Comput Sci Res Dev (2011) 26: 247–256
DOI 10.1007/s00450-011-0168-y

SPECIAL ISSUE PAPER

Mapping communication layouts to network hardware characteristics on massive-scale blue gene systems

Pavan Balaji · Rinku Gupta · Abhinav Vishnu · Pete Beckman

Published online: 6 April 2011

Abstract For parallel applications running on high-end computing systems, which processes of an application get launched on which processing cores is typically determined at application launch time without any information about the application characteristics. As high-end computing systems continue to grow in scale, however, this approach is becoming increasingly infeasible for achieving the best performance. For example, for systems such as IBM Blue Gene and Cray XT that rely on flat 3D torus networks, process communication often involves network sharing, even for highly scalable applications. This causes the overall application performance to depend heavily on how processes are mapped on the network. In this paper, we first analyze the impact of different process mappings on application performance on a massive Blue Gene/P system. Then, we match this analysis with application communication patterns that we allow applications to describe prior to being launched. The underlying process management system can use this combined information in conjunction with the hardware characteristics of the system to determine the best mapping for the application. Our experiments study the performance of different communication patterns, including 2D and 3D nearest-neighbor communication and structured Cartesian grid communication. Our studies, that scale up to 131,072 cores of the largest BG/P system in the United States (using 80% of the total system size), demonstrate that different process mappings can show significant difference in overall performance, especially on scale. For example, we show that this difference can be as much as 30% for P3DFFT and up to twofold for HALO. Through our proposed model, however, such differences in performance can be avoided so that the best possible performance is always achieved.

Keywords Process mapping · Blue gene · Torus networks

This work was supported in part by the National Science Foundation Grant #0702182 and by Office of Advanced Scientific Computing Research, Office of Science, US Department of Energy, under Contract DE-AC02-06CH11357.

P. Balaji (✉) · R. Gupta
Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, USA
e-mail: balaji@mcs.anl.gov

R. Gupta
e-mail: rgupta@mcs.anl.gov

A. Vishnu
High Performance Computing Group, Pacific Northwest National Laboratory, 902 Battelle Blvd, Richland, WA 99352, USA
e-mail: abhinav.vishnu@pnl.gov

P. Beckman
Argonne Leadership Computing Facility, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, USA
e-mail: beckman@mcs.anl.gov

1 Introduction

The massive-scale systems being deployed throughout the world will utilize a large amount of shared hardware including shared caches, memory, and network infrastructure. For example, systems such as the IBM Blue Gene (BG) [1] and Cray XT [2] have started utilizing flat 3D torus networks where each node connects directly to its six neighbors. Therefore, unless each node communicates with only its physically nearest neighbors, it will be forced to share network links with other communication. Unless the physical placement of application processes matches the communication characteristics of the application, such sharing can result in significant communication contention and performance loss.

Communication libraries such as the Message Passing Interface (MPI) optimize application communication and process mapping at runtime using virtual topology functionality (such as Cartesian topologies or graph topologies). The idea of such functionality is that the application can specify the layout of the data it wants to process, and the runtime system can reorder process ranks while matching the data layout to the network hardware topology. A number of applications do not take full advantage of such functionality, however, mainly because it is not descriptive enough. That is, it does not allow the application to specify its exact communication pattern within the data layout.

For example, computational kernels such as partial differential equation solvers [3], molecular dynamics simulations [4], climate/ocean-modeling systems [5], and ray-tracing applications with domain-based parallelism [6] all use Cartesian grids of different dimensions and structure (2D, 3D, unstructured). Depending on the application and what it is trying to solve, however, the communication pattern might be completely different. For instance, partial differential equation solvers and climate/ocean-modeling systems perform nearest-neighbor communication along the edges or diagonals or both. Fast Fourier transforms (FFTs), on the other hand, communicate with all processes in each dimension. Ray-tracing applications with domain-based parallelism communicate with processes along their diagonals. With the current model of communication libraries, the application can specify that it want its processes to be laid out as a 2D or 3D Cartesian grid, but it cannot specify that it will communicate with its neighbors along the edges, or neighbors along the diagonals, or all processes in its row/column, or any other communication pattern.

Thus, in practice, many applications try to manually map processes to the network topology. While this is a reasonable approach for small to medium scale topologies, it is not a feasible solution for massive-scale systems with hundreds of thousands of processors. The important aspect is that application developers know their applications communication pattern while the application is being launched. But, there currently exists no good way to provide this information to the process management system in order to optimize the process layout while taking network hardware characteristics into account.

In previous work [7, 8], we noticed the variance of communication performance with process mapping, and we studied the network congestion characteristics of flat torus networks that cause this behavior. In this paper, we extend that research by first performing a detailed analysis of the mapping of logical process layouts on the physical network topology. Then, we take advantage of this analysis to allow the application to describe its communication pattern, estimate the expected network contention through a simple first-order approximation model, and use this estimate to identify the right mapping to use. Moreover, we study the performance benefits such a model would provide, using evaluations with various micro-benchmarks and real application kernels including P3DFFT [9] and the HALO ocean-modeling kernel [10]. Our experiments, which scale up to 131,072 cores of the largest BG/P system in the United States (using 80% of the total system size), demonstrate that different process mappings can show significant difference in overall performance, especially on large-scale systems. For example, we show that this difference can be as much as 30% for P3DFFT and up to twofold for HALO. Through our proposed model, however, such differences in performance can be avoided, so that the best possible performance is always achieved.

