Optimizing Job Placement on the Cray XT3

Deborah Weisser, Nick Nystrom, Chad Vizino,
Shawn T. Brown, *and* **John Urbanic**
Pittsburgh Supercomputing Center

ABSTRACT: *The Cray XT3's SeaStar 3D mesh interconnect, which at PSC is configured as a 3D torus, offers exceptional bandwidth and low latency. However, improperly scheduled jobs produce contention that undermines otherwise scalable applications. Empirical observations indicated that production jobs were being badly fragmented, seriously degrading their performance. Recently we have quantified the impact of non-contiguous partitions on a variety of communication-intensive applications, including production codes, which are representative of PSC's workload. We are investigating different combinations of job scheduling strategies which automatically map processors to jobs or accommodate user specifications for required or desired layouts. A flexible placement policy supports multiple protocols to accommodate PSC's varied workload and users' needs while mitigating the effects of fragmentation.*

KEYWORDS: Cray XT3, processor allocation, job placement, job scheduling, optimization

1. Introduction

The Cray XT3's SeaStar 3D mesh interconnect, which is configured on BigBen[1] at PSC as a 3D torus, offers exceptional bandwidth and low latency. However, improperly placed jobs produce contention that undermines otherwise scalable applications. If multiple communication-intensive jobs are running concurrently, performance will degrade when messages for one job are routed through processors owned by another job.

Empirical observations indicated that production jobs were being badly fragmented, substantially degrading their performance. The default processor-to-job assignment policy, which assigns jobs to cabinets in numerical order, is far from optimal because numerically sequential cabinets are not necessarily directly connected to each other, and even the connected subsets of processors assigned to a job are not typically compact. This protocol increases latency, and it also increases job fragmentation, seriously undermining performance.

Recently we have quantified the impact of non-contiguous job placements, described in Section 3, on a several communication-intensive applications which are representative of PSC's workload. We experimented with different job layout protocols: the default job placement protocol, and an optimized protocol which assigns jobs to connected cabinets of processors in row-major order. The results are compelling: communication-intensive jobs representative of PSC's workload clearly show the effects of fragmentation under the default protocol, and their efficiency increases substantially when they are run on compact, convex connected subsets of processors.

In addition to the optimized protocol used in the experiments described in Section 3, we are evaluating different job placement protocols, discussed in Section 4, to minimize contention, maintain compactness, and minimize fragmentation. We are investigating combinations of different job scheduling strategies to accommodate PSC's varied workload and users' priorities and to reduce the effects of fragmentation. We are also exploring the use of pairwise communication profiling information obtained from CrayPat[2] to automatically generate job placements optimized for particular applications.

The results and techniques described in this paper, obtained on BigBen and described in the context of the Cray XT3, can be generalized to other 3D mesh computers.

Related Work

A polynomial-time approximation scheme for assigning processors to jobs to minimize the average pairwise distance is described in Bender et al.[3] The algorithm guarantees a placement within $2-\frac{1}{2d}$ of optimal and was implemented on the Cray T3D. The Cray T3E job placement protocol assigned cubes of connected processors to jobs (or an approximation if a perfect cube was not available).[4]

2. Machine Organization

PSC's Cray XT3 has 2090 processors, of which 2068 are configured as compute processors (the remaining 22 are service I/O processors, serving as login processors, Lustre OSTs, and network interfaces). BigBen is physically arranged in 2 rows of 11 cabinets. Each cabinet contains 3 cages, and each cage contains 8 blades. Each compute blade contains 4 Opteron processors, yielding up to 32 compute processors per cage and up to 96 compute processors per cabinet[5].

