Mapping Strategies for Switch-Based Cluster Systems of Irregular Topology*

Sangman Moh[†¶], *Chansu Yu*[†], *Hee Yong Youn*[‡], *Ben Lee*[§], *and Dongsoo Han*[†]

[†]School of Engineering, Information and Communications Univ.
58-4 Hwa-am, Yu-sung, Taejon, 305-348 KOREA
{smmoh,cyu,dshan}@icu.ac.kr

[‡]Dept. of Electrical and Computer Eng., Sungkyunkwan Univ., Suwon, KOREA
youn@ece.skku.ac.kr

[§]Dept. of Electrical and Computer Eng., Oregon State Univ., Corvallis, OR 97331
benl@ece.orst.edu

[¶]Electronics and Telecommunications Research Institute, Taejon, KOREA
smmoh@etri.re.kr

Abstract

Mapping virtual process topology to physical processor topology is one of the most important issues in parallel computing. The mapping problem for switch-based cluster systems of irregular topology is very complicated due to the connection irregularity and routing complexity. This paper proposes two mapping schemes for irregular cluster systems, which try to map the nearest neighbors in the process topology to physically adjacent processors. In addition, an application-oriented performance metric, weighted cardinality, is introduced to represent the quality of mapping. Simulation study shows that, for a virtual topology of a 16×16 *mesh, the proposed mapping schemes result in better mapping quality and about* $15 \sim 20\%$ *shorter communication latency compared to random mapping. The proposed algorithms should also be beneficial when they are applied to metacomputing and cluster of cluster systems, where the communication costs are an order of magnitude different depending on the relative position of the processor nodes.*

Index terms: Cluster system, mapping strategy, virtual topology, irregular network, MPI.

1 Introduction

Domain decomposition is one of the most important issues in parallel computing. It involves dividing a problem domain into nearly equal-sized partitions and forming a computational graph, called a *process topology*. In a process topology, each vertex represents the required computation or process, and an edge represents the communication between two processes. To execute a parallel program, processes in its process topology are mapped onto processors, or *processor topology*. The mapping is relatively easy when both process and processor topologies are regular. Typically, the process topology is regular since many well-known computation-intensive problems are parallelized or decomposed into regular structures such as meshes or rings. The processor topology is also regular in traditional parallel machines that employ regular interconnection networks.

This paper discusses the mapping problem in switch-based cluster systems [1, 2], where the processor topology is irregular, *i.e.*, switches are interconnected in an arbitrary manner[1]. The primary advantages of the irregularity are higher interconnection flexibility and incremental system expandability, which are not attainable in traditional regular interconnection networks. However, one important disadvantage of the irregular topology is *routing complexity* to avoid *deadlock* among

*This research was supported in part by the Ministry of Information and Communication under Grant No. 99-159-01.

[1]Some cluster systems (*i.e.*, Beowulf and RWCP [1]) employ a completely connected graph topology with redundant paths to provide sufficient bisection bandwidth. Therefore, due to the limited number of ports as well as the severe cable length restriction, cascading of switches becomes imperative to build a large, scalable cluster [3].

multiple packets traveling simultaneously [4, 5, 6, 7]. Therefore, the mapping problem in switch-based cluster systems becomes very interesting and challenging.

In this paper, two new mapping schemes are proposed for irregular cluster systems assuming that the 2-D mesh process topology[2] is specified as an interprocess communication pattern. Key idea of the proposed schemes is to map the nearest neighbors in the process topology to physically adjacent processors, for example, those connected to the same switch. The two proposed mapping schemes are *switch-based mapping* (*SM*) and *binary mapping* (*BM*), which have time complexities of $O(n \log n)$ and $O(n^2 \log^2 n)$, respectively, where n is the number of processes of a 2-D mesh to be mapped onto the processor topology. For a process topology of a 16×16 mesh, our simulation study shows the proposed mapping schemes result in about $15 \sim 20\%$ shorter communication latency compared to *random mapping* (*RM*).