2 Complexity in process mapping

In this section, we describe the complexity of process mapping on large-scale systems. Specifically, in Sect. 2.1, we describe the application viewpoint of application processes and their logical layout. In Sect. 2.2, we tie such logical mapping to the physical system topology for 3D torus-based systems.

2.1 Application logical process layout

Most communication libraries, including MPI, do not expose the physical layout of the underlying system to the application in order to improve portability across different systems. Thus, applications form logical topologies of the processes available to match the problem they are trying to solve (often depending on the data layout corresponding to the application science). For example, if we consider ocean/climate modeling and other computational fluid dynamics applications, the data representation is often a 2D plane, 3D volume, or even a multidimensional unstructured grid [11]. Thus, these applications form logical grids of the available processes, with each process having a part of the overall data. The computation on the local data, however, depends on the partially evaluated results from neighboring processes on the logical process grid formed by the application.

In other computational kernels such as FFTs [9, 12], the interaction is based on the Cartesian dimension. Processes are divided into a logical multidimensional grid (e.g., 2D grid). Each process initially has data corresponding to one dimension of the data grid. Once all processes are done computing their parts, the entire data grid is transposed, resulting in each process locally getting a different dimension of the data grid. Thus, a process interacts only with other processes in each dimension it is a part of. For example, in a 2D grid, each process interacts only with $\sqrt{N}$ processes in each of the two dimensions.

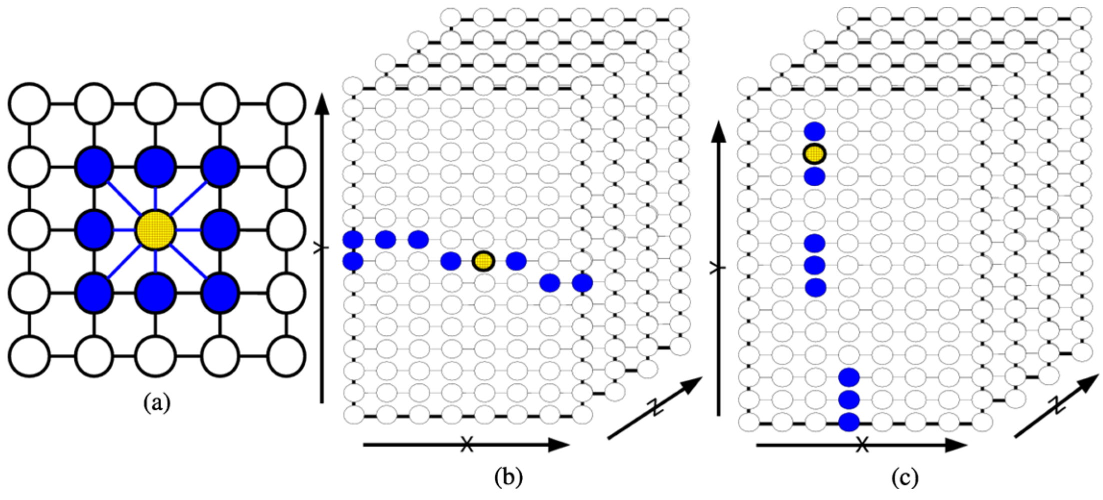


Fig. 1 2D mapping for nearest-neighbor communication: (**a**) Logical mapping, (**b**) XYZ physical mapping, (**c**) YXZ physical mapping

Similarly, in ray-tracing applications with domain-based parallelism [6], while working on one particular ray, each process interacts only with other processes that share data corresponding to that ray. This is several times a diagonal group of processes in a 2D logical grid.

The common aspect in these and many other applications is that all of them rely on a logical layout of the processes without any information about the physical placement or mapping of the actual processes.

2.2 Process mapping for BG/P

Optimal mapping of logical communication patterns onto the actual physical topology of the system is an intricate task. As the complexity of the communication patterns and system topology increases, this task becomes even more challenging. This section describes the mapping complexity for applications with different logical process layouts while being mapped onto a 3D physical system grid. The complexity is similar for other layouts such as Cartesian dimension-based communication and diagonal communication patterns, but these are not discussed here.

One-dimensional communication is one of the simpler patterns among nearest-neighbor communication patterns—every process communicates with its two logically neighboring processes. Consider mapping such an application having 512 processes onto 512 nodes connected in a $8 \times 8 \times 8$ 3D physical grid topology. Such a grid will have 8 nodes along each of the X, Y, and Z axes. The process management framework on BG/P allows application processes to be mapped onto the system in different ways, as represented by the mapping string; for example, with an XYZ mapping, processes are mapped to nodes along X-axis first, followed by Y-axis and Z-axis respectively. Thus, for a 1D logical nearest-neighbor communication, most of the application processes will get mapped next to each other in a sequential order.

Two-dimensional and three-dimensional communication patterns are more challenging to map optimally. Every process communicates with up to 8 (in 2D mapping) and 26 (in 3D mapping) logically neighboring processes.[1] Figure 1 illustrates such process mapping for 2D nearest-neighbor communication, while Fig. 2 shows 3D mapping. Figure 2(a) shows a portion of the 3D logical grid as viewed by the application. Figures 2(b) and 2(c) show the process mapping of this logical grid on to a $8 \times 16 \times 8$ physical grid with XYZ and XZY mappings, respectively. As can be seen from these figures, communicating groups of processes can get dispersed throughout the physical network, resulting in increased communication overlap between process groups and performance degradation. More important, the dispersal of processes depends heavily on the process mapping (e.g., an XYZ mapping in these figures shows a different, and slightly smaller, dispersal as compared with the YXZ mapping), which causes different mappings to have different performance characteristics.