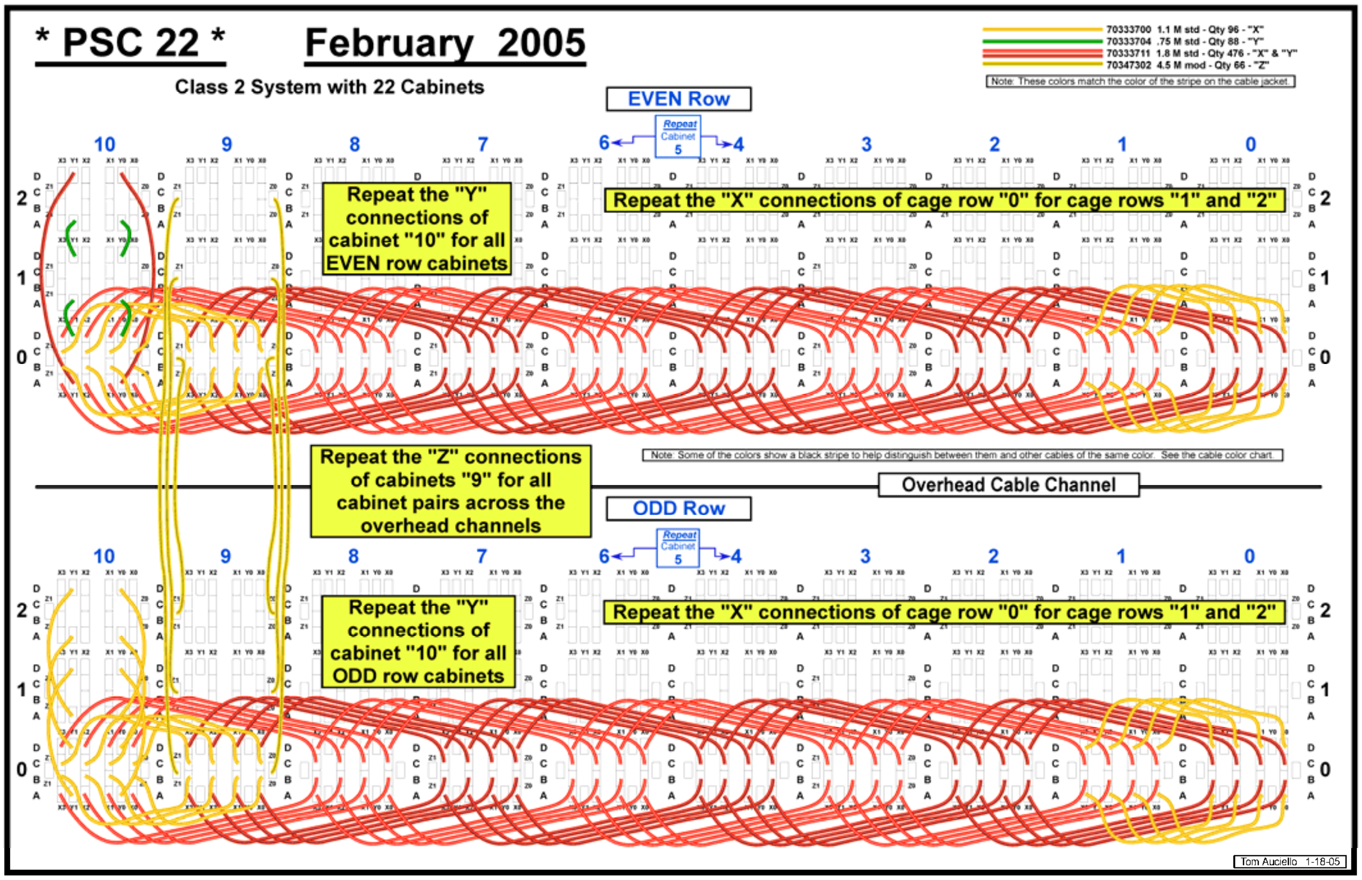
The cabinets are connected to form a 3D torus. The *x*-dimension cables connect cabinets within a row to each other; every cabinet is connected to 2 other cabinets. The *y*-dimension cables connect cages within a cabinet and are internal to a cabinet, and the *z*-dimension cables connect pairs of cabinets between rows. The wiring diagram for BigBen is shown in Figure 1. There are additional *y*- and *z*-dimension connections which are internal to a cage and do not appear in the wiring diagram. This topic is discussed in more detail in Section 4.2.

Note that the *x*-dimension cables connect each cabinet to 2 others, forming a folded torus to minimize the maximum cable length. Thus along each row the cabinets are physically located in the following order:

10 – 9 – 8 – 7 – 6 – 5 – 4 – 3 – 2 – 1 – 0

They are connected by *x*-dimension cables in the following order:

0 – 2 – 4 – 6 – 8 – 10 – 9 – 7 – 5 – 3 – 1 – 0 – …



* PSC 22 *
February 2005
Class 2 System with 22 Cabinets
70333700 1.1 M std - Qty 96 - "X"
70333704 .75 M std - Qty 88 - "Y"
70333711 1.8 M std - Qty 476 - "X" & "Y"
70347302 4.5 M mod - Qty 66 - "Z"
Note: These colors match the color of the stripe on the cable jacket.
EVEN Row
Repeat Cabinet 5
10 9 8 7 6 4 3 2 1 0
Repeat the "Y" connections of cabinet "10" for all EVEN row cabinets
Repeat the "X" connections of cage row "0" for cage rows "1" and "2"
Repeat the "Z" connections of cabinets "9" for all cabinet pairs across the overhead channels
Note: Some of the colors show a black stripe to help distinguish between them and other cables of the same color. See the cable color chart.
Overhead Cable Channel
ODD Row
Repeat the "Y" connections of cabinet "10" for all ODD row cabinets
Tom Auciello 1-18-05

Figure 1. The BigBen wiring diagram shows which cabinets (and cages) are connected.