The major contributions of this paper are threefold: First, this paper introduces and formulates the mapping problem in the context of switch-based irregular networks. Second, a new variant of the processor topology, called *routing topology*, which properly reflects the deadlock-free routing requirement is introduced. Third, the concept of *weighted cardinality* is introduced as a metric for evaluating the quality of mapping.

The rest of the paper is organized as follows. The formal description of the mapping problem is presented in the following section. The mapping problem for irregular cluster systems is discussed in Section 3. In Section 4, the proposed mapping schemes and the complexity analysis in terms of cost-quality tradeoff are presented. The proposed schemes are evaluated using simulation in Section 5. The conclusion and future work are discussed in Section 6.

2 The Mapping Problem

Let the graph of the processor topology be denoted as $G_p(N_p, E_p)$, where N_p is the set of processors and E_p represents the set of interconnection links among the processors. Let the graph of the problem or *virtual topology*[3] to be mapped onto the processor topology be denoted as $G_v(N_v, E_v)$, where N_v corresponds to the

[2]One such example is `MPI_Cart_create()` in the MPI standard. See Subsection 3.2 for details.

[3]Throughout this paper, we use process topology and virtual topology interchangeably. The latter is used to contrast with the physical processor topology.

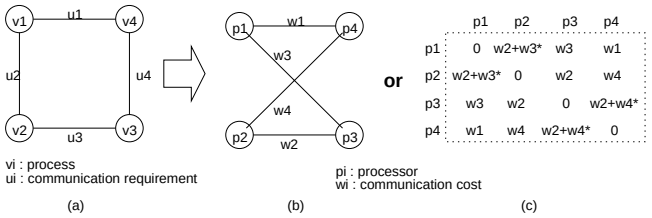

Figure 1: Mapping problem: (a) a virtual topology G_v, (b) a processor topology G_p, and (c) a cost matrix $\mathbf{M}_p$ of G_p ($^*F_p(p_1, p_2) = w_2 + w_3$ or $w_1 + w_4$, $F_p(p_3, p_4) = w_2 + w_4$ or $w_1 + w_3$).

set of processes or computations and E_v represents the set of communication paths among the processes.

Weights on edges in E_p and E_v represent the communication costs and the communication requirements, respectively. The two topologies can alternatively be defined by the functions $F_p : N_p \times N_p \rightarrow \mathbf{R}$ and $F_v : N_v \times N_v \rightarrow \mathbf{R}$. For example, $F_p(s, t) = w$ means that for a system architecture G_p, the cost to communicate between two processors s and t is w, where $s, t \in N_p$. Similarly, $F_v(x, y) = u$ denotes that for a problem graph G_v, processes x and y require u amount of communication to solve the problem. Figure 1 shows an example of the virtual and processor topologies as well as the $N_p \times N_p$ cost matrix, $\mathbf{M}_p$, which is another representation of F_p. For example, in Figure 1(b) and (c), the communication cost between p_1 and p_3 is w_3, while the cost between p_1 and p_2 is either $w_2 + w_3$ or $w_1 + w_4$ depending on the intermediate node(s) along the communication path.

Definition 1 *Given two weighted graphs $G_p(N_p, E_p)$ and $G_v(N_v, E_v)$, a mapping of G_v onto G_p is a function $f_m : N_v \rightarrow N_p$.*

The quality of mapping can be estimated by the overall communication cost. In the past, bisection bandwidth has been used to compare different interconnection networks [8]. However, bisection bandwidth is insufficient to represent the communication requirement of an application mapped onto a specific system architecture for two reasons. First, the bisection bandwidth only considers the interprocess communication of a parallel application on the links between two partitions. Second, bisection bandwidth is a machine-specific characteristic, and thus it does not properly represent the communication performance of a given application when running on a specific system. Therefore, a new metric, *weighted cardinality*, is introduced to represent the quality of mapping. The weighted cardinality is an application-oriented performance metric

which accounts for the bandwidth along all network links specified in the communication pattern of an application.

