocations. This type of interference is not addressed in this work, but may potentially be greatly reduced by enabling applications to use only the service nodes within their allocation prism when writing new files [18]. This strategy would not help when reading a restart dump from a prior run, unless the job happened to be assinged to the same node feature as the prior run. However, typical batch jobs write much more data then they read, mostly in the form of solution snapshots and checkpoints. However, this IO strategy may also significantly reduce IO throughput for a given job, since its IO operations could use only a fraction of the full system's IO nodes, and therefore only a fraction of the Lustre file system's OST resources.

TABLE I. APPLICATION SPEEDUPS FOR XZ ORDERING SCHEMES

App	Nodes	Nov.11 2Y	Nov.11 4Y	Nov.4 2Y	Nov.4 4Y
MILC	1372	1.00	1.02	1.15	0.91
MILC	2744	1.52	1.47	1.43	1.31
NWChem	3000	1.34	1.22	1.32	1.39
PSDNS	1024	1.09	1.15	1.22	1.74
Changa	1024	--	--	1.00	0.95
NAMD	1368	1.62	1.77	0.91	0.91
WRF	1386	--	--	1.01	1.01
CESM	600	1.01	1.00	--	--
DNS_distuf	512	1.13	1.13	1.05	0.98
AVERAGE		1.24	1.25	1.14	1.19

III. ADAPTIVE/CRAY/NCSA COLLABORATION

In 2013, a collaboration between Adaptive Computing, NCSA (Blue Waters project), and Cray was begun in order to address run time consistency issues by helping to improve application communication performance and reduce job-job interference through scheduling policies that take into account the characteristics of both the interconnect and the applications. These capabilities are expected to be included in a production release of the scheduler by the third quarter of 2014.

A. Goals and Design Considerations

The overarching goal of this collaboration is to add topology awareness to the Moab job scheduler for Cray systems with the gemini interconnect. However, the foundational changes implemented in Moab for this purpose will facilitate topology-aware scheduling capabilities for systems with other types of interconnects in the future.

The main objectives to be achieved by the new scheduler are to:

- Improve application performance through better-localized job placement.
- Improve application run-time consistency by eliminating job-job interference due to communication.
- Improve system throughput and maintain reasonably high utilization.
- Provide relevant configuration tools for users and administrators.
- Provide monitoring and diagnostic information to help users and administrators understand how jobs are being scheduled on the system.

In order to meet the first two objectives, we designed a series of tests (described below) to quantify the effect of node allocation shape and job-job interference for a set of representative applications running on Blue Waters.

Regarding the third objective, defining overall system throughput in a production environment is a non-trivial task. One might choose to measure throughput in terms of the number of floating point operations (FLOP) devoted to science and engineering applications running on the system over a fixed amount of time. However, we must account for the fact that different applications can achieve very different FLOP rates even on a dedicated system. Therefore, as described later, we use a synthetic workload with job sizes and run times derived from actual Blue Waters production workloads.

Given a representative workload, we can define the total system efficiency as the product of the scheduling efficiency and the average application efficiency, where the latter term is calculated as the production usage distribution weighted by application efficiency. The application efficiency is defined as the ratio between the measured run time and the run time obtained for the same benchmark problem on a dedicated system with the best possible mapping of tasks to nodes. For example, suppose 20% of the production workload is application A, 50% is application B, and 30% is application C. Suppose further that we examine application run times during a scheduler test, and find that application A is 93% efficient, application B is 87% efficient, and application C is 74% efficient. Then the average application efficiency "E_AVE" is

$$E\_AVE = 0.20 * 0.93 + 0.50 * 0.87 + 0.30 * 0.74 = 0.84$$

If the measured scheduling efficiency is 88%, then the total system efficiency is 74.2%. This metric can be compared for different versions of the scheduler running the same workload in order to determine whether increased application performance due to well-shaped allocations compensates for decreased utilization.

In the initial implementation of the new scheduler, a strict policy prohibiting job-job interference due to application communication is enforced. In order to do so, Moab chooses

only node allocations which guarantee that intra-job communication would not be routed over links used by any other job. Preliminary results gathered from the system-wide throughput test run on Mar 21, 2014 indicated that this strict enforcement of non-interference resulted in highly consistent job run times, with measured variations of only 1 to 4% for jobs run with the same prism dimensions. This strict policy is expected to have a relatively high impact on utilization, however.

Subsequent implementations may enable user-specified indications of communication sensitivity (the degree to which the application slows down when communication from other jobs shares the same links) and communication intensiveness (the degree to which the application saturates the links it uses). This information would allow the scheduler to more aggressively pack workloads that include applications that have low sensitivity and/or are less communication intensive via allocations that result in some link sharing. Such an approach would increase scheduling efficiency at the cost of possibly reduced application performance. Studies are planned to allow better workload characterization to determine the viability of this approach.

Most jobs do not fit perfectly into a cuboid or rectangular prism which is required to guarantee non-overlapping communication. Service and down nodes within a given prism tend to complicate this further, making 'perfectly matching' fits difficult to locate. Consequently, in most cases, some nodes within the bounding box of the prism will not actually be allocated to the job. Moab will attempt to utilize these 'internal' idle nodes by launching small jobs on them which are guaranteed to not interfere with the main job running in the same prism. Because of this optimization and the potential that all idle nodes could be subsequently allocated, Moab may charge the main job only for the nodes in the prism which are allocated and utilized, not for all nodes within the prism. Constraints on the small jobs utilizing the internal idle nodes could include limits on the run time, communication intensiveness, and/or requiring them to be pre-emptible, so that the system can signal them write a final application-level checkpoint and terminate shortly after the main job completes. In return for obeying these constraints, such small jobs could be charged at compellingly low rates.