Based on these observations, from the system perspective we can conclude that the physical layout of the application processes (and thus the communication overhead) is highly dependent on the mapping type and the dimensions of the physical grid/torus.

While in theory any mapping of processes to cores is possible, different systems have different restrictions on what mappings they allow. For example, the IBM Blue Gene/P system allows different combinations of process mappings along the X, Y, Z dimensions of the 3D torus. We note that

[1] Some applications treat diagonally intersecting processes as neighbors as well (star stencils), while some applications treat only processes intersecting along the edges as neighbors (box stencils).

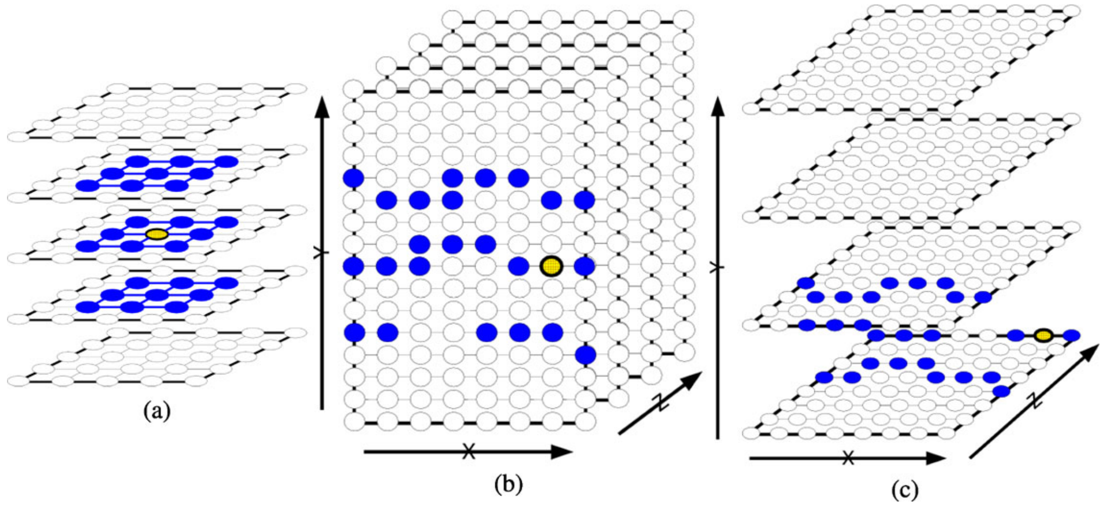


Fig. 2 3D mapping: (**a**) Logical mapping, (**b**) XYZ physical mapping, (**c**) XZY physical mapping

Table 1 Argonne BG/P system torus dimensions

Nodes	Dimensions
512	$8 \times 8 \times 8$
1024	$8 \times 8 \times 16$
2048	$8 \times 8 \times 32$
4096	$8 \times 16 \times 32$
8192	$8 \times 32 \times 32$
16384	$16 \times 32 \times 32$
32768	$32 \times 32 \times 32$

for simplicity of description we have discussed process mappings only along the X, Y, Z dimensions of the 3D torus. However, BG/P systems consist of four cores per compute node, which can be considered to be a fourth dimension represented as "T." Thus, an application can be mapped by using various mappings such as $TXYZ$, $XYZT$, and $TYXZ$.[2] Other mappings that do not form a symmetric ordering of ranks in one of these orders are not supported. Thus, the "best performance" we can achieve is artificially restricted by this requirement.

Further complicating this issue, applications on the BG/P system run on a subset of nodes defined by a "partition." Two different partitions can have the same number of cores while having completely different dimension sizes. For example, partitions of both $8 \times 16 \times 32$ and $16 \times 16 \times 16$ dimensions can be used for a 4,096-process job. The dimensions of different partition sizes of the torus on the BG/P system can be configured by the system administrator and would determine how much network sharing, and consequently contention, each mapping would cause for any given partition shape. Table 1 shows the dimensions of the torus for varying system sizes for the Argonne BG/P system.

3 Communication contention model

In this section, we describe our methodology for analyzing different communication patterns of applications and mapping them optimally.

The basic idea of our approach is to (1) understand the communication pattern of the application by allowing the application to describe it, (2) understand the physical platform network topology and the routing algorithm behavior on the platform, and (3) map these two categories of information to calculate the network contention for the given application pattern on the available network. The contention analysis model, as its output, predicts an optimal mapping that would give the best performance for a given application on a given platform. We have chosen the IBM BG/P system as an example platform for this paper. This information, consisting of application pattern and its optimal mapping, can then be integrated in the IBM BG/P job launching system. Applications can then specify their application pattern during job submission time, and the optimal mapping can be automatically and transparently picked up by the runtime system while launching the application.

Our current analysis is specific to symmetric communication patterns where all processes are involved in similar communication. Instances of such symmetric communication include 1D, 2D, and 3D nearest-neighbor communication patterns, which we analyze in this section.

[2] $TXYZ$ indicates that Message Passing Interface processes are ordered with priority given first to cores of a node, followed by nodes on the X-axis, Y-axis, and Z-axis, respectively.

3.1 Routing on Blue Gene/P

Our contention analysis model relies on the routing algorithm used by BG/P. Specifically, for small messages, BG/P uses a static routing algorithm in a dimension-wise order (e.g., first along the X-axis, then the Y-axis, and finally the Z-axis). For large messages, BG/P uses a destination-based adaptive routing algorithm. To avoid livelocks, however, it always picks one of the minimum distance routes. In other words, at each hop, the adaptive routing algorithm considers only the outgoing links that reduce the hop count to the destination node. Among the outgoing links being considered, it selects the one with the least amount of other ongoing traffic.