The weighted cardinality, $|f_m|$, can be obtained by adding the weights (*i.e.*, communication costs) on the corresponding edges in E_p for all edges in E_v. Note that the addition must be a weighted sum because the edges in E_v can have different communication requirements. This is given by

$$|f_m| = \sum_{(x,y)\in E_v} F_v(x,y) \cdot F_p(f_m(x), f_m(y)).$$

If x and y are processes in N_v, then $f_m(x)$ and $f_m(y)$ are corresponding processors in N_p with the mapping function f_m. $F_v(x,y)$ is the communication requirement between x and y on the virtual topology, while $F_p(f_m(x), f_m(y))$ is the communication cost between two corresponding processors on the processor topology. The above equation is a generalized form of those defined in [9, 10] in that it includes the communication requirement as well as the communication cost.

Definition 2 *Given two weighted graphs $G_p(N_p, E_p)$ and $G_v(N_v, E_v)$ with their cost functions F_p and F_v, respectively, the mapping problem is to find a mapping function f_m which minimizes the weighted cardinality $|f_m|$.*

Based on Definition 2, the mapping problem is to find one among the $|N_v|!$ possible cases. The general formulation of this problem is known to be computationally equivalent to the graph isomorphism problem, which has been shown to be NP-complete [9, 11]. However, under certain circumstances, G_v and G_p can be reasonably simplified and thus efficient heuristic solutions exist.

There are some software packages such as Chaos [12] and SCOTCH [13] for graph partitioning. In applying to domain decomposition and mapping, however, these packages were designed primarily for typical parallel systems such as hypercube and mesh architectures rather than for switch-based irregular cluster systems.

3 Mapping for Irregular Cluster Systems

This section covers the mapping problem for switch-based irregular cluster systems. Then, two objects involved in the mapping, virtual topology and processor topology, are discussed. The concept of *routing topology* is also introduced.

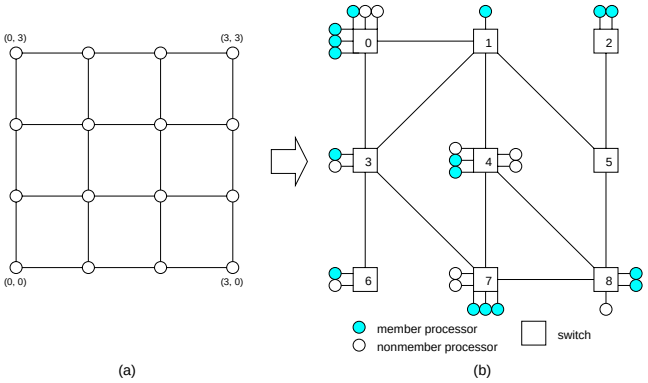

Figure 2: Virtual and processor topologies: (a) a virtual topology G_v and (b) a processor topology G_p.

3.1 Processor and Routing Topologies

We assume that G_v is a 2-D mesh (which is not a perfectly connected graph) and not weighted. We also assume that $|N_v| = |N_p|$, and there is one-to-one mapping between N_v and N_p. Each edge (s,t) in G_p is assumed to have a weight which represents the communication cost between the nodes s and t. As an example, consider the problem of mapping a 4×4 mesh type of virtual topology onto a processor topology consisting 16 processors as shown in Figure 2. Each switch has a set of ports and each port is connected to a computational node or another switch. Some ports may be left open and they can be used for future system expansion. The system in Figure 2 consists of nine 8-port switches, 26 processors, and 12 inter-switch links. In this example, the problem is to map the 4×4 mesh type of virtual topology onto the 16 processors (the dark circles in Figure 2(b)) selected for running a parallel application. The set of processors S selected are called *member processors*, and the number of member processors connected to a switch is referred as the *switch size*. For example, the switch size of switch 0 is four (*i.e.*, $|S_0| = 4$).