B. Workload Test and Results

The purpose of this test is to create an environment which reproduces to the extent possible the target production environment, while allowing reasonable measurement of both scheduling efficiency and application performance. Consequently, the following steps were taken:

- use test resources that match the production resources in terms of
  - scale
  - architecture
  - mix of GPU and non-GPU nodes
  - topology
  - create test workload that matches the production workload in terms of
  - backlog size/depth
  - job submission timing distribution
  - job size distribution
  - job duration distribution (scaled down in time to fit test time window)
  - job walltime accuracy distribution
  - mix of applications including representative communication patterns
  - mix of node types required
- use scheduler configuration that matches the production scheduler, including policies, priorities, and limitations
- create starting state via reservations that match the steady state fragmentation of the production environment

To get the starting state for both the topology-aware scheduler and the older one, we measure a typical steady state in a long-running simulation of a system with a realistic workload and a topology-aware scheduler. This state is less fragmented than the one which the steady-state simulation would provide in the non-topology-aware environment.

TABLE II. SYNTHETIC WORKLOAD CODES AND PARAMETERS

Application	Node count/type	Time limit (m)
Changa	1024 XE	60
Chroma	768 XK	50
DNS-DISTUF	512 XE	10
MILC	324, 576, 1372, 4116 XE	30
NAMD	1, 2, 4, 8, 16, 32, 64, 100, 128, 256, 456, 640, 1368 XE and XK; 2000, 3272 XE	60
NWChem	400 XK; 1000, 7000 XE	30-45
PSDNS	3072 XE	30
QMCPACK	700 XK; 4800 XE	30
SpecFEM3D_Globe	5419 XE	60
WRF	456, 1386, 3298 XE	30

With such a test environment created, the following performance metrics were collected:

- overall steady state scheduling efficiency
- overall application efficiency

- per application runtime consistency and performance for each allocation shape and orientation

Table II lists the applications, node counts, and requested times for the synthetic workload used to test the scheduler on Blue Waters. SpecFEM3D_Globe [19] is a seismic wave propagation application whose unstructured computational grid discretizes the entire planet. Chroma [20] is a lattice QCD application based on the QUDA [21] library to utilize GPUs. Finally, QMCPACK [22, 23] is an electronic structure application based on Quantum Monte Carlo techniques.

In the workload, which is tailored to represent the actual set of job sizes run on Blue Waters in recent months (although the actual mix of applications is somewhat different from the synthetic workload), multiple instances of each job are submitted at different times. There are a much larger number of jobs with small node counts than large node counts, but the vast majority of the total service units delivered are consumed by jobs using 512 or more nodes. Many of the requested run times are the same (30 minutes), which reflects the maximum time limit imposed by the site. We reduced the lengths of the jobs in the workload compared to the real jobs so that we could complete a scheduler test in 3 hours which represents about 6 days of typical production.

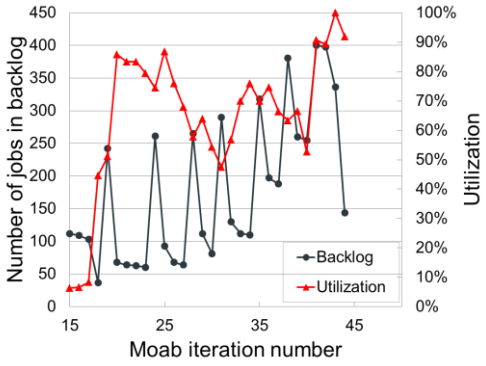

Figure 5. Utilization and backlog for Topology-aware scheduler test

Fig. 5 shows the backlog and utilization for the final 30 Moab scheduling iterations in a test of the topology-aware scheduler that was truncated after 2 hours due to a Gemini failure, which was corrected later via a warm swap operation. Each iteration took 3-4 minutes of wall clock time, which is not significantly greater than the time for a typical iteration of the old scheduler. In the test, groups of jobs are submitted together in multiple waves, corresponding to the spikes in backlog. Soon after the rise in backlog, the scheduler allocates nodes for many of the jobs, the backlog drops, and utilization increases somewhat. However, utilization tends to improve even more whenever the scheduler is able to start one of the largest jobs in the workload. Although the desired steady-state statistics were not obtained after only 2 hours, averaged over the last 10 iterations utilization was a respectable 71%.

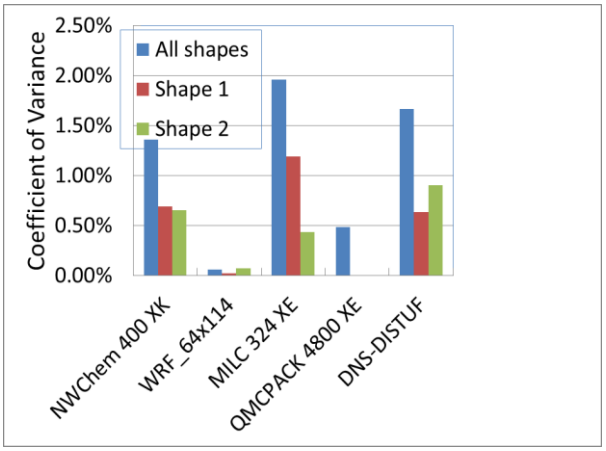

Figure 6. CoV for application run times during topology-aware scheduler test.