Sequentially numbered cabinets are not necessarily directly connected to each other. Table 1 is an adjacency matrix of the number of hops along the *x*-axis between pairs of cabinets.

In the next section we will examine the relation between two job placement protocols, cabinet connectivity, and application performance.

Table 1. Number of hops messages must traverse between cabinets within each row. Cabinets are numbered sequentially along each row; however, to avoid long wires, logical connectivity leapfrogs cabinets as described in the text.

cabinet	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0											
1	1										
2	1	2									
3	2	1	3								
4	2	3	1	4							
5	3	2	4	1	5						
6	3	4	2	5	1	5					
7	4	3	5	2	5	1	4				
8	4	5	3	5	2	4	1	3			
9	5	4	5	3	4	2	3	1	2		
10	5	5	4	4	3	3	2	2	1	1	

3. Quantifying Job Placement Strategies

3.1 Placement Based on Default Placement Policy

The default scheduling policy assigns free processors to a job by processor number, issuing free processors numerically from low to high. The processors are numbered by cabinet, alternating rows; i.e. the lowest-numbered processors are in row 0 cabinet 0, then row 1 cabinet 0, then row 0 cabinet 1, etc. As indicated in Table 1, numerically sequential processors may be in cabinets which are separated from each other by up to 5 hops.

We use screen captures of PSC's 3D Monitor[6] to display a typical mix of jobs on BigBen in various ways. The processors are color-coded by job. Free, disabled, service, and down processors are designated by white, red, yellow, and gray blocks, respectively, and users' jobs are designated by a range of colors from brown to pink.

Figure 2 shows the distribution of jobs on the processors of BigBen, where the processors are shown in their physical locations within cabinets. The seven jobs in Figure 2 were assigned to processors using the default job placement algorithm, which places jobs on numerically sequential cabinets. From the figure, there is little

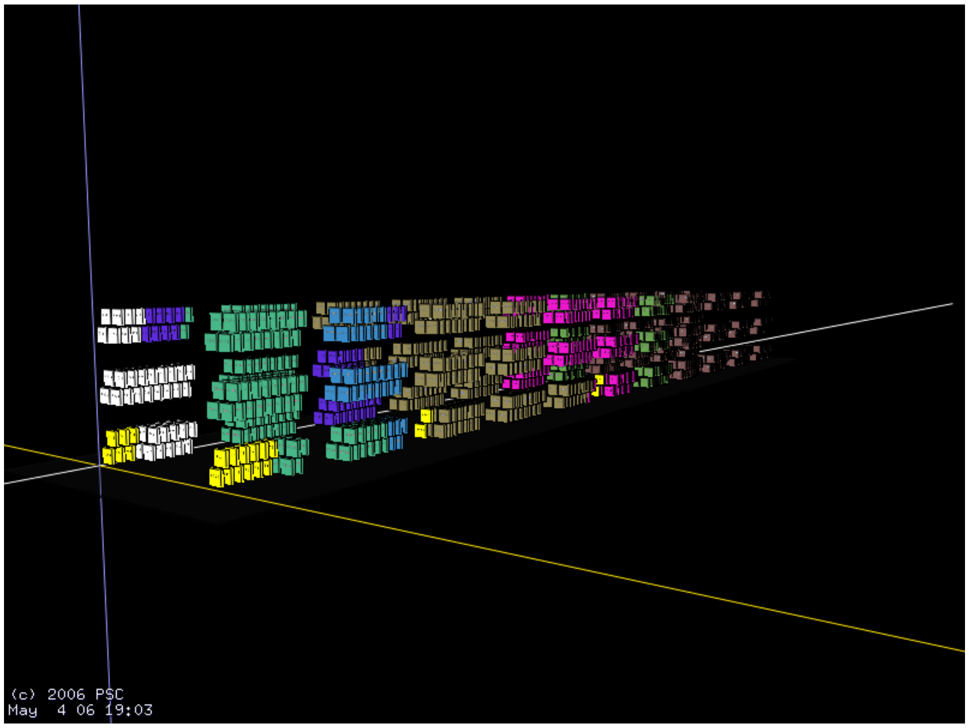


Figure 2. The distribution of jobs by color on the processors of BigBen, where the processors are shown in their physical locations within cabinets. The assignment of processors to jobs was done with the default placement protocol.

