Optimizing task layout on the Blue Gene/L supercomputer

G. Bhanot
A. Gara
P. Heidelberger
E. Lawless
J. C. Sexton
R. Walkup

A general method for optimizing problem layout on the Blue Gene®/L (BG/L) supercomputer is described. The method takes as input the communication matrix of an arbitrary problem as an array with entries $\boldsymbol{C(i, j)}$, which represents the data communicated from domain $\boldsymbol{i}$ to domain $\boldsymbol{j}$. Given $\boldsymbol{C(i, j)}$, we implement a heuristic map that attempts to sequentially map a domain and its communication neighbors either to the same BG/L node or to near-neighbor nodes on the BG/L torus, while keeping the number of domains mapped to a BG/L node constant. We then generate a Markov chain of maps using Monte Carlo simulation with free energy $\boldsymbol{F = \Sigma_{i,j}\, C(i, j) H(i, j)}$, where $\boldsymbol{H(i, j)}$ is the smallest number of hops on the BG/L torus between domain $\boldsymbol{i}$ and domain $\boldsymbol{j}$. For two large parallel applications, SAGE and UMT2000, the method was tested against the default Message Passing Interface rank order layout on up to 2,048 BG/L nodes. It produced maps that improved communication efficiency by up to 45%.

Introduction

The Blue Gene*/L supercomputer (BG/L) [1] is a massively parallel computer with two communication networks: a nearest-neighbor network with the topology of a three-dimensional (3D) torus and a global collective network. In normal use, the torus is the primary communications network and is used both for point-to-point and for many global or collective communications. The collective network is used for collective communications, such as `MPI_REDUCE`.

Compute nodes on the BG/L are logically arranged into a 3D lattice, and the torus communications network provides physical links only between nearest neighbors in that lattice. Therefore, all communications between nodes must be routed in a manner that makes use of the available physical connections, and the cost of communications between nodes will vary depending on the distance between the nodes involved. The challenge is to optimally map an arbitrary parallel application of Message Passing Interface (MPI) tasks that minimizes the total execution time. In most cases, total execution time is the sum of the time for communication and the time for computation.

In this paper, we focus on the problem of minimizing the communication time. We describe a general method for optimizing the mapping of parallel applications to a grid of nodes for which communications costs are dependent on task placement. For the purpose of illustration, we restrict the discussion to point-to-point communications and collective communications that are implemented using these point-to-point communications, and so focus only on optimizations involving the torus network of BG/L. Collective communications using the global collective network and compute load imbalance between domains (if they exist in the application) can be included in the analysis by modifying the cost function appropriately.

The compute nodes of BG/L typically have 512 MB of memory and two central processing units (CPUs), each capable of a peak performance of 2.8 gigaflops (Gflops). Thus, the peak performance of the node is 5.6 or 2.8 Gflops depending on whether the CPUs are both used for computation or one is used for computation and the other for communication. Each compute node has six torus links built in. Each is connected to its six nearest neighbors in the $x+$, $y+$, $z+$, $x-$, $y-$, $z-$ directions, respectively, so there is one hop between nearest

0018-8646/05/$5.00 © 2005 IBM

neighbors. The links on the torus have a peak bandwidth of 1.4 Gb/s. Each node can communicate with its neighbors at the rate of 1.4 Gb/s in each direction.

For our purposes, we assume that an application consists of a number of domains. Computations are performed in one domain before information is accessed from other domains, followed by more computations, and so on. *Communication traffic* is the amount of data exchanged between domains. We assume that one or more domains are mapped to a single BG/L node. We seek to find the optimal mapping of the domains to BG/L nodes such that the communication time is minimized. Note that the definition of domain depends on how the communication data is collected. In the examples used in the present paper, since the data was obtained from runs of the MPI-based applications code on BG/L, the domain was an MPI task. However, a domain might just as well be a single variable or a collection of variables. If the code is used for dynamic load balancing, the data could also be collected at runtime. In this case, a domain could be a process or system thread.

For the purpose of this paper, we assume that computation time is uniform across processors and that our cost function is required to minimize only communication time. We note that the information necessary as input into our method can be obtained from the runtime system (RTS) during the execution of an application or can be provided as data from a prior test run.

Related work

The problem of assigning tasks to the processors of a parallel processing computer in a manner that achieves the optimal load balance and minimizes the cost of interprocessor communication is important to the efficient use of parallel computers. Many groups have done work on this problem in recent years. However, the relative emphasis placed on computational balance as opposed to communication costs and the different assumptions made concerning the number of processors and the interprocessor network architecture have led to many different approaches to the problem.

For small numbers of processors, many techniques can be applied successfully. For instance, a simple heuristic followed by iterative improvement processes is developed in [2] to optimize the layout of tasks on processor networks with up to 64 processors. This work is unusual in that it includes link contention and total traffic volume during layout. A more complex algorithm, presented in [3], produces good results for small numbers of heterogeneous processors. However, it is assumed that communication costs are independent of the communication endpoints, and so, while it is useful for processors linked by a switched network, it is less applicable to parallel computers using more complex network topologies.

Graph-partitioning techniques have been used in the load-balancing problem and also in clustering tasks to optimize locality in hierarchical networks (for instance, a cluster of symmetric multiprocessing nodes linked by a switched network [4]). Graph bipartitioning has also been used for task clustering and mapping on eight-node hypercubes [5]. Simulated annealing (SA) and related techniques have been applied to the mapping problem by several groups [5, 6].

SA can create good mappings but is computationally expensive [6]. Mean field annealing is an algorithm with similarities to SA. It is applied to problems with up to 400 tasks and 32 processors in hypercube and mesh topologies. It is compared with SA in [6].

Other work has been limited to problems displaying certain communication patterns. For instance, an algorithm was developed for mapping problems with a rectilinear topology [7]. This is extended in [8] for problems with a k-ary n-cube work and communication pattern.

Our work has similarities to that described above; however, there are important differences. We are primarily concerned with mapping N tasks to N processors, which is a more constrained problem than mapping $M \gg N$ tasks to N processors. BG/L is far larger than the target architectures of previous research; scaling the mapping to thousands of nodes is essential. Furthermore, the 3D torus of BG/L adds complexity to the mapping problem.