We found that the scheduler ran multiple instances of a given job (i.e., application and node count) in allocations with only 1-2 different shapes, although the location of an allocation of a given shape often varied from run to run. Fig. 6 shows the coefficient of variance for the run times of several applications for both shapes, and for each shape individually (QMCPACK ran in only one shape). Even though there were two different shapes with similar node counts for most of these applications, the largest CoV value is < 2%. As expected, for a given job, the CoV of run times for a given shape is substantially less than the CoV for two different shapes.

Comparing typical run times for PSDNS on 3072 nodes in production under the old scheduler to typical run times in our test of the topology-aware scheduler, we observed a 2.4X improvement with the new scheduler due to the favorable prism shape and elimination of job-job interference. For MILC, using grid_order even in the test of the new scheduler, the speedup on 324 nodes was 1.5X, while on 4116 nodes the speedup was 1.8X. The MILC run time CoV decreased from 18% with the old scheduler to 5% with the new scheduler. As we describe in the next section, a large (~2X or more) additional total run time improvement can be obtained for MILC when the Topaware tool is used to provide a near-optimal task layout.

Based on our preliminary results for the larger applications in the workload, we estimate an average performance improvement of roughly 40%. Anticipating a reduction in utilization of around 20% compared to the old scheduler, we expect an overall improvement in system throughput of approximately 20% for the topology-aware scheduler, which represents a very significant benefit for the Blue Waters science teams.

IV. TOPAWARE NODE SELECTION AND TASK PLACEMENT

To facilitate more effective use of topology-aware task mapping tools developed by Cray, new capabilities were added to Moab which allow the user to request a minimum required number of geminis (each with two available compute nodes) along each z-pencil through a prism. This enables near-optimal assignment of tasks to nodes for applications with primarily nearest-neighbor communication patterns. With this new capability, Moab takes into account any service or down nodes in each potential prism under consideration and rejects or adjusts the size of the prism as needed to provide the specified number of nodes in each z-pencil.

A. Topaware Algorithm

The Topaware node selection and task placement tool [7] provides near-optimal mappings of tasks to nodes in systems having gemini networks for applications with 2D, 3D, or 4D Cartesian grid process topologies. Its node selection strategy can be illustrated via an example for a 3D virtual topology with D1 by D2 by D3 partitions in each virtual dimension. In the absence of unavailable nodes, Topaware would map the tasks onto a prism of LX by LY by LZ geminis, and the pair of nodes attached to each gemini would have NX by NY by Nz partitions, such that

D1 = LX * NX
D2 = LY * NY
D3 = LZ * NZ

(Note that the 3 virtual dimensions can be aligned with any permutation of the 3 torus dimensions, and Topaware determines which permutation to use.)

The values of NX, NY, NZ are constrained by the requirement that their product be no larger than the desired number of tasks per node pair (often equal to the number of cores). The NX, NY, NZ and LX, LY, LZ values are further constrained by the dimensions of the region of the torus in which the search for usable nodes is conducted.

As an example, a virtual topology with 32 by 32 by 32 partitions can be mapped to a logical grid of 8 by 8 by 8 geminis, provided 4 by 4 by 4 partitions are placed on each pair of nodes attached to each gemini. If we chose instead to run the job with 8 tasks per node, we could use, for example, a prism with 16 by 16 by 8 geminis and 2 by 2 by 4 partitions per node pair. However, we could not use 8 tasks per node with 32 by 8 by 8 geminis and 1 by 4 by 4 partitions per node pair, since the Blue Waters torus has only 24 geminis along each dimension.

In the presence of unavailable nodes in the system, Topaware selects (from within a specified region of the system) a regular prism of geminis that has LY xz planes, each xz plane having LX pencils along z that all have at least LZ geminis with available compute node pairs. This "logical grid" of geminis is constructed by scanning the z-pencils beginning at the "leftmost" end of the search region and continuing until LZ available compute node pairs are identified, skipping over any geminis with one or more unavailable nodes. Fig. 7 illustrates the selection process for a single xz search plane of the torus with 8 by 9 geminis including service nodes (green squares without numbers). Here, the desired logical grid is 8 by 8 by 8 geminis, and we are able to obtain an xz plane of this logical grid in this region because all 8 z-pencils through the search plane have at least 8 available compute node pairs (numbered white squares).

The actual set of geminis to be used by the application (the "selected geminis") often has an irregular surface normal to z at both ends, but especially on the rightmost side, since most z-pencils can be expected to have one or more unavailable nodes somewhere along their length. The skipped node pairs add an extra hop or two along the torus z direction for nearest-neighbor communication paths in their vicinity, which increases the load on the local z links somewhat. However, the increased contention due to the skipped node pairs is usually much less than the contention arising from placing groups of tasks that should be neighbors onto the torus in a non-conforming pattern, which is nearly always the case when using the grid_order tool.

Topaware can be allowed to construct logical grids whose

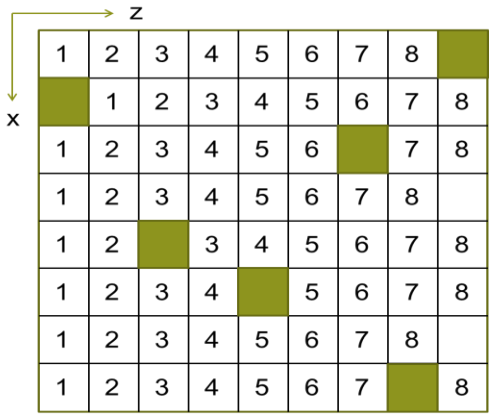

Figure 7. Topaware node selection algorithm

selected geminis have a non-contiguous set of xz planes or some z pencils that are left completely idle in order to use as many nodes as possible in a single benchmark run. However, in a production environment, if the search region is such that the selected geminis have any idle internal xz planes or z-pencils, it would be better to reject that layout or try a different search region. We therefore expect that in production, the selected geminis will always have LY xz planes with LX z-pencils.