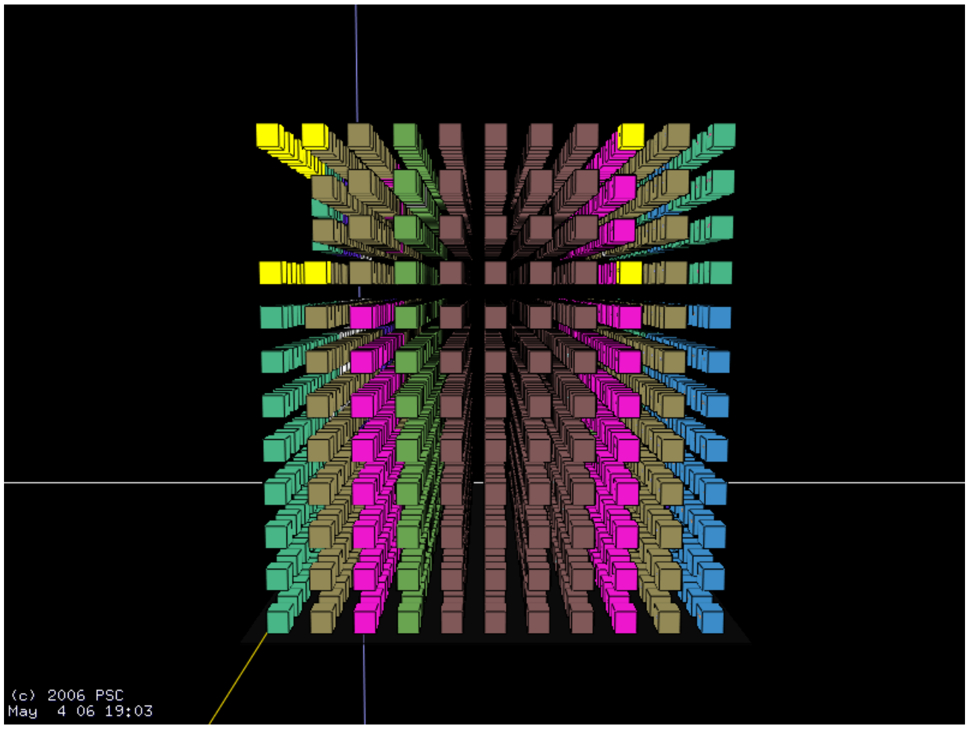


Figure 3. The processors of BigBen, colored by job, arranged by processor connectivity. Processors are adjacent only if they are directly connected.

fragmentation when using cabinet numbering as the metric: jobs are clustered together in nearby cabinets.

If we examine which processors are actually connected to each other, the inherent fragmentation of this policy becomes clear. Figure 3 shows processor connections; processors are adjacent only if they are directly connected to each other. It is clear that the default placement protocol systematically fragments jobs, sandwiching jobs around one another. In the case of the pink job, shown in Figure 3 and in isolation in Figure 4, interprocessor communication may very well occur through processors owned by another job, in this case, the brown job. Additionally, the pink job will bear the effects of communication traffic generated by the tan job which surrounds it. When the mix of jobs is communication-intensive, as is the norm on the XT3 because of its high-performance interconnect, this arrangement is clearly less than ideal. In the next section, we quantitatively demonstrate the effects of fragmentation on the performance of important, representative communication-intensive applications.

3.2 Quantifying the Effects of Fragmentation

In order to quantify the effects of job placement, we measured the performance of several communication-intensive applications under different processor allocation algorithms. We used actual production codes and a communication-intensive benchmark.

Toward our goal of optimizing performance on our production system, we strove to simulate realistic conditions, neither idealistically good nor pathologically bad, and gather results on important, representative user codes. We validated our experiments with the default job placement policy by comparing results to actual production runs under the default placement policy.

The applications we chose to use for our experiments are DNSmsp[7], a direct numerical simulation code for analysis of turbulence, NAMD[8], a scalable molecular dynamics code for simulations of large biomolecular systems, and PTRANS, a parallel matrix transpose code which is part of the HPCC benchmark[9]. These applications were chosen because they are all communication-intensive, and they cover a wide range of communication patterns. NAMD sends many small point-to-point messages, and the communication pattern is irregular. DNSmsp sends many relatively small messages in a regular communication pattern. PTRANS sends many large all-to-all messages. Additionally, NAMD and DNS are production codes important to PSC's workload.