Much attention has been paid to achieving partitions of tasks that balance computational load and minimize interpartition communication. Far less attention has been spent on placing those partitions on processors linked by a network, such as a torus or mesh, in which the communication costs between different processor pairs vary considerably. We seek to redress that imbalance in this paper. This is especially important for a computer such as BG/L, since the cost differential between a good and a bad mapping in a torus increases with processor count.

Measure of communication time and layout quality

Consider a single communication event within an application in which one MPI task sends a message to another using nonblocking MPI functions. If the `MPI_IRECV` is posted at time t_r and the corresponding `MPI_ISEND` is posted at time t_s, the receiver receives the message at a time $t = \max(t_r, t_s + L + S/B)$, where L is the latency, S is the message size, and B is the bandwidth. The latency and bandwidth are not just simple numbers. In the runtime environment of a real application, they

depend on many parameters, such as the location of the sender and receiver on the network, the availability of buffers, and network congestion from other messages. This makes it impossible to write a simple function that measures the total message transit time for an application. We must make simplifications from general considerations.

The cost function, which is a measure of communication time, must reflect at least the following properties:

- An increase in the length of a path increases communication latency and should increase the cost function. Each time a packet is passed from one node to another, it must be routed. This introduces extra latency.
- If a message can access multiple paths, the bandwidth increases and the cost function should decrease. In BG/L, each node is connected to six torus links, and data is constrained to move toward its destination. If two nodes are on a line in one of the three planes, there is only one available shortest path between them. If the two nodes are on opposite corners of a rectangle or cuboid, there are many available paths. The total bandwidth is limited by the number of links into or out of the node that can actually be used by a communication primitive. This is illustrated in **Figure 1**. Thus, the maximum available bandwidth for a communication is one, two, or three times single-link bandwidth. Increasing the number of torus dimensions that links use increases the available bandwidth for communications.
- Increasing path length increases the probability of link contention.

We claim that a cost function that attempts to reduce the latency and contention will reduce communication time, even though it cannot predict communication times exactly. This claim is backed by results from experiments on BG/L hardware, presented in the results and discussion section.

To this end, we propose a general measure of quality of the form

$$\sum_{i,j} C(i, j)d[\pi(i), \pi(j)],$$

where task i exchanges $C(i, j)$ bytes with task j, tasks i, j are placed on nodes $\pi(i)$, $\pi(j)$, and nodes a and b are a distance $d(a, b)$ apart. Our definition of distance may depend on the characteristics of our application. For instance, an application with many small communications—where latency, rather than bandwidth, is most important—might use hop count $h(a, b)$ as the distance metric. For an application in which

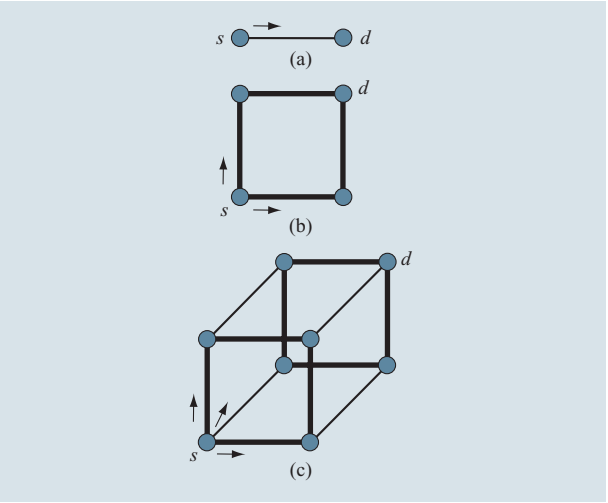

Figure 1

(a) If a path uses only one dimension of the torus, the available bandwidth is, at most, the bandwidth of a single link. If two or three dimensions are used, as in (b) and (c), two or three times link bandwidth is available. However, the extra hops may increase latency.

communication consists of infrequent, large, and non-overlapping messages, one might wish to maximize available bandwidth and use a distance metric such as

$$d(a, b) = \frac{3}{D}h(a, b),$$

where D is the number of dimensions used in the shortest path from a to b. Another possible measure, which combines aspects of the other two, is the Euclidean distance metric

$$d(a, b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2}.$$

By default, we use the hop-count distance metric.

The method

Assume that we are given a problem with D domains that are to be mapped to a BG/L computer with $L \times M \times N$ nodes. Let $C(i, j)$ be the amount of data communicated from domain i to domain j, and let $H(i, j)$ be the cost per unit of data for communication between domain i and domain j. $C(i, j)$ is fixed by the underlying application and remains unchanged and constant throughout the optimization process. $H(i, j)$, on the other hand, is directly dependent on the specific mapping of domains to nodes at any given step in the optimization and changes constantly, depending on link utilization. For simplicity, we assume that $H(i, j)$ depends only on the distance between the nodes to which domains i and j are mapped

and is independent of the actual amount of data to be transmitted.

On BG/L, nodes are arranged in a 3D lattice and connect only to their six nearest neighbors through the torus interconnect. Each node in the lattice is labeled by its (integer) Cartesian coordinates (x, y, z) within the lattice. On this interconnected lattice, the simplest choice for the $H(i, j)$ is to set it equal to the number of hops in the minimum path on the torus connecting node i to node j.

With this definition, the cost function we propose to minimize is given by

$$F = \sum_{i,j} C(i, j)H(i, j). \tag{1}$$

In the following, we adopt the terminology of simulated annealing and call F the *free energy*. For numerical work, it is helpful to normalize the free energy so that it is constrained to a fixed range. On the 3D BG/L torus, each node has six nearest neighbors (distance one hop), 18 next-nearest neighbors (distance two hops), etc. The free energy F is a sum over all domains i of a term $\Sigma_j C(i, j)H(i, j)$. The minimum achievable value for each of these individual terms is obtained if the domains j are "dealt out" in order of decreasing value of $C(i, j)$ first to the same node as domain i, then to the nearest neighbors of i, then to the next-to-nearest neighbors of i, etc. The minimum achievable value for the total free energy is then given by the sum over domains of each of these individual domain minimum values. We denote this minimum free energy by $F_{\min}$. In most cases (and particularly for nontrivial patterns), it is not possible to achieve this minimum value. Nevertheless, it is a useful concept for normalization and as a "theoretical" lower bound.