However, the processor topology G_p does not properly reflect the routing complexity and deadlock-free requirement. Therefore, a variant of the processor topology, called routing topology G'_p, will be used. A routing topology shows how the communication must occur between the switches for deadlock-free routing (shown later in this section). A routing topology is configured at the startup of a cluster system. Each switch computes its own *breadth-first spanning* (*BFS*) tree and all switches eventually agree on a unique spanning tree. The BFS tree chosen is the one that minimizes the height of the tree. Deadlock-free routing is achieved

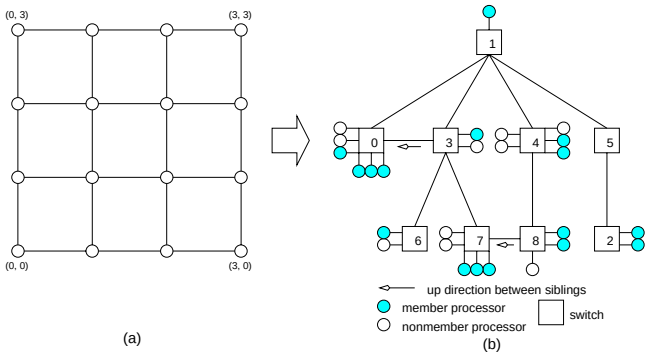

Figure 3: Virtual and routing topologies: (a) a virtual topology G_v and (b) a routing topology G'_p.

using a variant of the *turn model* [14], called *up/down routing* [4], with the following rule: *"A legal route traverses zero or more links in the upward direction until a node with a direct downward path is reached. This is followed by traversing zero or more links in the downward direction."*

For the processor topology of Figure 2(b), the corresponding routing topology G'_p is shown in Figure 3(b). The set of processors N'_p of G'_p is the same as N_p, but E'_p is different from E_p. Thus, the routing topology is denoted as $G'_p(N_p, E'_p)$. For example, switches 4 and 7 are immediate neighbors in G_p, as shown in Figure 2(b), but they are three links (hops) away in G'_p, as shown in Figure 3(b). This is due to the deadlock-free requirement of the routing algorithm and it implies that some communication links in the processor topology may not actually be utilized.

3.2 MPI Virtual Topology Model

There has been extensive research as well as actual implementations of message passing libraries for distributed memory multicomputers. Among them, MPI [15] has gained a wide acceptance in the industry mainly due to its simple interface. When multiple processes are created by `MPI_Init()` routine, they form a *communicator* among the process group to exchange messages. Depending on the communication pattern of an application, a node may exchange more messages with one particular node than it does with others. In order to exploit the application-specific characteristics, there must be a way to inform the communication pattern of a parallel application. As with other message-passing libraries[4], MPI provides the concept of virtual topology, on which applications can be implemented [9]. Using the `MPI_Cart_create()` and `MPI_Graph_create()` routines, a virtual topology can be specified.

In many parallel programming interfaces, a very little consideration is given to the mapping of processes to processors. Usually the mapping is either random or left to the user to be done manually. In case of MPI, the mapping between the virtual topology and the processor topology is implementation-dependent and beyond the scope of MPI. Therefore, our main effort in this paper is to propose efficient mapping algorithms that place the adjacent processes on adjacent processors as much as possible.

4 The Proposed Mapping Schemes

In this section, two mapping schemes, *switch-based mapping* (SM) and *binary mapping* (BM), are proposed. As mentioned before, simple naive approach is to randomly map a group of processes to processors, which is referred to as *random mapping* (RM) in this paper. Even though this approach does not allow an efficient mapping, it is used as a reference for comparing different mapping strategies. Time complexity of RM is $O(n)$, where n is the number of processors, *i.e.*, $n = |N_p|$.

4.1 Switch-Based Mapping (SM)

SM is based on the assumption that the communication cost between two processors connected to different switches are high. Therefore, adjacent processes in the virtual topology are mapped to processors connected to the same switch.