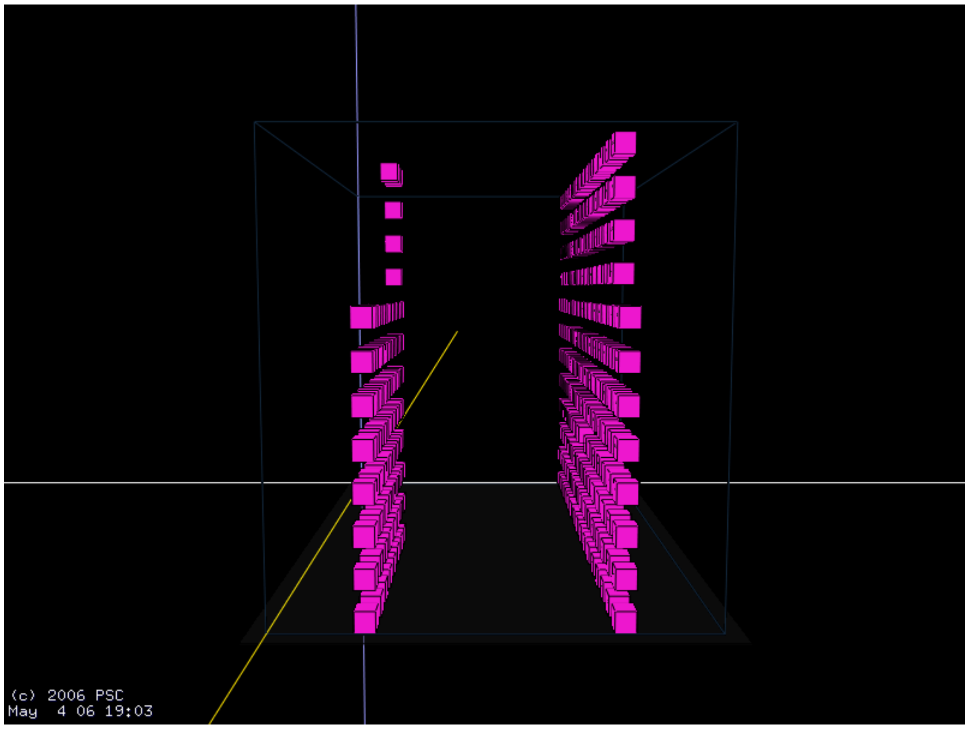


Figure 4. The processor connectivity view of one job. It is fragmented, the connected subsets of processors are not very compact, and the connected subsets of processors partition the machine. Both connected subsets consist of processors in cabinets facing each other across rows, i.e. cabinets which are connected in the *z*-dimension and therefore which are only connected along the long edges. For legibility, the processors are shown as a cube, so there are connections between processors on the faces and edges which are not shown. The two separate subsets of processors are not connected however. This topic is discussed further in Section 4.2.

Table 2. Application performance under three different scenarios: explicitly placed according to default protocol, explicitly placed according to an optimized protocol, and run as production jobs with default placement protocol. "Application" and "p" specify which combinations of applications and processor counts were run. (Each combination was run separately.) "Placed: default protocol" contains results from a controlled experiment in which the application was run concurrently with a 1024-processor PTRANS job and placed on the machine using the default protocol. "Placed: optimized for connectivity" contains results from a controlled experiment in which the application was run concurrently with a 1024-processor PTRANS job and placed on the machine so that each job was in a compact, contiguous block of connected cabinets. "Production: average" and "Production: standard deviation" contain results from at least 3 runs done at different times on the production system with other jobs running. "Improvement: optimized vs. production" is the percent improvement of the placed optimized performance over the production average.

Application	p	Placed: default protocol	Placed: optimized for connectivity	Production: average	Production: standard deviation	Improvement: optimized vs. production
PTRANS	1024	129.3 GB/s	146.5 GB/s	131.2 GB/s	21.2 GB/s	11.7%
DNS	512	316.5 s	296.0 s	310.5 s	9.2 s	4.7%
DNS	192	198.0 s	163.0 s	181.7 s	23.5 s	10.3%
NAMD	512	161.4 s	150.1 s	167.1 s	13.1 s	9.8%
NAMD	32	252.7 s	228.3 s	252.0 s	12.2 s	9.4%

The results of the experiments and comparisons to production runs, shown in Table 2, dramatically demonstrate the effects of job fragmentation for communication-intensive applications. "Placed: default protocol" and "Placed: optimized for connectivity" show the results of the experiments, in which each application was placed on an empty machine along with a 1024-processor PTRANS job to create contention. "Placed: default protocol" contains results of the experiment using the default placement protocol, in which processors were assigned both jobs in numerically sequential order. "Placed: optimized for connectivity" contains the results of the experiment using an optimized protocol, in which each job was placed in a contiguous block of connected cabinets. We assigned connected cabinets in *x*-major order (as opposed to *z*-major order) when cabinets are connected in the z-dimension, i.e. they face each other across rows, only the processors are only connected between cabinets along the long edges. When two cabinets are directly connected in the same row, every processor in one cabinet is connected to a processor in the other cabinet.[1] This topic is discussed in depth in Section 4.2.