The general optimization method we propose is simulated annealing (SA) [9]. SA is an iterative method that repeatedly attempts to improve a given configuration by making random changes. To seed the process, an initial configuration must be set up. We have adopted the following method, which we have found to give good starting configurations, but there are many other suitable approaches possible. Define the function $c(i)$ as

$$c(i) = \sum_j C(i, j), \tag{2}$$

where $c(i)$ is the total data communicated by domain i. By a simple reordering, we arrange that the domains are labeled such that $c(1) \le c(2) \le c(3) \cdots$. The processors of BG/L are determined by their Cartesian coordinates (x, y, z). Let $p = D/(LMN)$ be the number of domains mapped to each BG/L node on average. Let $n(x, y, z)$ be the number of domains mapped to node (x, y, z). For compute load balancing, we restrict n to be close to p. The tightest constraint is to require that

$$\text{Int}(p) \le n(x, y, z) \le \text{Int}(p + 1), \tag{3}$$

where $\text{Int}(p)$ is the integer part of p.

An initial heuristic map is defined by the following algorithm:

1. Map domain $i = 1$ to an arbitrary location (x, y, z).
2. Map all domains with which domain $i = 1$ communicates either to location (x, y, z) or to its neighboring locations on the BG/L torus while satisfying the constraint of Equation (3).
3. Next, map domain $i = 2$ (if it is not yet mapped) to an arbitrary location (x', y', z') and the unmapped domains with which it communicates either to the same node or to a neighboring node on the torus while satisfying the constraint of Equation (3).
4. Repeat this last step for $i = 3, 4, \cdots, D$.

This heuristic can be made more sophisticated by first mapping those domains with the greatest communication volume to already-mapped domains, rather than simply mapping domains 2, 3, 4, $\cdots$ in order.

Starting from this heuristic map, we do SA to find a minimum of the free energy F. To this end, define a Markov chain [10] of mappings $\{M_i, i = 0, 1, 2, \cdots, n\}$, where M_0 is the heuristic map and M_i is derived from M_{i-1} by applying the following algorithm (called the Metropolis algorithm [11]):

1. Introduce a new parameter β, which we interpret at inverse temperature ($\beta = 1/kT$), and initially set β to some small value.
2. Generate a candidate new map M_i' from M_{i-1} by either swapping two domains between randomly chosen locations on the torus or by moving a domain from a random location on the torus to another, provided this does not violate Equation (3).
3. Accept M_i', i.e., set $M_i = M_i'$ iff

   $$R < e^{-\beta\{F(M')/F_{\min} - F(M_{i-1})/F_{\min}}, \tag{4}$$

   where R is a random number uniformly distributed in (0, 1).
4. Continue to generate new candidate mappings using the same β value until the free energy decreases to a stable equilibrium.
5. Now, in steps, increase β and continue the Markov chain process until, for each new β, a new free energy equilibrium is reached. The initial β is chosen so that the acceptance rate is between 10% and 20%. The final β is chosen so that the acceptance rate is approximately zero. We chose to take 100 steps between the initial and final values of β.

This procedure is called *simulated annealing* because it is analogous to the annealing process in metallurgy, where metals are first heated and then slowly cooled to remove impurities. Note also the normalization used for the free energy factors into Equation (4). This normalization is helpful computationally.

Example application

As a simple example to test these methods, consider a problem that has 8^3 domains on a regular 3D torus, where each domain communicates a fixed amount of data to its six nearest neighbors. We attempt to map this problem onto an $8 \times 8 \times 8$ BG/L torus. The problem is small and simple, and it has an optimum map that is obvious; however, it is complex and large enough to be a useful test of the methods we propose here.

Figure 2 shows the structure of the communications matrix $C(i, j)$ for this problem. Nonzero entries in this matrix are shown as crosses. The main adjustable parameters in the method are the values of β that are used and the number of iterations that are executed for any given value of β.

For the method to be successful, it is important to ensure that the acceptance rate for moves is not too low. A low acceptance rate signals that the mapping is not changing from one step to the next. A reasonable range for the acceptance rate is 10–50%. We recall from Equation (4) that the acceptance rate is governed by the condition that a random number is less than the exponential of the change in the normalized free energy $(F = F_{\min}) \cdot \beta$.

By construction, $F/F_{\min} \leq 1$, but for a good mapping, we expect $F/F_{\min}$ to be close to 1 in value. Typical starting values for β are then in the range $0 \leq \beta \leq 10$. With these ranges, the argument of the exponential is typically less than $O(10)$, and reasonable acceptance rates can be achieved for the Metropolis step.

To determine the number of iterations to execute for a given value of β, it is sufficient to monitor the behavior of the free energy function as the annealing proceeds. Typically, this drops quickly for a time and then reaches a plateau. The plateau signals the onset of equilibrium for the given value of β, and it is usual to switch to a new β once the plateau has been clearly established for a given β.

Figure 3(a) shows the evolution of the normalized free energy for an example run of the SA algorithm. **Figure 3(b)** plots the values of normalized free energy achieved at the tail end of each fixed β segment for the same run. Observe in both figures that the value of the normalized free energy falls steadily and eventually achieves the ideal value 1. Inspection of the mapping of domains to nodes for this final achieved configuration shows that it corresponds to the ideal in which domains are mapped to the torus in a manner in which only nearest-neighbor communication is required.