As a first-order approximation, we assume that the data packets in a large message are split equally on all possible paths at each hop. This approximation allows us to estimate the amount of data traffic on each hop caused by the communication between any pair of processes.

We, currently, consider only symmetric communication patterns in our research. We note that every partition on the BG/P is fully symmetric and forms a torus on all dimensions. This symmetry allows us to model the contention caused by a single process for its communication with all its peers, and simply extend it to all the processes in the entire system.

3.2 Example contention analysis

To analyze the optimal topology for a given physical partition layout and application, our model needs to understand the contention that may arise because of shared network links during the application communication. We explain the logic used in our model through an example.

Consider a 128×128 2D process grid (16,384 cores). On BG/P, this will require 4,096 quad-core nodes. The partition dimensions for 4,096 nodes on the Argonne BG/P are $8 \times 16 \times 32$ along the X-, Y-, and Z-axis, respectively.

Let us analyze the $TXYZ$ mapping of processes, with a row-major mapping of the 2D 128×128 processor logical grid. Each row of 128 processors of the logical grid will occupy the four rows of the X-axis (8 nodes; i.e., 32 cores lie along the X-axis). Now consider a node communicating with its eight nearest neighbors. Its neighbors on either side of it, on the same row, will lie either on the same physical node or on a neighboring node (since all cores of a node are allocated before the next node), thereby incurring minimal communication overhead (no link contention). The three neighbors on the row above it will lie on nodes on the same X-axis at a distance of four hops (since one row of the 2D grid spans four rows of the physical torus) from its node. Thus, each link along these four hops will be used three times by this communicating center node. Each of these links along the four hops will be used by other sets of nearest-neighbor communicating nodes as well, in both directions bringing its average contention count to 24.[3]

In a $TZXY$ mapping, on the other hand, the Z-axis has 32 nodes (i.e., 128 processors along its axis). Thus, a single row of the 2D grid can map along a single Z-axis plane. Analyzing this reveals that for every node, the neighbors above or below a process are mapped to a single node, one hop away. On average, the bidirectional contention count for every link is 6. Similarly an $XYZT$ mapping would have an average bidirectional contention count for every link of about 10.5. Thus, contention seen by the $TZXY$ mapping is the least, while that seen by $TXYZ$ is the highest. This analysis is substantiated by the results presented in Sect. 4. Note that for completely symmetric communication patterns such as nearest neighbor, it is sufficient to evaluate the contention caused by one process.

We performed similar analyses for other communication patterns such as dimension-wise communication (used by 3DFFT libraries) and diagonal communication (used by ray-tracing applications). However, descriptions for those have been omitted because of space restrictions.

Contention counts will vary with the type of mapping, type of communication, partition size of the system, logical grid size of the application and order (row major, column major, etc.) in which logical grid rows are mapped onto the partition cores. In the rest of the paper, because of lack of space, we do not analyze contention count further. Rather, we demonstrate the results obtained from our experiments on the BG/P system.

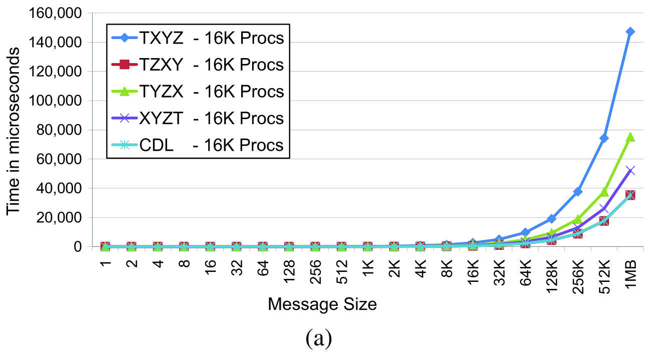
4 Experiments and analysis

In this section, we discuss the impact of various mappings on different micro-benchmarks and applications. Our experiments also demonstrate the performance achieved by the model described in Sect. 3 (called 'Contention Detection Model' and represented by "CDL" in the charts), which is automatically and transparently picked by the runtime system when launching the application.

4.1 Microbenchmark-based evaluation

We first evaluate 2D and 3D logical nearest-neighbor communication. Both experiments use star stencils, so each process has $(3^d - 1)$ neighbors, where d is the logical process grid dimensionality.

[3]"Contention count" can be considered to be an indication of how many communication streams are going over a link at the same time.



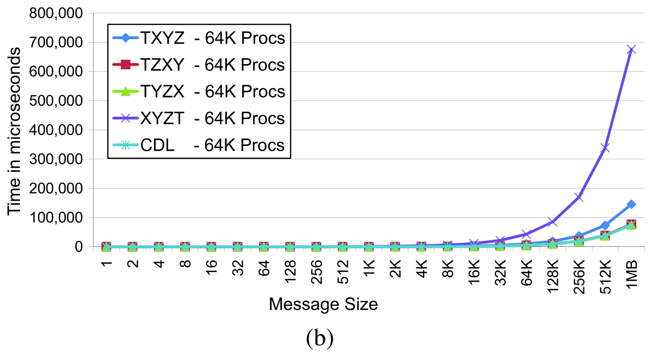
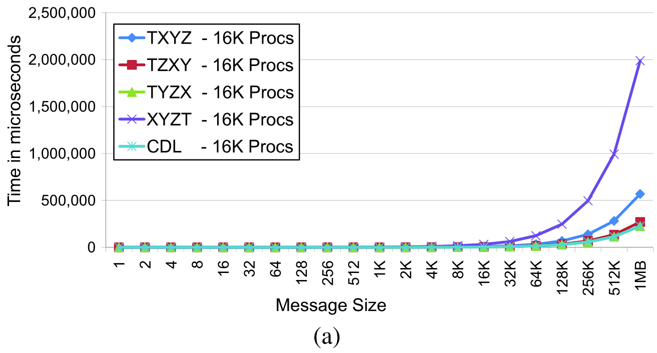