In order to validate our simulation of contention under the default scheduling algorithm and gather data about the variance in performance, we also ran the applications as production jobs with the default job placement policy. Each application was run at least 3 times concurrently with user production jobs. "Production: average" and "Production: standard deviation" contain the averages and standard deviations of the production runs.

"Improvement: optimized vs. production" is the percentage improvement in performance using the optimized placement policy over the production average. Assigning jobs to cabinets which are directly connected to each other invariably improved performance, often by approximately 10%. This improvement is significant and translates directly to increased scientific throughput.

Two additional points are worth noting. First, there is some variability, as one would expect for scheduling on a large, production system. However, optimizing job placement for connectivity invariably reduces execution time. Second, similar effects are expected for other XT3 topologies, e.g. a 2D mesh, whenever there are more than a few cabinets per row. Finally, we expect the effect of optimized job placement to increase with XT3 system size, both because fragmentation will increase and because bandwidth-intensive applications can become increasingly sensitive to resource availability at higher processor counts.

In Section 3.3 we discuss specific changes made by PSC to the job scheduling infrastructure. In Section 4, we discuss details of some job placement protocols being explored at PSC. We also discuss an overall strategy of incorporating a combination of job layout protocols to accommodate PSC's varied workload and users' priorities and to reduce fragmentation.

3.3 PSC Modifications to the Scheduling Infrastructure to Support Customized Job Placement

PSC modified the job scheduling infrastructure, including PBS Pro[10] to support dynamic assignment of processors to jobs by moving processor to job assignment from the CPA (compute processor allocator) to the scheduler[11]. The default assignment of processors to jobs had previously been made by the CPA, always in numerical order from low to high from the list of free processors, and the scheduler merely provided the CPA with a count of the number of free processors. At PSC, the scheduling infrastructure was modified so that the list of free processors is passed to the MOM from the scheduler, and the MOM in turn passes the list of processors for a given job to the CPA. A new CPA data structure was added to contain the processor list.

These modifications enable flexible, dynamic job scheduling. The processor allocation algorithm can be modified on the fly, and processors can be allocated differently for different jobs.

4. Ongoing and Future Work

A flexible, dynamic job placement strategy which supports different job placement algorithms best suits the diverse workload and priorities of users at PSC, while reducing fragmentation. We first describe our overall strategy for using a combination of job placement protocols and then discuss specific job placement protocols in detail.

4.1 Overall Strategy: Different Placement Protocols for Different Jobs

Some applications are more sensitive to job placement than others, and users have different preferences regarding the balance between performance and throughput. Our goal is an overall job placement strategy which accommodates our varied workload and user needs while keeping fragmentation in check. We provide several different options for job placement:

[1] This assumes that both cages are fully populated with compute processors; otherwise there may be some unmatched processors.

Required shape: For a given *x*, *y*, and *z*, the scheduler waits until [connected] free processors in the required shape are available. With some codes, communication patterns are well-understood, and it is possible to directly determine optimal *x*, *y*, and *z* dimensions for job layout. In other cases, optimized dimensions can be inferred through pair-wise communication profiling with CrayPat.

Preferred shape: For a given *x*, *y*, and *z*, the scheduler tries to place the job on [connected] free processors in the preferred shape if they are available. Otherwise it places the job on available free processors, attempting to minimize communication interference between jobs as described in Section 4.2.

Optimized for communication: The scheduler places the job on free processors, attempting to minimize communication interference between jobs as described in Section 4.2.

Not optimized for communication: The scheduler places the job on scattered clusters of free processors, starting with the smallest cluster first, reserving large contiguous blocks for communication-intensive jobs.

4.2 Minimizing Communication Interference Between Jobs and Avoiding Excessive Fragmentation Over Time

In this section we discuss job placement protocols in the case where optimization for communication is desired, and explicit *x*, *y*, and *z* dimensions are not being used. We are currently evaluating protocols for job layout in terms of performance over time, which is affected by fragmentation as well as communication interference and latency. Over time, fragmentation of free processors is inevitable. We seek a balance between minimizing communication interference by placing jobs compactly on processors and minimizing fragmentation over time. We are evaluating performance and fragmentation over time using protocols described below.

4.2.1 Processor Ordering

If the job placement protocol always attempts to assign jobs to processors which are connected in cubes or near-cubes for example, initially communication contention will be low and compactness will be high. Given a mix of job sizes, fragmentation will be an issue over time, however, reducing opportunities for compact, contiguous job placement. We are exploring approaches which can sustain optimized job placement over time on a production system with a varied workload.