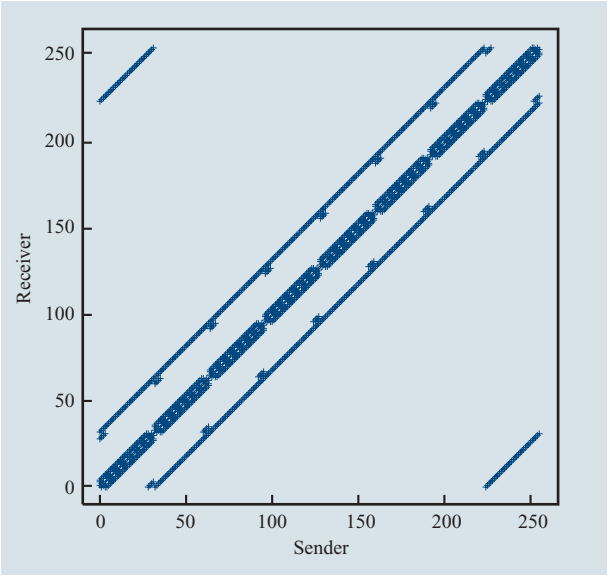

Figure 2

Structure of the matrix $C(i, j)$ for the nearest-neighbor communications problem described in the text. The horizontal axis specifies the identity of the sender, i. The vertical axis specifies the identity of the receiver, j. A cross at a point (i, j) represents a communication between MPI task number i and MPI task number j. Notice the regular banded structure, which is prototypical of nearest-neighbor communications.

An enhancement

The previous section has demonstrated that, at least in simple cases, SA can be used to find a good mapping of problem domains to BG/L compute nodes. However, the simple implementation described above does not scale well enough to be used on a full-size BG/L system, which could contain up to 65,536 compute nodes. Compute times for the simple implementation are already hours long for optimizing layouts on $O(10^3)$ processors and would become prohibitive for the full BG/L machine.

Happily, however, the annealing method is very robust and admits almost infinite variation. In this section we describe a divide-and-conquer approach that we have implemented which allows a fast generation of slightly suboptimal mappings. This approach can be used to generate very good mappings. In our examples, the final maps are quite close to the optimum map found by full SA and much better than the default MPI rank order mapping.

For many parallel applications, the communication matrix $C(i, j)$ is sparse. In many problems, domains communicate mostly with nearby domains and relatively

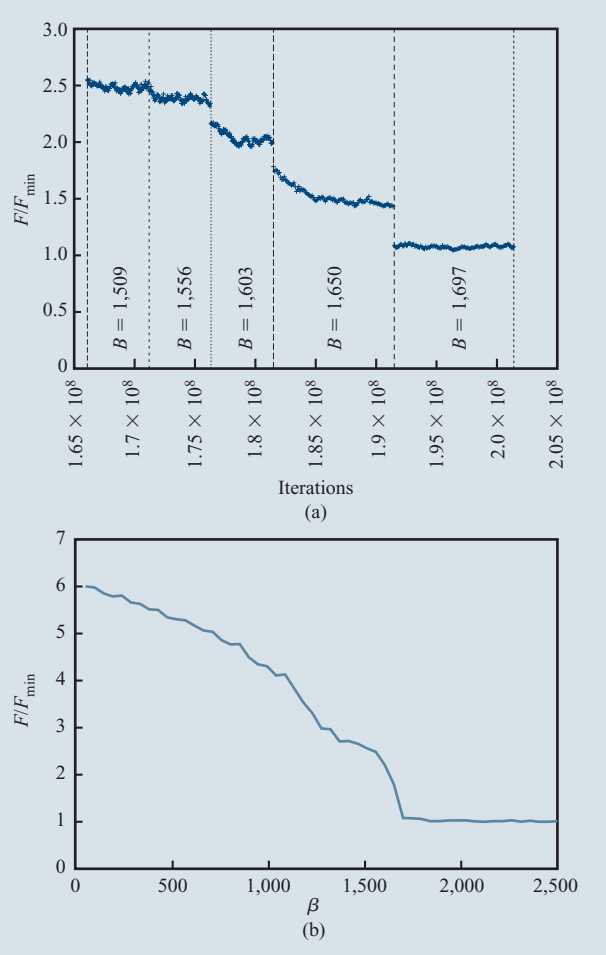

Figure 3

(a) Normalized free energy as a function of iteration. The vertical dividers on this plot define the points at which β is increased. Within each fixed β band, the free energy initially falls and then plateaus. As one moves left to right between bands, β increases and the free energy equilibrium or plateau value decreases. Early in the simulation, convergence is slow, and plateaus are not really obvious. For the last β change shown, convergence occurs very quickly. This type of convergence profile is quite common in annealing. (b) The converged plateau value of the free energy achieved as a function of β. For this problem, the free energy converges to the ideal minimum value of 1 as β increases beyond a certain value. The resulting layout is the optimal mapping of domains to the BG/L torus for a 3D nearest-neighbor problem.

infrequently with distant domains. This suggests that the map of problem domains to nodes might be constructed one domain at a time. The advantages of such an approach are clear. For an n-node system, the possible mappings that have to be searched for an optimum layout are $O(n!)$. It is clearly more efficient to optimize k maps of size $n = k$ than one map of size n. Of course, in general, the free energy minimum that results from a divide-and-conquer approach is not optimum because the search space over which the optimization is done is truncated.

The divide-and-conquer approach we have implemented proceeds as follows. First, a graph $G(V, E)$ is constructed in which each vertex $v \in V$ represents a problem domain. Two vertices i, j are joined by an edge $e_{ij} \in E$ if $C(i, j) > 0$. Each edge is assigned a weight $C(i, j)$. A subgraph $S(V_s, E_s)$ of G consists of a subset V_s of V and a subset E_s of E where $\forall e_{ij} \in E_s$, $i, j \in V_s$.

The communications graph G is now divided into subgraphs to minimize communications between subgraphs. We have chosen to use a public domain package called METIS [12] for this partitioning. METIS is the standard tool in graph partitioning. It attempts to produce balanced partitions that minimize the edge cut of the partitions. The *edge cut* is the sum of the weights of all edges that link vertices belonging to different partitions. This partitioning scheme allows us to identify groups of problem domains that are maximally independent from one another and so can be independently assigned to compute nodes with minimal interference.

Having obtained a METIS partitioning, we proceed to sequentially map one subgraph at a time to a set of nodes using the heuristic method defined in the method section above; we then use SA to optimize the subgraph maps one by one. At any given step, annealing is applied only to the subgraph being mapped. Those subgraphs that were previously mapped are left unchanged. The first subgraph to be mapped to the compute nodes is selected at random. Thereafter, the next subgraph to be mapped is determined by choosing the subgraph with the largest connectivity to subgraphs already mapped.