For the sake of illustration, consider the virtual topology G_v and the routing topology (and its associated cost matrix $\mathbf{M}'_p$) shown in Figure 4. As can be seen in $\mathbf{M}'_p$, inter-switch communication cost is high (as indicated by a 1), while intra-switch communication cost is negligible (as indicated by a 0). The SM algorithm tries to partition the virtual topology of 2-D mesh so that the adjacent processes are placed to the set of processors connected to the same switch. This mapping problem is similar to the 2-dimensional bin packing or the processor allocation problem, which is known to be NP-hard. Therefore, we adopt the *buddy heuristic* [18], where the virtual topology G_v is recursively divided into a partition with two buddies until

[4] In PARMACS [16] and p4 [17] libraries developed at the ANL (Argonne National Laboratory) and GMD (German National Research Center for Computer Science), the virtual topology can either be a Cartesian structure with up to three dimensions or a general graph [16].

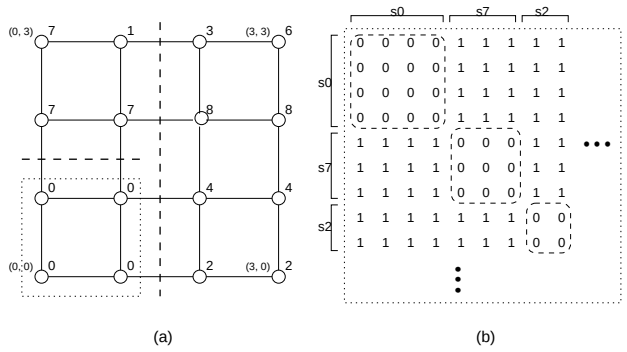

Figure 4: An example of the switch-based mapping: (a) a virtual topology G_v and (b) a cost matrix $\mathbf{M}'_p$.

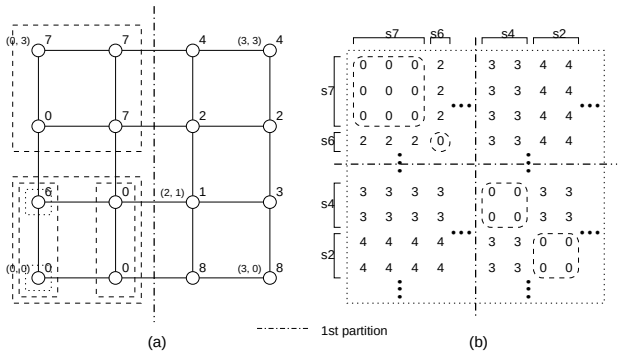

Figure 5: An example of the binary mapping: (a) a virtual topology G_v and (b) a cost matrix $\mathbf{M}'_p$.

the member processors of the largest switch can fit into a buddy.

In Figure 4, when the SM algorithm first maps the set of four processors S_0, N_v is first divided into two buddies with eight processes. One of the buddies in the left is divided again into two subsets, each subset having four processes. A set of processes in the lower left quadrant are then mapped to S_0. Next, the set of processors connected to the second largest switch, in this case S_7, are assigned to the virtual topology after a proper partition is obtained. This procedure is repeated until no processors remain in the routing topology.

The following summarizes the SM algorithm, which has a time complexity of $O(n \log n)$. For each of the switches, the buddy partitioning in the third step takes $O(\log n)$.

SM Algorithm $(G'_p(N_p, E'_p),\ G_v(N_v, E_v))$

1. Start with the set of processes N_v and the set of processors N_p.
2. From N_p, find the set of processors S connected to the (next) largest switch size $|S|$.
3. Find a subset of processes Q_i from N_v such that $|Q_i| = 2^{\lceil \log_2 |S| \rceil}$. If necessary, this is done by repeatedly dividing N_v into two buddies until a subset Q_i with $2^{\lceil \log_2 |S| \rceil}$ processes is obtained.
4. Map $|S|$ processes in Q_i to the set of processors S.
5. Remove the set of processes Q_i from N_v and the set of processors S from N_p.
6. Repeat Steps $2 \sim 5$ until there are no unmapped processes in N_v.