The job placement strategy we used in our optimized experiment was to assign jobs to processors which are in directly connected cabinets. In this strategy we assign free processors by cabinet in *x*-major order:

0 – 2 – 4 – 6 – 8 – 10 – 9 – 7 – 5 – 3 – 1 – 0 …

When all processors of one row are assigned, processors are assigned from the other row in the identical order.

The reason that the list of free processors is in *x*-major order is that on the Cray XT3, most connections which leave a cage or a cabinet are along the *x*-axis, as shown in Table 3. Of the connections which leave a cage, 72.7% are along the *x*-axis, and of the connections which leave a cabinet, 88.9% are along the *x*-axis. 100% of *x*-dimension connections are between 2 cabinets, 25% of *y*-dimension connections are between 2 cages (always internal to a cabinet), and 12.5% of *z*-dimension connections are between 2 cabinets. Thus when considering two cabinets facing each other across a row, i.e. connected in the z-dimension, only 12.5% of the processors in one cabinet are connected to a processor in another cabinet. When two cabinets are directly connected in the same row, every processor in one cabinet is connected to a processor in the other cabinet[1].

In order to increase compactness and reduce fragmentation, we are also examining a strategy, currently in use at PSC, where we assign jobs to processors in contiguously connected cages. In this strategy we assign free processors by row, cage position within a cabinet, and cabinet, so that successive cages, designated by

Table 3. Each processor has 2 connections in each dimension, one for each direction. This table contains the number of external connections per cage and per cabinet by dimension. It also contains the percentage of all connections leaving a cage or cabinet in each dimension.

Dimension	Connections Leaving Cage[2]		Connections Leaving Cabinet	
	Number (out of 64)[3]	% of those leaving	Number (out of 192)[2]	% of those leaving
x	64	72.7	192	88.9
y	16	18.2	0	0
z	8	9.1	24	11.1

[1] This assumes that both cabinets are fully populated with compute processors; otherwise there may be some unmatched processors.

[2] This includes connections leaving a cage and remaining in the same cabinet (*y*-dimension) and connections leaving the cabinet and therefore the cage (*x*- and *z*-dimensions).

[3] For each compute processor, there are 2 connections for each dimension, one for each direction. Therefore there are 2 × 32 connections per cage and 2 × 96 connections per cabinet when the cage is fully populated with compute processors; otherwise the numbers vary slightly.

(*cabinet*, *cage position*, *row*) triples, are directly connected to each other in *x*-major order. Table 4 shows the (*cabinet*, *cage position*, *row*) order we use.

Since all *x*-dimension connections are between two cabinets, allocating processors by cage in *x*-major order will generally result in more compact sets of processors than allocating by cabinet. Every cage of processors forms a subplane in the connection graph shown in Figure 3. Each processor is connected to at most 4 processors in its cage (and cabinet) out of 6 connections. Two cages in the same cabinet are only connected along one edge. When two cages in the same cage position (0, 1, or 2) and row (0 or 1) are in directly connected cabinets, every processor in one cage is connected to a processor in the other cage.[1]

4.2.2 Assigning Jobs to Processors

We are exploring different strategies for optimizing the assignment of jobs to particular groups of cabinets or cages. In the case of cage-wise job placement, we can keep track of the number of free processors within each cage. If the number of free processors is above a certain threshold, we consider that cage to be eligible for assignment. We can keep track of lists of contiguous eligible cages, ordered as in Table 4. We then assign a job to the smallest ordered sublist of contiguous, eligible cages which can accommodate the job. For example, suppose that the threshold for a cage to be considered eligible is 28. If the only eligible cages are the (*cabinet*, *cage position*, *row*) triples (1,1,0), (3,1,0), (4,1,0), (2,1,0), (0,1,0), and (0,2,0), a 48-processor job will be placed on processors in cages (1,1,0) and (3,1,0) with this protocol.

Table 4. (*cabinet*, *cage position*, *row*) ordering used to assign free processors to jobs in *x*-major cage-wise ordering, i.e. the order is (0,0,0), (2,0,0), …, (3,0,0), (1,0,0), (1,1,0), (3,1,0), …, (1,2,0), (1,2,1), …, (0,0,1)

Cabinet Order (*x*)	Cage Position (*y*)	Row (*z*)
0, 2, 4, 6, 8, 10, 9, 7, 5, 3, 1	0	0
1, 3, 5, 7, 9, 10, 8, 6, 4, 2, 0	1	0
0, 2, 4, 6, 8, 10, 9, 7, 5, 3, 1	2	0
1, 3, 5, 7, 9, 10, 8, 6, 4, 2, 0	2	1
0, 2, 4, 6, 8, 10, 9, 7, 5, 3, 1	1	1
1, 3, 5, 7, 9, 10, 8, 6, 4, 2, 0	0	1