Fig. 3 2D communication benchmark performance: (**a**) 16 K cores, (**b**) 64 K cores



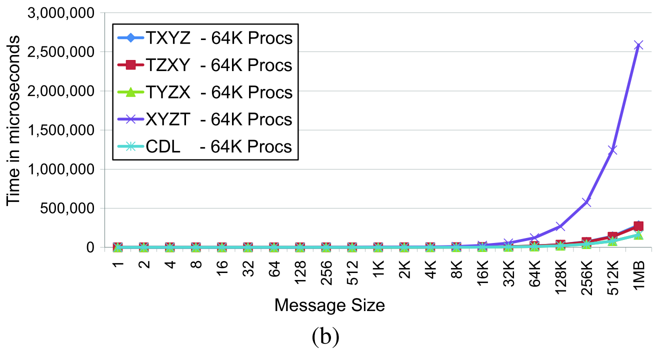


Fig. 4 3D communication benchmark performance: (**a**) 16 K cores, (**b**) 64 K cores

2D nearest-neighbor communication In this experiment, each process does point-to-point communication exchanging some data with its logical neighbors. In real application kernels, this data would typically correspond to the *ghost cells* that form the bordering data points between the two processes. Figure 3(a) shows the communication performance for a system size of 16,384 processes, and Fig. 3(b) shows the same for a system size of 65,536 processes.

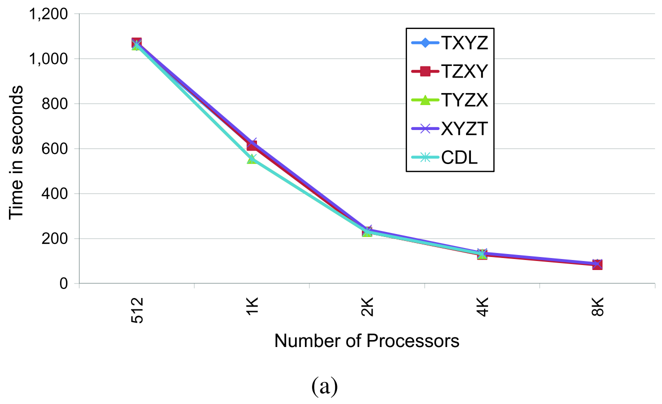
From the figure, we see that the difference in performance between the various mappings increases with both message size (as shown in each subfigure) and system size (comparing the two subfigures). In fact, for a system size of 65,536 cores and a message size of 1 MB, we notice almost a *sevenfold* difference in performance between the different mappings. This behavior is expected; as the message size increases, each network link is used for longer periods of time, increasing the possibility of congestion and performance degradation. Similarly, as the system size increases, the number of communication flows increases proportionally with the number of *pairs* of processes available, that is, $O(N^2)$, where N is the number of processes in the system. But the number of network links increases only as $O(N)$; this situation leads to more congestion and performance degradation. The results in Fig. 3(a) also show that CDL model can determine which layout has the least contention (in this case $TZXY$) and automatically pick that for the user. These results are similar to those reported in Sect. 3.

3D nearest-neighbor communication This experiment is similar to the 2D nearest-neighbor experiment except that processes are logically laid out in three dimensions. Thus, each process has 26 logically nearest neighbors with which it performs point-to-point communication. Because of space restrictions, we simply point out, as can be seen in Figs. 4(a) and (b), that the overall trend is similar to that of the 2D nearest-neighbor benchmark: (1) there is a significant performance difference between different mappings, (2) the performance difference increases with message size and system size, and (3) the CDL model allows the system to automatically pick the best mapping. In fact, for 65,536 processes and a message size of 1 MB, we notice an order-of-magnitude difference in performance.

4.2 Evaluation of application kernels

Next, we use two application kernels, P3DFFT and Halo, to analyze the impact of process mapping on overall performance.

P3DFFT library FFT [13] is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse.



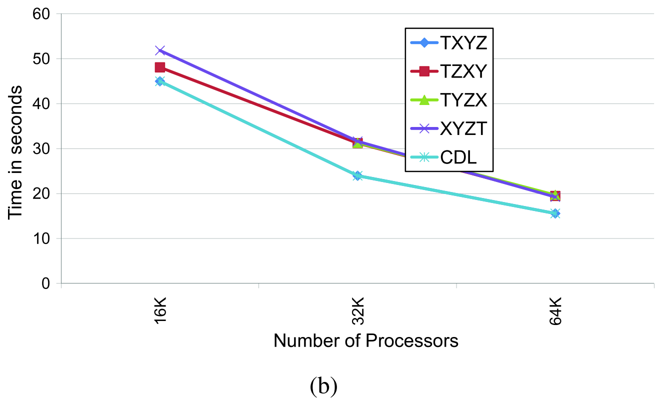
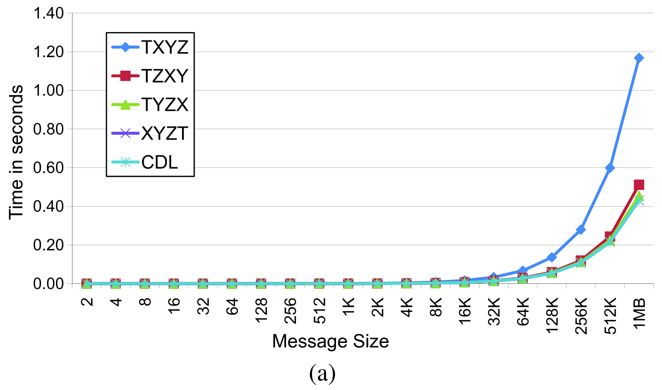