If n is the largest number of unavailable node pairs along any z-pencil, then the selected nodes fit within a bounding box of LX by LY by PZ geminis (including those with unavailable node pairs), where PZ = LZ + n. Since the number of service nodes and down nodes is (normally) a small fraction of the number of available compute nodes, the value of n is typically between 1 and 3, depending on the size of the logical grid and the region of the torus being searched. The IO service node pairs are scattered more or less at random throughout the system, but other unavailable node pairs may be clustered together.

B. Results for 3D Halo Exchanges

In order to compare nearest-neighbor communication times for 2D, 3D, or 4D Cartesian grid virtual topologies, we developed a synthetic application that performs halo exchanges, i.e., point-to-point message passing between adjacent partitions along each virtual dimension. This code uses the Cray rca library to obtain the node ID and torus coordinates of each task, so that it can determine the actual path on the interconnect that each message takes. We submit a batch job that targets a node feature whose shape allows Topaware to obtain a near optimal layout. In the same batch job, we also run the halo-exchange code using the custom rank order generated by grid_order. The nodes used by the grid_order run are the first M nodes in the allocation, whose node order is determined by ALPS using the new scheme described in section 2. The Topaware run uses the M selected nodes that provide the logical grid of geminis required for the near-optimal mapping. A third run in the same batch job uses the default (SMP) rank order on the first M nodes in the ALPS list, in order to quantify the benefit of using grid_order.

For the 3D virtual topology in the example above with 32 by 32 by 32 partitions and 32 tasks per node running in a node feature with 12 by 8 by 12 geminis, Table III shows the averaged slowest (over all tasks) timings for halo exchanges. Here, non-blocking MPI send and receive operations are performed, and all messages are initiated at the same time, in order to avoid requiring the messages to arrive in any particular order and to potentially enable overlap of various operations such as copying data to/from cache/memory with the transmission of message packets across links between node pairs. The message size was 32 kB.

TABLE III. 3D HALO-EXCHANGE TIMINGS

Placement	Iter time (ms)	Max hops
Default	11.315	9
Grid_order	7.722	16
Topaware #1	2.771	2

The default rank order places 32 consecutive tasks on the first node in the allocation, the next 32 tasks on the second node, etc. Since there are 32 tasks in each virtual dimension, this placement eliminates off-node communication for an entire dimension. For the grid_oder run, the per task layout was 4 by 2 by 4 partitions, which helps to decrease the communication time by reducing the amount of off-node communication compared to default placement. The halo-exchange code counted a maximum of 16 hops for messages in the grid_order run, but a maximum of 9 hops for default placement. The default task placement actually resulted in a smaller maximum for the hop count, and yet the communication time was nearly 1.5X shorter for grid_order placement. For Topaware placement, the largest hop count was only 2 and communication times are reduced by a factor of nearly 2.8 compared to grid_order placement. Ideally, the maximum hops count would be 1, but the presence of unavailable nodes precludes achieving such a perfect layout. Evidently, there are no z-pencils through the selected nodes with more than one unavailable node in this particular experiment.

C. 4D Virtual Topologies

For 4D virtual topologies, we constrain the partitions of the 4th dimension (called "T" for time) to fit on each node pair, and treat the remaining 3 dimensions in the manner described above for 3D virtual topologies. For example, a virtual topology with 8 partitions in each dimension can be mapped onto the system in several ways with 16 tasks per node:

1) LX = 8, LY = 4, LZ = 4, NX = 1, NY = 2, NZ = 2, NT = 8
2) LX = 4, LY = 4, LZ = 8, NX = 2, NY = 2, NZ = 1, NT = 8
3) LX = 8, LY = 2, LZ = 8, NX = 1, NY = 4, NZ = 1, NT = 8

All of these layouts have NT = 8 to place all 8 T partitions on each node pair (4 T partitions per node).

A second 4D example has 11 by 12 by 11 by 12 partitions (17424 tasks), which will run on 545 nodes with 32 tasks per node using default placement or grid_order (although some nodes will have fewer than 32 tasks). In contrast, the perfectly balanced layout Topaware obtains uses only 24 tasks per node, but 726 nodes. Topaware's layout parameters are:

LX = 11, LY = 3, LZ = 11, NX = 1, NY = 4, NZ = 1, NT = 12

For message sizes ~24 kB, Table IV gives the timings for halo exchanges run in a node feature with 12 by 4 by 12 geminis. Topaware reduced the communication time by a factor of more than 4.3 compared to grid_order, although it used 4/3 more nodes in doing so.

TABLE IV. 4D HALO-EXCHANGE TIMINGS

Placement	Time per iter (ms)	Max hops
Default	9.65	21
Grid_order	7.46	19
Topaware	1.72	2

D. Real 4D Application: MILC

To quantify the improvement in overall run times when Topaware is used (rather than grid_order) for a real application with a 4D virtual topology, we used MILC, a Lattice Quantum Chromodynamics community code [6]. MILC spends a significant portion of its total run time on communication operations including halo exchanges and reductions (All-reduce) when grid_order is used.