The method is presented in detail in **Figure 4**.

An important aspect of the process is the decision on the size of the individual partitions. Small partitions anneal quickly. However, large partitions are required to achieve optimum or close to optimum mappings. In a subgraph $S(V_s, E_s)$ of G, the ratio

$$l = \frac{\sum_{i,j \in V_s} C(i, j)}{\sum_{i \in V_s, j \in V} C(i, j)} \tag{5}$$

describes how connected the subgraph S is to the rest of the communications graph G. Constraining l would be the method normally used to control quality in a production implementation of annealing for mapping optimization. For this paper, however, we have not adopted this approach. Instead, we show results for various partition sizes. On an IBM pSeries* 1,200-MHz POWER4* processor, the typical execution time for optimization using small partitions (e.g., eight nodes per partition, 128 partitions total) is 30 minutes. Typical execution time for

optimization using large partitions (e.g., 256 nodes per partition, four partitions total) is ten hours.

Results and discussion

As a test of these methods, we consider four separate communication problems.

- *Cubic 1:* This is the 3D nearest-neighbor problem already discussed in the example application section above. There is one domain for each node on an $8 \times 8 \times 8$ torus, and the task is to lay them out on an $8 \times 8 \times 8$ BG/L torus. The communications pattern is that each domain sends equal-sized packets to each of its six nearest neighbors on the torus.
- *Cubic 2:* This problem is similar to Cubic 1, but instead domains send full-sized packets to their six nearest neighbors (distance one hop) and half-sized packets to their six next-nearest neighbors (distance two hops) along the Cartesian directions of the torus. Packets are sent in straight lines only; no turns on the grid are included.
- *SAGE:* SAGE, from Science Applications International Corporation (SAIC), stands for the SAIC adaptive grid Eulerian hydrodynamics application [13]. In the example used here, we consider only a simple case with the `timing_h` input set, which does heat conduction and hydrodynamics without adaptive mesh refinement. The communications pattern is complex enough that this case is useful by itself. The requirements of the full code with adaptive mesh refinement (AMR) would be similar, requiring us to find a mapping to BG/L every time the problem grid is redefined. The main points of the analysis are the same in the full code, and our methods would also apply in this case.

  SAGE uses a regular Cartesian grid, in which cells are grouped into blocks, with each block containing eight cells in a $2 \times 2 \times 2$ configuration. Blocks are distributed in (x, y, z) order. For scaling purposes, we use a constant number of cells per task. With the `timing_h` input set, the global domain is a cube. For small task counts, the decomposition strategy results in something close to a slab decomposition of the global domain. The precise shape of the local domain depends on the number of cells per task and on the number of tasks.
- *UMT2000:* This is an ASCI Purple Benchmark, which solves a photon transport problem on an unstructured mesh [14]. The application is written in Fortran 90 using MPI and, optionally, OpenMP [15]. The unstructured mesh is statically partitioned in the code using the METIS library. Since there is a significant spread in the amount of computational work per task, this code can have a serious compute load imbalance. However, in this paper, we address only its communication imbalance, which is equally significant.

Input
T, the set of compute nodes in the BG/L torus network
The full unpartitioned graph $G(V, E)$
A subgraph $S(V_s, E_s)$ of G
$P \subset V$, where $p \in P$ are vertices already mapped to compute nodes in T
A mapping $M : P \leftarrow T$
L, a limit on subdivisions.

Output:
Updates M to include mappings from V_s to T. Updates P to include V_s
(1) Use METIS to divide S into $S_1(V_1, E_1)$ and $S_2(V_2, E_2)$
(2) $l_1 \leftarrow \frac{\Sigma_{i,j \in V_1} C(i,j)}{\Sigma_{i \in V_1, j \in V} C(i,j)}$, $l_2 \leftarrow \frac{\Sigma_{i,j \in V_2} C(i,j)}{\Sigma_{i \in V_2, j \in V} C(i,j)}$
(3) **if** $l_1 < L$ or $l_2 < L$
(4) Assign each $v \in V_s$ to a node $t \in T$ using simulated annealing
(5) Update P: $P \leftarrow P \cup V_s$
(6) Update M to include the mappings from V_s to T
(7) **else**
(8) $c_1 \leftarrow \Sigma_{i \in V_1, j \in P} C(i,j)$
(9) $c_2 \leftarrow \Sigma_{i \in V_2, j \in P} C(i,j)$
(10) **if** $c_1 > c_2$
(11) Partition(T, G, S_1, P, M, L)
(12) Partition(T, G, S_2, P, M, L)
(13) **else**
(14) Partition(T, G, S_2, P, M, L)
(15) Partition(T, G, S_1, P, M, L)
(16) **return** M

Figure 4

The partition algorithm recursively bisects a graph until the subgraphs are small enough to be mapped to the torus using simulated annealing.

The communications patterns for the Cubic 2, SAGE, and UMT2000 are shown in **Figures 5(a)–5(c)**. The Cubic and SAGE cases have regular communications patterns, while UMT2000 has an irregular pattern.

Tables 1, 2, and **3** give the optimization results for various grid sizes and SA variants for each of the test cases studied. In these tables, *ideal* denotes the ideal "theoretical" value for the free energy discussed before. *Heuristic* is the value achieved when the heuristic placement algorithm in the section above on method is used. In the tables, *MPI rank order* is the value achieved when domains are placed in MPI rank order. We define *random* as the average value achieved for 100 random placements of domains onto the BG/L torus. The *ideal*