Fig. 5 P3DFFT performance: (**a**) 512 to 8 K cores, (**b**) greater than 8 K cores



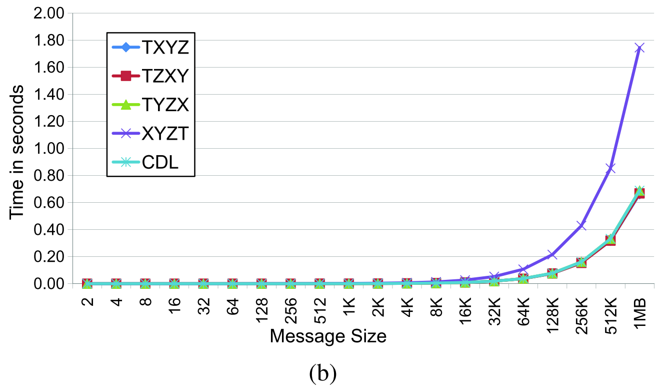


Fig. 6 Halo application on (**a**) 32 K cores, (**b**) 128 K cores

FFT, as applied to a three-dimensional volume of space, is typically referred to as 3DFFT. The goal of 3DFFT is essentially to perform 1D Fourier transform on each of the three dimensions of the 3D data mesh. Most parallel 3DFFT implementations rely on a sequential version of 1DFFT (that performs the transform on one dimension at a time) and then transpose the data grid when needed. Multiple implementations of parallel 3D FFT are available today. P3DFFT [14, 15] is a popular implementation of 3DFFT used in various application domains, including digital speech and signal processing, solving partial differential equations, molecular dynamics simulations, and Monte Carlo simulations.

Figure 5 shows the performance of P3DFFT with increasing system size for different process mappings. For viewability, we split the data into two graphs—Fig. 5(a) shows the performance for small system sizes (up to 8,192 cores), while Fig. 5(b) shows the performance for large system sizes (more than 8,192 cores). As shown in the figures, for small systems, process mapping has no significant impact. When the system size increases beyond 8,192 cores, however, we do notice a performance difference between the different mappings. In fact, this difference is as high as 25% for 65,536 cores, which makes the Contention Detection Model's (CDL) benefits substantial for large-scale systems.

The Halo application kernel The NRL Layered Ocean Model (NLOM) [16] simulates semi-enclosed seas, major ocean basins, and global ocean. The current implementation of the model uses a tiled data-parallel programming style. Its general nature allows implementations in various programming models including MPI, OpenMP, Co-Array Fortran, and shared memory. This makes NLOM a good candidate for benchmarking both hardware and the associated communication software. The Halo benchmark [10] simulates an NLOM 2-D exchange for an $N \times N$ subdomain for different values of N. In other words it performs nearest-neighbor exchanges from a 2D array. Each of these exchanges is performed by using several algorithms, different for each programming technique. When Halo is used on a smaller number of nodes, the performance is affected predominantly by the system latency, whereas for larger node counts the bandwidth becomes a dominating factor [10].

Figure 6 shows the performance of Halo with varying message size, for different process mappings (system sizes of 32,768 cores and 131,072 cores). Again, we notice a significant improvement in overall performance for the CDL model, depending on the mapping. For 131,072 cores, we notice more than *twofold* difference in performance. Figure 7 shows the performance of Halo with varying system

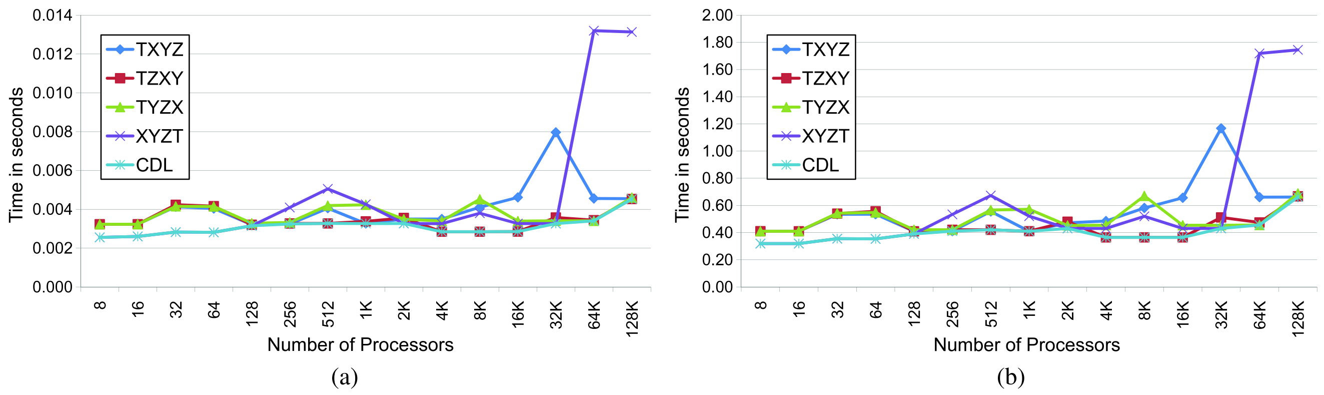


Fig. 7 Halo application trends on varying system size: (**a**) 8 KB message size, (**b**) 1 MB message size

sizes (message sizes of 8 KB and 1 MB). We notice three trends: (1) no single mapping performs the best in all cases, which is also true for all other results shown in the paper; (2) the overall performance difference between the best mapping and the worst mapping generally increases with system scale; and (3) the CDL model picks the best mapping in all cases.