We considered a lattice with 84 by 84 by 84 by 144 points, and divided this grid into 21 by 2 by 21 by 24 partitions (21168 tasks), so that it could run on 1764 nodes with 12 tasks per node in a node feature with 24 by 2 by 24 geminis. For grid_order, we chose a per-node layout with 1 by 1 by 1 by 12 partitions so that it matches the Topaware per-node layout, and therefore any difference in run times must be due to Topaware's careful placement of neighboring tasks onto neighboring nodes in the torus, rather than differences in the node count or even the per-node task layout. Message sizes range from 2.3 kB to 48 kB. The run times in Table V show that using Topaware instead of grid_order improves overall performance by a factor of 2.2X for this test case.

TABLE V. MILC TIMINGS

Placement	Run Time (10 iterations)
Grid_order	254.0
Topaware	116.4

E. 2D Virtual Topologies

For 2D virtual topologies, the virtual domain must be carefully folded into multiple layers (called "supertiles"), each of which fits into one plane of the selected set of geminis. The direction in which tasks for each layer are placed onto the torus is reversed at each fold to ensure that tasks on either side of the fold in virtual space are on neighboring nodes of the torus. Communication between tasks on different layers occurs only at the folds. This communication travels on the torus in the direction that the layers are stacked, using only links on the surface of the selected geminis. Communication between tasks in a given supertile away from the edges stays within the plane of the supertile.

Consider a 2D virtual topology with D1 = 110 by D2 = 120 partitions to be placed in a node feature with 12 by 4 by 12 geminis (including service nodes) and no more than 16 tasks per node. The virtual domain will be folded into supertiles along x and/or z, and the supertiles will be stacked along the torus y dimension in this case, because it is the shortest dimension of the node feature. We require

D1 = LX * NX * (1 + number of folds in x), and
D2 = LZ * NZ * (1 + number of folds in z).

Topaware finds several viable layouts, all with 4 layers:

1) LX = 11, LY = 4, LZ = 10, NX = 5, Nz = 6,
1 fold in z, 1 fold in x
2) LX = 10, LY = 4, LZ = 11, NX = 6, Nz = 5,
1 fold in z, 1 fold in x
3) LX = 11, LY = 4, LZ = 10, NX = 10, Nz = 3,
3 folds along z
4) LX = 10, LY = 4, LZ = 11, NX = 3, Nz = 10,
3 folds along x

For messages of size 4000 B, Table VI shows iteration times in ms. Communication for the best-performing Topaware layout is over 1.3X faster than it is for the grid_order run. We also note that the maximum hop count is not a strong predictor of performance for the Topaware runs.

TABLE VI. TIMINGS FOR 2D HALO EXCHANGES

Placement	w/Stagger	w/o Stagger	Max hops
Default	0.2743	0.2745	15
Grid_order	0.2149	0.2153	18
Topaware #1	0.1638	0.1624	3
Topaware #2	0.1597	0.1597	5
Topaware #3	0.1671	0.1676	2
Topaware #4	0.1863	0.1868	3

The layouts (#1 and #2) with folds along both virtual dimensions tend to give better performance, probably because of the smaller aspect ratio of the per node-pair partition layout, which reduces the amount of off-node communication. For these layouts, some links at the folds are used by multiple pairs of communicating layers. There seems to be sufficient bandwidth for this not to be a bottleneck, despite the fact that the links between layers are the slower y links.

Table VI presents the results of two different batch jobs, one in which for Topaware layouts #1 and #2 we attempt to

reduce the loads on the y links between the four layers by staggering pairs of communicating supertiles by one gemini in x, as proposed in [7] (Fig. 8), and one batch job in which this staggering was disabled (Fig. 9). Since the two sets of run times are practically the same, it appears to be better to avoid the use of staggering, since it enlarges the bounding box of the selected geminis with little or no improvement in communication time. Interestingly, the maximum hop counts are the same for the layouts with and without staggering, indicating that adding an additional hop along x for the some pairs of xz planes does not necessarily affect the longest paths, which are presumably along the z-pencils with the most unavailable node pairs. The two sets of timings for the rest of the layouts show that the variation from one run to another (in this case in the same node features) is within < 1%.

We observe much smaller improvements when using Topaware for 2D virtual topologies compared to 4D, probably because the 4D communication pattern is more intensive (messages are sent to twice as many neighbors as they are for 2D), and therefore the links are driven closer to capacity.

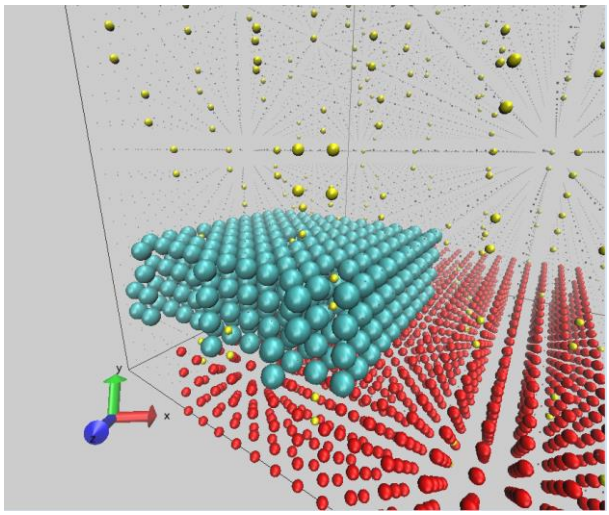

Figure 8. 2D Layout with no staggering.

F. Unbalanced Layouts

The Topaware layouts presented above are perfectly balanced, since each node has the same number of active ranks. However, virtual topologies often have a partition count along one or more dimensions that does not factor into the product of an integer number of geminis that fits onto the torus and an integer number of partitions per node pair. Topaware was recently enhanced to allow unbalanced layouts in order to handle such cases and to enable a much larger number of near-optimal layouts for a given virtual topology on a given system.