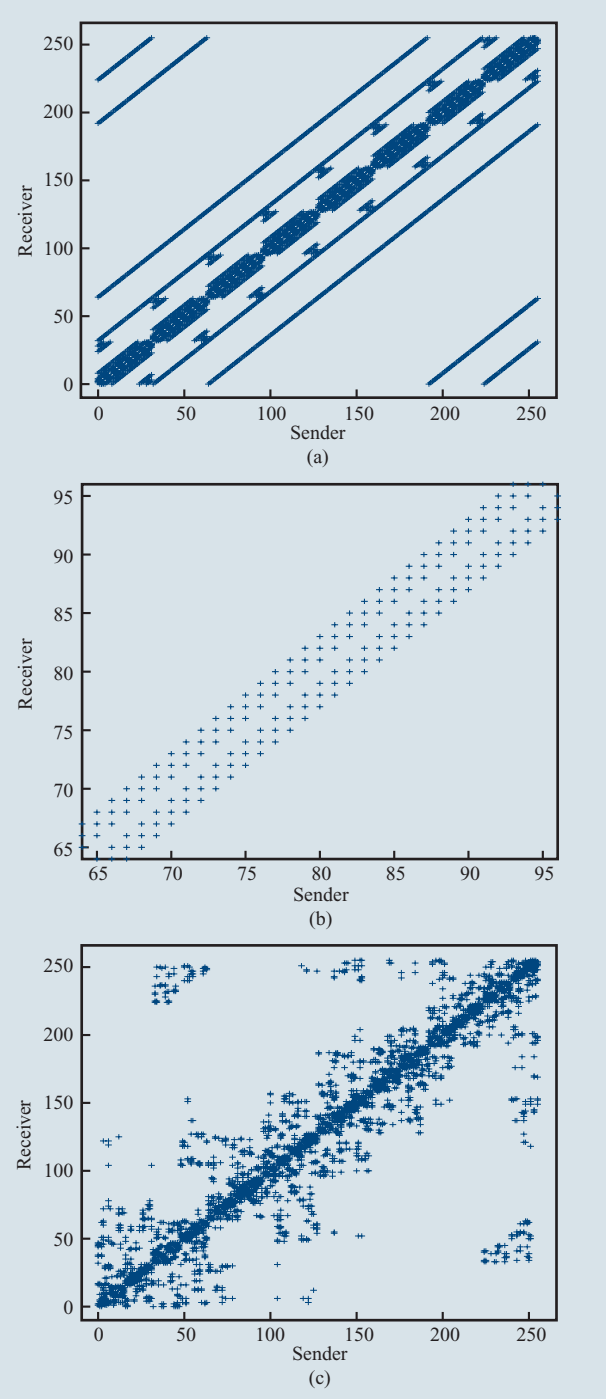

Figure 5

(a) Communications pattern for the Cubic 2 problem on an $8 \times 8 \times 8$ torus. For all three plots, a cross at a point (i, j) represents a communication between MPI task number i and MPI task number j. (b) Communications pattern for SAGE on an $8 \times 8 \times 8$ torus. SAGE has an extremely regular communications pattern where nodes communicate with some of their MPI rank order neighbors. We show only a subset of the full pattern here. (c) Communications pattern for UMT2000 on an $8 \times 8 \times 8$ grid.

Table 1 Optimization results achieved for the Cubic 1 and Cubic 2 cases. These are 3D nearest- and next-to-nearest-neighbor mapping problems, chosen because the ideal mapping is obvious and easy to identify. The ideal mapping case has a normalized free energy of 1. The heuristic, MPI rank order, and random cases provide reference values for the free energy values, which can be achieved by the various placement strategies described in the paper. For these problems, we attempted annealing only with a single partition (SA 1) containing all of the nodes, which generated the optimum layout with normalized free energy equal to 1 (exact solution).

Method	*Cost, F*	*Normalized cost,* $F/F_{\min}$
Cubic 1, 512 nodes		
Ideal	3,072	1.0
Heuristic	7,360	2.4
MPI rank order	3,072	1.0
Random	18,245	5.94
SA 1	3,072	1.0
Cubic 2, 512 nodes		
Ideal	12,288	1.0
Heuristic	25,956	2.11
MPI rank order	12,288	1.0
Random	54,820	4.56
SA 1	12,288	1.0

Table 2 Results achieved for UMT2000. The improvement over the default mapping for MPI is a factor of 1.68 and 1.65 for 256 and 1,024 tasks, respectively.

Method	*Cost, F*	*Normalized cost,* $F/F_{\min}$
UMT2000, 256 nodes		
Ideal	193,516	1
Heuristic	389,722	2.01
MPI rank order	420,990	2.18
Random	776,693	4.01
SA 7	264,394	1.37
SA 4	282,490	1.46
SA 2	267,286	1.38
SA 1	252,398	1.30
UMT2000, 1,024 nodes		
Ideal	782,708	1
Heuristic	1,714,512	2.19
MPI rank order	2,002,734	2.56
Random	4,967,656	6.35
SA 32	1,479,240	1.89
SA 18	1,486,504	1.90
SA 14	1,451,362	1.85
SA 8	1,309,312	1.67
SA 4	1,280,488	1.64
SA 2	1,252,798	1.60
SA 1	1,217,352	1.55

Table 3 Results achieved for SAGE. The improvement over default MPI layout is a factor of 1.52 and 2.45 for 256 and 2,048 MPI tasks, respectively.

Method	*Cost, F*	*Normalized cost,* F/F_{min}
SAGE, 256 nodes		
Ideal	169,152	1
Heuristic	289,892	1.71
MPI rank order	366,194	2.16
Random	826,264	4.88
SA 32	303,184	1.79
SA 16	264,170	1.56
SA 8	269,446	1.59
SA 4	256,714	1.52
SA 2	248,712	1.47
SA 1	239,392	1.42
SAGE, 2,048 nodes		
Ideal	730,958	1
Heuristic	1,019,078	1.39
MPI rank order	2,383,612	3.26
Random	7,257,358	9.93
SA 128	1,110,450	1.52
SA 64	1,105,844	1.51
SA 34	1,131,802	1.55
SA 32	1,127,680	1.54
SA 16	1,066,160	1.46
SA 4	988,772	1.35
SA 1	975,400	1.33