5 Related work

In the past few years, a lot of research has been devoted to studying techniques for mapping application topology (tasks) onto the system's actual physical topology [17–20] and to optimizing these techniques [21–23]. Research [24–27] provides different methods for optimizing problem layout on BG/L. All this research assumes that applications can describe their communication patterns effectively to the runtime system; however, this is not currently the case. Most mapping techniques assume a possible communication pattern based on the data layout and optimize the process layout accordingly.

The NAS Parallel Benchmarks [28] were evaluated with different mappings such as Gray codes and permutations of the X, Y, Z coordinates on the BG/L systems spanning up to 256 processors. While that work is similar to a subset of research proposed here, it was done only with microbenchmarks and on very small system size. None of the prior work has shown these effects at the scales we have shown. Our experiments are done on a scale nearly 100 times larger than the previous papers (80% of the largest BG/P system in the United States). Since network sharing is going to increase in future exascale systems, scale of the applications is key to understanding network saturation behavior.

Work also has been done in the context of STAR-MPI, which focuses on adaptive techniques for various collective operations and utilizes the most optimal communication algorithm [29]. While such tools can help choose the best communication algorithm for a given system process mapping, they do not explicitly try to understand the network congestion behavior caused by different mappings; thus, they are mostly complementary to the work done in this paper.

6 Concluding remarks

We analyzed the impact of different process mappings on application performance on a massive BG/P system. Using this analysis, we propose a communication description model/language (CDL) that applications can use to specify their communication patterns prior to being launched. We demonstrated the performance benefits the CDL model can provide, using microbenchmarks and real application kernels including P3DFFT and the Halo ocean-modeling kernel. Our experiments, which scaled up to 131,072 cores of the Blue Gene/P system indicate that different process mappings can show significant difference in overall performance, especially on large-scale systems. Through our CDL model, however, one can avoid these differences in performance, thereby obtaining the best possible performance.

References

1. IBM Blue Gene Team (2008) Overview of the IBM Blue Gene/P project. IBM J Res Dev 52(1–2):199–220
2. Cray Research, Inc (1993) Cray T3D system architecture overview
3. Argonne National Laboratory. PETSc. http://www.mcs.anl.gov/petsc
4. Kumar S, Huang C, Almasi G, Kale LV (2007) Achieving strong scaling with NAMD on Blue Gene/L. In: IEEE international parallel and distributed processing symposium
5. Naval Research Laboratory. Naval research laboratory layered ocean model (NLOM). http://www.navo.hpc.mil/Navigator/Fall99_Feature.html
6. Rabiti C, Smith MA, Kaushik D, Yang WS, Palmiotti G (2008) Parallel method of characteristics on unstructured meshes for the UNIC code. In: PHYSOR, Interlaken, Switzerland, 14–19 Sept 2008

7. Balaji P, Chan A, Thakur R, Gropp W, Lusk E (2009) Toward message passing for a million processes: characterizing MPI on a massive scale blue gene/P. J Comput Sci Res Devel Special edn (presented at the International supercomputing conference (ISC)); Best Paper Award
8. Balaji P, Naik H, Desai N (2009) Understanding network saturation behavior on large-scale blue gene/P systems. In: Proceedings of the international conference on parallel and distributed systems (ICPADS), Shenzhen, China, 8–11 Dec 2009
9. Pekurovsky D (2009) P3DFFT webpage, Feb 2009. http://www.sdsc.edu/us/resources/p3dfft/index.php
10. Wallcraft AJ (1999) The Halo benchmark. http://www.navo.hpc.mil/Navigator/PDFS/Fall1999.pdf
11. Fischer P, Lottes J, Pointer D, Siegel A (2008) Petascale algorithms for reactor hydrodynamics. J Phys Conf Ser 125(1). doi:10.1088/1742-6596/125/1/012076
12. Frigo M, Johnson SG (2005) The design and implementation of FFTW3. Proc IEEE 93:216–231
13. Cooley JW, Tukey JW (1964) An algorithm for the machine calculation of complex Fourier series. Math Comput 19(90):297–301
14. San Diego Supercomputing Center. P3DFFT. http://www.sdsc.edu/us/resources/p3dfft/
15. Chan A, Balaji P, Gropp W, Thakur R (2008) Communication analysis of parallel 3D FFT for flat Cartesian meshes on large blue gene systems. In: Proceedings of the IEEE/ACM international conference on high performance computing (HiPC), Bangalore, India, 17–20 Dec 2008
16. Wallcraft AJ (1991) The NRL layered ocean model users guide. NOARL Report 35, Naval Research Laboratory, Stennis Space Center, MS
17. Traff J (2002) Implementing the MPI process topology mechanism. In: SC, pp 1–14
18. Hur J (1999) An approach for torus embedding. In: ICPP, Washington, DC, USA. IEEE Computer Society, Los Alamitos, p 301
19. Ou C, Ranka S, Fox G (1996) Fast and parallel mapping algorithms for irregular problems. J Supercomput 10(2):119–140
20. Bokhari S (1981) On the mapping problem. IEEE Trans Comput 30(3):207–214
21. Bollinger SW, Midkiff S (1991) Heuristic technique for processor and link assignment in multicomputers. IEEE Trans Comput 40(3):325–333
22. Mansour N, Ponnusamy R, Choudhary A, Fox GC (1993) Graph contraction for physical optimization methods: a quality-cost tradeoff for mapping data on parallel computers. In: ISC, New York, NY, USA. ACM, New York, pp 1–10
23. Chockalingam T, Arunkumar S (1992) Randomized heuristics for the mapping problem. The genetic approach. In: Parallel computing, pp 1157–1165
24. Bhagnot G, Gara A, Heidelberger P et al (2005) Optimizing task layout on the Blue Gene/L supercomputer. IBM J Res Dev 49(2–3):489–500. doi:10.1147/rd.492.0489
25. Almasi G, Archer C, Castanos J et al (2004) Implementing MPI on the BlueGene/L Supercomputer. In: Euro-Par, pp 833–845
26. Yu H, Chung I, Moreira J (2006) Topology mapping for Blue Gene/L supercomputer. In: SC, New York, NY, USA. ACM, New York, p 116
27. Agarwal T, Sharma A, Laxmikant A, Kale LV (2006) Topology-aware task mapping for reducing communication contention on large parallel machines. In: IPDPS, p 122
28. Smith B, Bode B (2005) Performance effects of node mappings on the IBM BlueGene/L machine. In: Euro-Par, pp 1005–1013
29. Faraj A, Yuan X, Lowenthal D (2006) STAR-MPI: self tuned adaptive routines for MPI collective operations. In: Proceedings of the 20th annual international conference on supercomputing (ICS), Cairns, Queensland, Australia, pp 199–208