An unbalanced layout is a layout in which one or more dimensions does not satisfy D = L * N. Instead, we set L equal to the smallest integer value that satisfies L * N > D for a proposed value of N. Again consider the 3D example with 32 by 32 by 32 partitions (32768 tasks). Topaware finds the following eight layouts including the balanced one presented above (as #1):

1) LX = 8, LY = 8, LZ = 8, NX = 4, NY = 4, NZ = 4
2) LX = 11, LY = 6, LZ = 11, NX = 3, NY = 6, NZ = 3
3) LX = 11, LY = 8, LZ = 8, NX = 3, NY = 4, NZ = 4
4) LX = 8, LY = 8, LZ = 11, NX = 4, NY = 4, NZ = 3
5) LX = 11, LY = 7, LZ = 8, NX = 3, NY = 5, NZ = 4
6) LX = 8, LY = 7, LZ = 11, NX = 4, NY = 5, NZ = 3
7) LX = 11, LY = 8, LZ = 7, NX = 3, NY = 4, NZ = 5
8) LX = 7, LY = 8, LZ = 11, NX = 5, NY = 4, NZ = 3

Note that the unbalanced layouts allocate more tasks than are needed by the virtual topology. For example, layout #3 has 33 by 32 by 32 partitions instead of the required 32 by 32 by 32. Most node pairs will have 3 partitions in x, but the node pairs in the 11th yz plane of the selected geminis containing the logical grid (i.e., the node pairs on the boundary of the selected geminis) are assigned only 2 partitions in x. Thus, these boundary node pairs have 2/3 of the workload of the rest of the node pairs. Since the most heavily loaded node pairs govern the rate of progress of a parallel application, having a modest fraction of nodes with a lighter load has no effect on the overall run time.

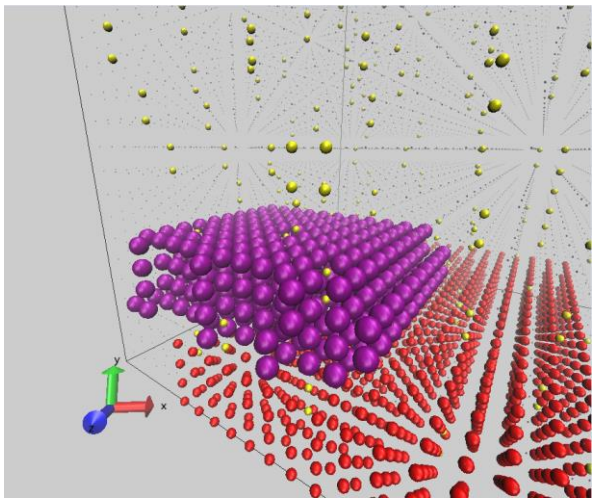

Figure 9. 2D Layout with staggering.

The balanced layout (#1 above) has 32 tasks per node, while the unbalanced layouts have 24, 27, or 30 tasks per node. Since there are fewer tasks per node than there are available cores for the unbalanced layouts, we can use Cray's "core specialization" feature to assign OS and other tasks to an idle

core on each node. This also enables communication overlap via the Asynchronous Progress Engine, so that the non-blocking send and receive operations can be overlapped with each other and with other overhead, such as copying data to/from message buffers. With fewer tasks per node, we also get more memory bandwidth per task, since the memory bandwidth per node is fixed and limits the rate of the copy operations. These benefits can be seen in Table VII, which presents the full set of timings for 3D halo exchanges using all 8 Topaware layouts.

TABLE VII. 3D HALO EXCHANGES (W/UNBALANCED LAYOUTS)

Placement	Iter time (ms)	Max hops
Default	11.315	9
Grid_order	7.722	16
Topaware #1	2.771	2
Topaware #2	1.287	2
Topaware #3	1.147	2
Topaware #4	1.214	2
Topaware #5	1.782	2
Topaware #6	1.737	2
Topaware #7	1.580	2
Topaware #8	1.690	2

The best performing layout (#3) is unbalanced. This run is over 2.4X faster than the balanced Topaware layout and nearly 10X faster than the grid_order layout at the cost of using 32/24 = 4/3 more nodes than the balanced layouts. It definitely seems worth exploring the use of layouts with fewer than 32 tasks per Cray XE node, and fewer than 16 tasks per Cray XK node, even without using Topaware, especially if the application uses non-blocking communication operations and initiates all of them at once.

As a second example with unbalanced layouts, consider a 2D virtual topology with 168 by 132 partitions that is to run in a node feature with 12 by 8 by 12 geminis using no more than 16 tasks per node. Topaware proposes both balanced and unbalanced layouts:

1) LX = 11, LY = 8, LZ = 11, NX = 6, Nz = 4,
1 fold in x, 3 folds in z
2) LX = 11, LY = 8, LZ = 11, NX = 3, Nz = 8,
3 folds in x, 1 fold in z
3) LX = 12, LY = 6, LZ = 11, NX = 7, Nz = 4,
1 fold in x, 2 folds in z
4) LX = 11, LY = 6, LZ = 12, NX = 4, Nz = 7,
2 folds in x, 1 fold in z
5) LX = 11, LY = 6, LZ = 11, NX = 4, Nz = 8,
2 folds in x, 1 fold in z
6) LX = 12, LY = 6, LZ = 11, NX = 14, Nz = 2, 5 folds in z
7) LX = 11, LY = 6, LZ = 12, NX = 2, Nz = 14, 5 folds in x
8) LX = 11, LY = 8, LZ = 11, NX = 12, Nz = 2, 7 folds in z
9) LX = 11, LY = 6, LZ = 11, NX = 2, Nz = 16, 7 folds in x

Topaware was unable to obtain sets of selected geminis that span 12 geminis in z, since the node feature in which the job ran spans only 12 geminis in z and there were some unavailable nodes in the allocation. This eliminated two of the balanced layouts, #4 and #7. The staggering technique was not used; it would have eliminated the two remaining balanced layouts, #3 and #6. We obtained the timings for 2D halo exchanges shown in Table VIII for 8 kB messages.