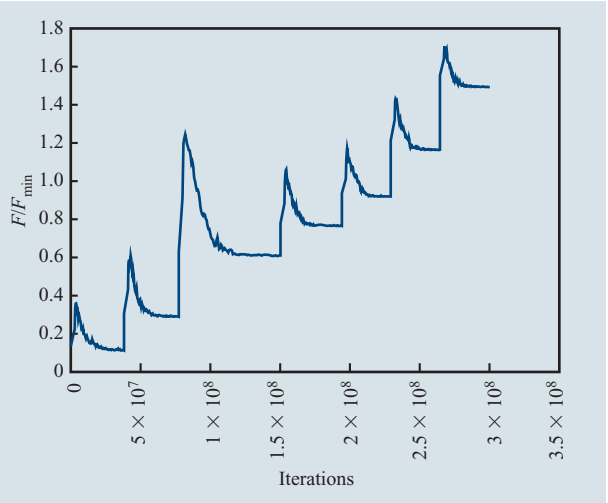

Figure 6

Convergence curve for the UMT2000 problem using the divide-and-conquer approach. The vertical jumps locate the positions in the simulation at which another subgraph is added to the problem. At a subgraph addition, the temperature is increased; then the added subgraph is optimized by annealing. The typical annealing profile of initial fast decrease, then plateau, is observed. Notice that the free energy values achieved for the plateaus increase from left to right. This is because, at each step, an additional subgraph has been mapped, and the free energy has more terms contributing. The number of steps equals the number of subgraphs used.

and *random* cases respectively provide lower and upper bounds for the normalized free energy. Finally, the SA *n* entries provide the results achieved for the divide-and-conquer variant in which the domains are divided into *n* subdomains using METIS, and placement of each of these subdomains is successively optimized until the full problem has been mapped. In this variant, the SA 1 case represents a full SA run with one partition containing all nodes.

Figure 6 shows a typical convergence curve generated by the divide-and-conquer approach. Table 1 shows that the method succeeds with the Cubic 1 and Cubic 2 maps where, as expected, it finds the known optimal map. For UMT2000 (Table 2), the improvement in the cost function over the default map (MPI rank order) is by factors of 1.68 and 1.65 for 256 and 1,024 tasks, respectively. For SAGE (Table 3), the improvement over the default MPI map is by factors of 1.52 and 2.45 for 256 and 2,048 tasks, respectively. These are quite significant effects and lead to improved communication performance, as we demonstrate in the next section. We are confident that the improvements for larger task counts will be equal or better.

Thus far we have discussed the improvement in the free-energy value. We now present results showing the variation in measured communication times when different mappings are used. An application was run on the BG/L hardware using a variety of mappings. The runtime for each mapping was recorded, and, more significantly, the time the application spent in MPI point-to-point communications was measured. We chose SAGE on a 512-node partition as the most appropriate application, since it is more communication-bound than UMT2000. The mappings were created using the methods discussed above. In addition, the best known handcrafted map was used. We use the costs and times associated with MPI rank order as normalization constants.

Figure 7 plots the improvement factor in cost function and communication time for the mappings. **Table 4** shows the costs and communication times relative to the default MPI rank order map; the results are interesting. The best mapping, created by a single-partition SA run, shows a factor of 1.79 improvement over the default MPI rank order map. The heuristic method and the four-partition SA run also show improvements by a factor of more than 1.7. In addition, all of the SA runs and the heuristic method generate maps with lower communication times than the handcrafted map. However, it is necessary to note that the handcrafted map actually has the lowest free

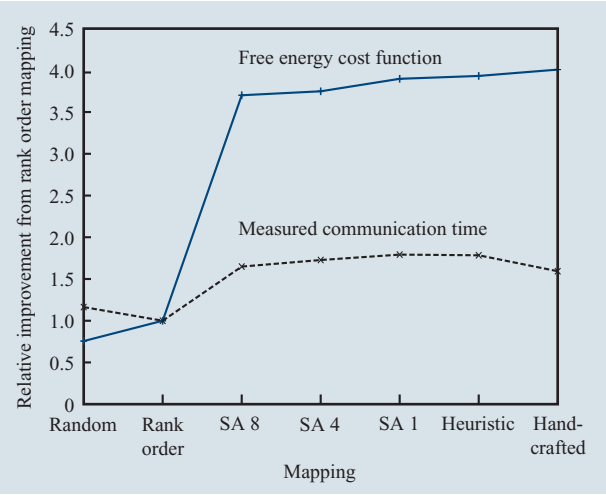

Figure 7

Improvement factor in cost function and communication time over the default MPI rank order mapping. The handcrafted map is described in the text and is available in this case only for machine sizes that are a multiple of 2^3, such as 8, 64, 512, etc.

energy, and in addition, the relationship between free energy and communication time is not linear. This is not surprising, since the free energy cost function does not model all aspects of the torus network. The premise upon which the cost function was chosen is that by locating those domains that communicate with one another close together, the total communication time should be reduced. The cost function does not fully model the communications patterns or network behavior (this would be prohibitively expensive), and so there is not an exact match between reduced free energy cost and reduced communication time. This is particularly noticeable when comparing the random and the MPI rank order layouts, which, for the SAGE application, are two extremes.

Summary and future prospects

In this paper we have presented several methods for generating good mappings. The choice of method to be used depends on the application. For a small application that runs for a short time, our simple heuristic method may be best. For larger applications, which may run for extended periods of time, the extra time required to generate a potentially better map using SA might be justified by the reduction in communication time. The communication characteristics of an application may themselves determine which mapping method works best. For instance, the simple heuristic method is sufficient for the SAGE application, while the SA method gives much better results for UMT2000 (Table 2).

For the cases presented in this paper, we found that the SA method produces good mappings and has the potential to be used as a general layout-optimization tool for parallel codes. Since the current serial code we have implemented to test the method is unoptimized, computation time to find the optimum map can be large (several hours on a typical personal computer.) However, the SA algorithm can easily be parallelized. For production implementation, good parallel code for our algorithm would be required and could itself be implemented on BG/L.

If one or more of the methods described here is practically useful (leads to performance gains across a spectrum of real applications), it could be incorporated into the runtime system of the BG/L computer, especially for applications that are expected to run repeatedly for the same problem sizes after an optimum layout is discovered. This can easily be done for the heuristic algorithm with little impact on application performance, and it might even be feasible with a parallelized

Table 4 Communication time results for a 512-node processor SAGE run using several different maps. The cost and time of the MPI rank order map are used respectively to normalize the costs and communication times.