Pavan Balaji holds a joint appointment as an Assistant Computer Scientist at the Argonne National Laboratory, a research fellow of the Computation Institute at the University of Chicago, and as an adjunct associate professor at the Institute of Software of the Chinese Academy of Sciences. He had received his Ph.D. from the Computer Science and Engineering department at the Ohio State University. His research interests include parallel programming models and middleware for communication and I/O, high-speed interconnects, efficient protocol stacks, cloud computing systems, and job scheduling and resource management. He has more than 75 publications in these areas and has delivered more than 100 talks and tutorials at various conferences and research institutes. He has received several awards for his research activities including an Outstanding Researcher award at the Ohio State University, the Director's Technical Achievement award at Los Alamos National Laboratory, and several best paper and other awards. Dr. Balaji has also served as a chairman or editor in nearly two-dozen journals, conferences and workshops including CCGrid, JHPCA, ICPP, IEEE Micro, Hot Interconnects, P2S2 workshop, and ICCCN, and as a technical program committee member in numerous conferences and workshops. He is a member of the IEEE and ACM. More details about Dr. Balaji are available at http://www.mcs.anl.gov/~balaji.

Rinku Gupta is a senior scientific developer at Argonne National Laboratory and the lead developer for the Fault Tolerance Backplane project. She received her M.S. degree in Computer Science from Ohio State University in 2002. She has several years of experience developing systems and infrastructure for enterprise high-performance computing. Her research interests primarily lie towards middleware libraries, programming models and fault tolerance in high-end computing systems. More details about Ms. Gupta are available at http://www.mcs.anl.gov/~rgupta.

Abhinav Vishnu is a research scientist in the high performance computing group at the Pacific Northwest National Laboratory. He received his Ph.D. from the Computer Science and Engineering Department at The Ohio State University in 2007. His research interests including scalable, fault tolerant and energy efficient parallel programming models, communication runtime systems, high speed interconnects and cloud computing systems. Dr. Vishnu has many publications cross-cutting the areas of scalability, energy efficiency, fault tolerance, with several demonstrations of fault tolerance using partitioned global address space models and scientific applications. Dr. Vishnu has served as co-chairman of International

Workshop on Parallel Programming Models and Systems Software for three years and co-editor of a special issue on International Journal of High Performance Computing and Applications for two years. He has also served as a technical program committee member in multiple conferences including CCGrid, IC3N, Cluster Computing, and HiPC. Details about Dr. Vishnu are available at http://hpc.pnl.gov/people/vishnu.

Pete Beckman is a recognized global expert in high-end computing systems. During the past 20 years, he has designed and built software and architectures for large-scale parallel and distributed computing systems. After receiving his Ph.D. degree in computer science from Indiana University, he helped found the university's Extreme Computing Laboratory. In 1997 Pete joined the Advanced Computing Laboratory at Los Alamos National Laboratory.

In 2000 he established a Turbolinux-sponsored research laboratory in Santa Fe that developed the world's first dynamic provisioning system for cloud computing and high performance computing (HPC) clusters. The following year, Pete became Vice President of Turbolinux's worldwide engineering efforts, managing development offices worldwide.

Pete joined Argonne National Laboratory in 2002 as Director of Engineering. Later, as Chief Architect for the TeraGrid, he designed and deployed the world's most powerful Grid computing system for linking production HPC computing centers. Pete then started a research team focusing on petascale high-performance software systems, wireless sensor networks, Linux, and the SPRUCE system to provide urgent computing for critical, timesensitive decision support.

In 2008 he became Director of the Argonne Leadership Computing Facility, home to one of the world's fastest open science supercomputers. He is currently the director of the Exascale Technology and Computing Institute, where he leads Argonne's exascale computing strategic initiative.