TABLE VIII. 2D HALO EXCHANGES (W/UNBALANCED LAYOUTS)

Placement	Iter time (ms)	Max hops
Default	0.5743	7
Grid_order	0.3343	10
Topaware #1	0.2364	4
Topaware #2	0.2312	4
Topaware #3	0.2523	4
Topaware #5	0.2583	4
Topaware #6	0.2732	2
Topaware #8	0.2365	2
Topaware #9	0.2823	2

All runs in this series used core specialization. The default and grid_order runs used 16 tasks per node, while the Topaware runs used from 12 to 16. The Topaware layouts that use more nodes (12 tasks per node) tend to have roughly 1.2X better run times than the Topaware layouts that use 16 tasks per node, but that does not quite make up for their using 1.33X more nodes. The slowest Topaware layout (#9) performs nearly 1.2X better than the grid_order run, and it uses the same number of node as the grid_order run. The most efficient layout appears to be #5, an unbalanced one, since it uses 16 tasks per node and is nearly 1.3X faster than the grid_order run.

Note that we launch these jobs using a Cray aprun command that creates the same number of tasks on each node, since it would be tedious to determine which boundary nodes get what number of tasks and launch a job in MPMD mode in order to create only the minimum number of tasks required for the virtual topology. In order to use an unbalanced layout conveniently, the halo exchange application was modified to leave tasks idle that are unused by the virtual topology. Topaware orders the tasks so that the first N MPI ranks are the N tasks required by the virtual topology. The application must be modified to split the MPI_COMM_WORLD communicator into a new one with only the first N ranks, and to use the new communicator in place of MPI_COMM_WORLD throughout the rest of the code. This should be a straightforward procedure in real applications, although this would require a thorough examination of the code.

G. Integration of Topaware and Scheduler

Topaware was originally designed as a tool for benchmarkers running jobs on a dedicated system, where it could select an optimal set of nodes for a benchmark requiring, say, ~25% of all compute nodes without restrictions. The new scheduler developed in this work that provides prism-shaped node allocations makes it much more practical for ordinary users to benefit from Topaware in a production environment. Without this new scheduler capability, the only way to get prism-shaped node allocations is to target the existing node features, which limits the choice of virtual topologies for which Topaware can obtain near-optimal layouts. In addition, in many cases one must ask for significantly more nodes than are necessary to run the job, so that the allocation includes nearly all nodes in the feature. Otherwise, there may not be sufficiently many available node pairs in each z-pencil through the allocation, and therefore a near-optimal layout will not be obtainable.

The new scheduler developed in this work allows the user to request an allocation with specified numbers of geminis along each torus dimension. This is all that is needed for applications with irregular or All-to-All communication patterns. For applications with Cartesian grid topologies, in order to make the best use of Topaware, the new scheduler can provide prism-shaped allocations with at least the requested number of available compute node pairs along each z-pencil through the allocation, ensuring that Topaware can obtain the desired logical grid of geminis within in the allocation for the application's tasks.

Topaware was recently enhanced as part of this work to enable it to be used routinely in tandem with the new scheduler. The user typically wants to run a simulation of a particular size (e.g., a fixed number of cells in a global grid, and a fixed number of grid cells per processing element based on available memory or scaling considerations) and chooses the desired number of compute nodes to use on that basis. The user can invoke Topaware, describing only the desired number of partitions in each virtual dimension and a target value for the number of tasks per node, and Topaware will determine multiple viable layouts that will fit within the specified region of the torus (i.e., a feature, a bounding box, the current allocation in a batch job, or the full system without restrictions). Topaware completes its work after only a second or two, including checking for down nodes and generating node lists and rank order files for all valid layouts that can be obtained within the specified torus region. This list of viable layouts is output as a string that can be passed to the new scheduler, which can search for prisms of available nodes that match any one of these layouts. This added flexibility in scheduling such jobs helps to shorten queue wait times and improve system throughput.

.

V. CONCLUSIONS

We set out to address the problem of widely varying application run times on large Cray systems with 3D torus interconnects. Before changes were made to the scheduler to provide prism-shaped node allocations, we changed Cray's ALPS node ordering scheme to favor allocations whose shapes tend to be flattened along the direction of the slowest links. This improved the performance of a representative Blue Waters workload by 12-19% without decreasing utilization. However, job-job interference is not eliminated by this scheme, and task layouts remain less than optimal for applications with nearest-neighbor communication patterns.

Prism-shaped node allocations improve run times for many types of applications by optimizing communication bandwidth, decreasing hop counts, and significantly reducing job-job interference. The new scheduler developed in this work places jobs in prism-shaped node allocations. It chooses the best prism shapes for each job based on various metrics, taking into account asymmetrical link speeds, torus dimensions, how completely the requested node count fills the prism, fragmentation of unused resources, etc., in order to maintain reasonably high utilization. The new scheduler enables different sites to find the best balance between application efficiency and utilization to optimize overall throughput for their workload.