Method	*Normalized cost*	*Improvement in cost*	*Communication time*	*Improvement in communication time*
MPI rank order	1	1	22.818	1
Random	1.322	0.76	19.614	1.163
Handcrafted	0.249	4.01	14.327	1.593
SA 8	0.270	3.70	13.837	1.649
SA 4	0.267	3.75	13.207	1.728
Heuristic	0.254	3.93	12.790	1.784
SA 1	0.257	3.90	12.731	1.792

implementation of the SA methods. It could also find use for AMR codes in which the mesh is dynamically redefined during the computation. Indeed, SAGE uses AMR, and the heuristic described here should be very effective for this application.

Acknowledgments

The Blue Gene/L project has been supported and partially funded by the Lawrence Livermore National Laboratory on behalf of the United States Department of Energy under Lawrence Livermore National Laboratory Subcontract No. B517552.

* Trademark or registered trademark of International Business Machines Corporation.

References

1. G. Bhanot, D. Chen, A. Gara, and P. Vranas, "The Blue Gene/L Supercomputer," *Nucl. Phys. B (Proc. Suppl.)* **119**, 114–121 (2003).
2. V. A. Dixit-Radiya and D. K. Panda, "Task Assignment on Distributed-Memory Systems with Adaptive Wormhole Routing," *Proceedings of the Symposium on Parallel and Distributed Processing (SPDP)*, 1993, pp. 674–681.
3. A. Billionnet, M. C. Costa, and A. Sutter, "An Efficient Algorithm for a Task Allocation Problem," *J. ACM* **39**, No. 3, 502–518 (July 1992).
4. J. L. Träff, "Implementing the MPI Process Topology Mechanism," *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2002, pp. 1–14; see *http://sc-2002.org/paperpdfs/pap.pap122.pdf*.
5. F. Ercal, J. Ramanujam, and P. Sadayappan, "Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning," *J. Parallel & Distr. Computing* **10**, No. 1, 35–44 (September 1990).
6. T. Bultan and C. Aykanat, "A New Mapping Heuristic Based on Mean Field Annealing," *J. Parallel & Distr. Computing* **16**, No. 4, 292–305 (December 1992).
7. D. Nicol, "Rectilinear Partitioning of Irregular Data Parallel Computations," *J. Parallel & Distr. Computing* **23**, No. 2, 119–134 (November 1994).
8. D. Nicol and W. Mao, "On Bottleneck Partitioning of *k*-ary *n*-cubes," *Parallel Process. Lett.* **6**, No. 6, 389–399 (June 1996).
9. P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Co., Dordrecht, Netherlands, 1987; ISBN: 9027725136.
10. P. Brémaud, *Markov Chains: Gibbs Fields, Monte Carlo Stimulation, and Queues*, Springer-Verlag, New York, 1999; ISBN: 0387985093.
11. *Markov Chain Monte Carlo in Practice*, W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, Eds., Chapman and Hall/CRC, Boca Raton, FL, 1996; ISBN: 0412055511. I. Manno, *Introduction to the Monte-Carlo Method*, Akademiai Kiado, Budapest, Hungary, 1999; ISBN: 9630576155.
12. See *http://www-users.cs.umn.edu/~karypis/metis*.
13. D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive Performance and Scalability Modeling of a Large-Scale Application," *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2001, pp. 37–37; see *http://www.sc2001.org/papers/pap.pap255.pdf*.
14. ASCI Purple Benchmark Web page; see *http://www.llnl.gov/asci/purple/benchmarks/*.
15. See *http://www.openmp.org*.

Received July 20, 2004; accepted for publication August 20, 2004; Internet publication April 6, 2005

Gyan Bhanot *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (gyan@us.ibm.com)*. Dr. Bhanot is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received his Ph.D. degree in theoretical physics from Cornell University. He works on bioinformatics, systems biology, and parallel computation.

Alan Gara *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (alangara@us.ibm.com)*. Dr. Gara is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received his Ph.D. degree in physics from the University of Wisconsin at Madison in 1986. In 1998 Dr. Gara received the Gordon Bell Award for the QCDSP supercomputer in the most cost-effective category. He is the chief architect of the Blue Gene/L supercomputer. Dr. Gara also led the design and verification of the Blue Gene/L compute ASIC as well as the bring-up of the Blue Gene/L prototype system.

Philip Heidelberger *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (philiph@us.ibm.com)*. Dr. Heidelberger received his B.A. degree in mathematics from Oberlin College in 1974 and his Ph.D. degree in operations research from Stanford University in 1978. He has been a Research Staff Member at the IBM Thomas J. Watson Research Center since 1978. His research interests include modeling and analysis of computer performance, probabilistic aspects of discrete event simulations, parallel simulation, and parallel computer architectures. He has authored more than 100 papers in these areas. Dr. Heidelberger has served as Editor-in-Chief of the *ACM Transactions on Modeling and Computer Simulation*. He was the general chairman of the ACM Special Interest Group on Measurement and Evaluation (SIGMETRICS) Performance 2001 Conference, the program co-chairman of the ACM SIGMETRICS Performance 1992 Conference, and the program chairman of the 1989 Winter Simulation Conference. Dr. Heidelberger is currently the vice president of ACM SIGMETRICS; he is a Fellow of the ACM and the IEEE.

Eoin Lawless *Trinity Centre for High Performance Computing, O'Reilly Institute, Trinity College, Dublin 2, Ireland (eoin@maths.tcd.ie)*. Dr. Lawless is a Research Intern at the IBM Thomas J. Watson Research Center. He received his Ph.D. degree in parallel network simulation from Trinity College, Dublin, and works in network simulation and parallel computation.

James C. Sexton *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (sextonjc@us.ibm.com)*. Dr. Sexton is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received his Ph.D. degree in theoretical physics from Columbia University and has held research positions at the Fermi National Accelerator Laboratory (Fermilab), the Institute of Advanced Studies at Princeton University, and Trinity College, Dublin.

Robert Walkup *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (walkup@us.ibm.com)*. Dr. Walkup is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received his Ph.D. degree in physics from the Massachusetts Institute of Technology. He currently works on high-performance computing algorithms.