[1] This includes connections leaving a cage and remaining in the same cabinet (*y*-dimension) and connections leaving the cabinet and therefore the cage (*x*- and *z*-dimensions)

The same strategy can be applied to cabinet-wise assignment or a more flexible cage-wise assignment strategy. In the protocol discussed above, the cages are considered to be contiguous if they occur sequentially in the (*cabinet*, *cage position*, *row*) ordering shown in Table 4. We can instead generalize this approach to consider all clusters of contiguous eligible cages, where cages are considered to be contiguous if they are directly connected in any dimension, with weighted preferences based on the number of connections and cable length.

4.3 Future Work

Future work includes supporting the ordering of processors within a job. Additionally, while we can already use CrayPat to learn about communication patterns, we are exploring the automatic generation of job layouts and processor orderings within jobs with pairwise communication profiling information obtained from CrayPat.

5. Conclusions

We have realistically quantified the effects of the default job layout protocol and an optimized protocol on communication-intensive applications on the Cray XT3. We ran our experiments on applications, including production codes, which are representative of our workload and span a variety of communication patterns under realistic conditions, verified by comparison to production runs. The results of our experiments clearly indicate the performance penalty paid by poorly placed communication-intensive applications.

We are studying different job placement protocols with the goal of minimizing communication interference and latency over time on a production machine with a varied job mix. With that goal in mind, we are investigating different combinations of job placement protocols to improve performance of communication-intensive jobs while keeping fragmentation in check over time.

Although the results and methods presented in this paper were obtained on BigBen and described in the context of the Cray XT3, they can be generalized to other 3D mesh computers.

Acknowledgments

The authors would like to thank Dennis Abts of Cray Inc. for useful discussions.

This material is based upon work supported by the National Science Foundation under Cooperative Agreement No. SCI-0456541.

About the Authors

Deborah Weisser, a member of PSC's Strategic Applications Group, is active in a variety of topics, including performance modeling, system and application performance, and parallel algorithms.

Nick Nystrom is Director of Strategic Applications at PSC, focused on advancing understanding through computational science. Nick has been active in developing and optimizing applications for Cray architectures range from the X-MP through the XT3.

Chad Vizino is a member of the Scientific Computing Systems group at PSC and is currently working on scheduling systems, resource management and accounting.

Shawn T. Brown, the Senior Support Specialist in computational chemistry at PSC, has worked in the development of quantum chemistry code and is now active in the application and development of massively parallel computational chemistry software.

John Urbanic is a Computational Science Consultant in the Strategic Applications group at PSC. He is focused on getting real applications to scale to full machine sizes and is always looking for new challenges.

The authors' email addresses are [dweisser, nystrom, vizino, stbrown, urbanic]@psc.edu.

References

[1] http://www.psc.edu/machines/cray/xt3/bigben.html
[2] L. DeRose, S. Kaufmann, D. Johnson, and B. Homer, *The New Generation of Cray Performance Tools*, CUG 2005, Albuquerque, NM, May 2005.
[3] M. Bender, D. Bunde, E. Demaine, S. Fekete, V. Leung, H. Meijer, and C. Phillips, *Communication-Aware Processor Allocation for Supercomputers*. Proceedings of the 9th Workshop on Algorithms and Data Structures (WADS), *2005.*
[4] *Cray T3E Architecture Overview*, Cray Inc.
[5] *Cray XT3 System Hardware Configuration Guide EDS-1019-1*, Cray Inc.
[6] J. Yanovich, R. Budden, and D. Simmel, XT3DMON 3D visual system monitor for PSC's Cray XT3. (2006). Native clients: http://www.psc.edu/~yanovich/xt3dmon Limited web version: http://bigben-monitor.psc.edu
[7] P. K. Yeung, S. B. Pope, and B. L. Sawford. *Reynolds number dependence of Lagrangian statistics in large numerical simulations of isotropic turbulence*. Submitted to J. Turbulence. http://www.psc.edu/science/2005/yeung.
[8] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, *NAMD2: Greater scalability for parallel molecular dynamics*. J. Comp. Phys., 151:283-312, 1999.
[9] J. Dongarra and P. Luszczek, *Introduction to the HPCChallenge Benchmark Suite*, ICL Technical Report, ICL-UT-05-01, (Also appears as CS Dept. Tech Report UT-CS-05-544), 2005.
[10] http://www.altair.com/software/pbspro.htm
[11] C. Vizino, *Batch Scheduling on the Cray XT3*, Cray User Group 2005 Proceedings, Albuquerque, New Mexico, 2005.