Putting all of these improvements together, in a preliminary throughput test with a realistic synthetic workload on Blue Waters, the new scheduler exhibited a total system throughput that is roughly 20% higher than that of the baseline scheduler. Moreover, both application performance and run time variations were significantly improved compared to the old scheduler.

Using the Topaware node selection and task placement tool with applications having nearest-neighbor communication patterns can lead to substantial reductions in overall run times. Results for MILC (4D virtual topology) demonstrate a 2.2X overall run time improvement when using Topaware instead of grid_order on the same set of nodes with the same per-node task layout. Support recently added to Topaware for unbalanced layouts enables near-optimal task placement for a much wider choice of virtual topologies on any given system with a torus interconnect. The new scheduler makes using Topaware in a production environment practical, because the scheduler locates the first available set of nodes that accommodates one of the near-optimal task layouts generated by Topaware for the application's problem-specific virtual topology.

An additional workload test using the old scheduler and then the new scheduler with Topaware enabled for the MILC jobs is planned. These results will improve our estimates of the improvements in system utilization, application performance, and overall system throughout for the new scheduler.

ACKNOWLEDGMENT

We wish to thank Cray's Carl Albing for his work in developing a modified version of the ALPS node ordering scheme that improves upon the existing "-OY" option.

REFERENCES

[1] Blue Waters System Description, https://bluewaters.ncsa.illinois.edu/user-guide.

[2] Abhinav Bhatele, I-Hsin Chung and Laxmikant V. Kale, "Automated Mapping of Structured Communication Graphs onto Mesh Interconnects", Computer Science Research and Tech Reports http://hdl.handle.net/2142/15407, April 2010.

[3] I. V. Lo, K. J. Windisch, Wanqian Liu, and B. Nitzberg, "Noncontiguous processor allocation algorithms formesh-connected multicomputers," IEEE Transactionson Parallel and Distributed Systems, vol. 8, no. 7, pp. 712-726, Jul. 1997.

[4] 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.

[5] 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.

[6] MIMD Lattice Computation (MILC) Collaboration code page, http://physics.indiana.edu/~sg/milc.html.

[7] R. Fiedler and S. Whalen, "Improving task placement for applications with 2D, 3D, and 4D virtual Cartesian topologies on 3D torus networks with service nodes", CUG 2013, May 6-9, 2013, Napa, CA, USA.

[8] Jose Antonio Pascual, Jose Miguel-Alonso, "Effects of Job Placement on Scheduling Performance", Actas de las XIX Jornadas de Paralelismo, pp. 393-398, 2008.

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

[10] 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, 2013, Napa, CA, USA.

[11] D. A. Donzis, P. K. Yeung, & D. Pekurovsky, "Turbulence simulations on O(10,000) processors". In TeraGrid 2008 Conference Proceedings, Las Vegas, NV, USA.

[12] M. Valiev, E.J. Bylaska, N. Govind, K. Kowalski, T.P. Straatsma, H.J.J. van Dam, D. Wang, J. Nieplocha, E. Apra, T.L. Windus, W.A. de Jong, "NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations" Comput. Phys. Commun. 181, 1477 (2010).

[13] C. Mei, Y. Sun, G. Zheng, E. J. Bohm, L. V. Kale, J. C. Phillips, and C. Harrison, "Enabling and scaling biomolecular simulations of 100 million atoms on petascale machines with a multicore-optimized message-driven runtime," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, (New York, NY, USA), pp. 61:1–61:11, ACM, 2011.

[14] Changa home page, http://www-hpcc.astro.washington.edu/tools/changa.htm.l

[15] 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.

[16] Community Earth System Model (CESM), http://www2.cesm.ucar.edu/

[17] Lucci, F., Ferrante, A. and Elghobashi, S. E. "Modulation of isotropic turbulence by particles of Taylor-lengthscale size", J. Fluid Mechanics, Vol. 650, pp. 5-55, 2010.

[18] K. Chadalavada and R. Sisneros, "Analysis of the Blue Waters File System Architecture for Application I/O Performance", CUG 2013, May 6-9, 2013, Napa, CA, USA.

[19] L. Carrington, D. Komatitsch, M. Laurenzano, M. M. Tikir, D. Michea, N. Le Goff, A. Snavely, and J. Tromp, ``High-frequency simulations of global seismic wave propagation using specfem3d_globe on 62k processors," in Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08, Piscataway, NJ, USA, IEEE Press, 2008.

[20] R. G. Edwards (LHPC Collaboration), B. Joó (UKQCD Collaboration), *"The Chroma Soft ware System for Lattice QCD"*, arXiv:hep-lat/0409003, Proceedings of the 22nd International Symposium for Lattice Field Theory (Lattice2004), Nucl. Phys B1 40 (Proc. Suppl) p832, 2005.

[21] M. A. Clark, R. Babich, K. Barros, R. C. Brower, C. Rebbi, *Solving Lattice QCD systems of equations using mixed precision solvers on GPUs* arXiv:0911.3191v1 [hep-lat]

[22] J. Kim, K. P. Esler, J. McMinis, M. A. Morales, B. K. Clark, L. Shulenburger, and D. M. Ceperley, "Hybrid algorithms in quantum monte carlo," Journal of Physics: Conference Series, vol. 402, no. 1, p. 012008, 2012.

[23] K. P. Esler, J. Kim, L. Shulenburger, and D. M. Ceperley, "Fully accelerating quantum monte carlo simulations of real materials on gpu clusters," Computing in Science and Engineering, vol. 14, p. 40, 2012.