4.2 Binary Mapping (BM)

In SM, the strategy is to exploit the modularity of the switches by mapping adjacent processes to processors within the same switch, which reduces the complexity of mapping problem. However, the cost matrix used is overly simplified resulting in limited performance improvement. Moreover, in situations where such modularity is not clearly defined, SM may not be directly applicable. Binary mapping (BM) is a more general solution, where the routing topology as well as the virtual topology is recursively partitioned. This divide-and-conquer approach is acceptable only when the conquered solutions can be properly merged. In BM, the topologies are carefully subdivided so that the merging process is not required as explained below.

Formally, a *partition* of a graph $G(N, E)$ is a division of its vertices N into disjoint subsets $(N_0, N_1, \cdots, N_k)$. For a partition of G with two subsets (N_0, N_1), an *edge cut* is defined as the set of edges connected between vertices in N_0 and N_1. The basic idea is to partition both topologies so that the edge cut of G_v has minimum cost (little communication requirement) while the edge cut of G'_p has maximum cost (expensive links). If N_v is mapped to N_p, the edge cut of G_v is effectively mapped to that of G'_p. Therefore, the mapping is translated into "*little communication requirement on expensive links.*"

Figure 5 is the same as Figure 4, except the cost matrix $\mathbf{M}'_p$. Note that the numbers in $\mathbf{M}'_p$ are obtained from the routing topology. Binary partitioning of the 2-D mesh G_v with the minimum edge cost is shown in Figure 5(a). BM algorithm partitions G'_p so that its edge cut has maximum cost. The resultant mapping is shown in Figure 5(b). In this example, the partition of G'_p is given by $\{S_0 \cup S_6 \cup S_7\}$ and $\{S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_8\}$. The submatrix in the upper left quadrant of $\mathbf{M}'_p$ represents the former set while the lower right quadrant

represents the latter set. Communication cost along the edge cut is explained by the numbers in the rest of $\mathbf{M}'_p$, *i.e.*, those in the upper right and lower left quadrants. Maximizing the sum of those numbers would result in the optimal binary partitioning of G'_p.

This procedure is known as *graph partitioning problem*, which is known to be NP-hard. Therefore, we use the *Kernighan-Lin* heuristic [19] that requires $O(n^2 \log n)$ time. It begins with an initial partition with two sets, and at each iteration, the algorithm swaps subsets of nodes if this produces an improvement. It is noted that the quality of a partitioning depends strongly on the initial partition [19]. One possible initial partition can be having processors connected to a switch to be in the same set. Time complexity of the binary mapping strategy is $O(n^2 \log^2 n)$ since the Kernighan-Lin algorithm in Step 3 is repeated $\log n$ times.

BM Algorithm $(G'_p(N_p, E'_p),\ G_v(N_v, E_v))$

1. Perform a binary partitioning on N_v to generate two sets of processes, *i.e.*, $N_v = (Q_0, Q_1)$.
2. Let the initial partition of N_p to be (R_0, R_1). This is done by starting with the largest switch, and then forming a set R_0 by merging the member processors connected to the switch with those of the next largest switch(es) until the number of processors becomes $\frac{|N_p|^2}{2}$. Include the rest of the processors in R_1.
3. Use the Kernighan-Lin Heuristic to obtain an improved partition, $N_p = (R'_0, R'_1)$ which minimizes the edge cost.
4. If there remains only one node in a partition, mapping between partitions can be trivially performed. Otherwise,
   call `BM Algorithm` $(G'_{p0}(R'_0, E'_p),\ G_{v0}(Q_0, E_v))$, and
   call `BM Algorithm` $(G'_{p1}(R'_1, E'_p),\ G_{v1}(Q_1, E_v))$.

---

5 Performance Evaluation

In this section, the performance of the proposed mapping strategies is evaluated using extensive simulation.

5.1 Experiment Environment

It is important to pay more attention to the communication patterns between adjacent processes rather than between distant processes. Therefore, three different cases of inter-process communication patterns are considered to evaluate the performance of the proposed mapping strategies: *i.e.*, each process communicates (i) only with its adjacent processes, (ii) 90% of the messages with its adjacent processes and 10% of the messages with processes random distance away, and (iii) 80% of the messages with its adjacent processes and 20% of the messages with processes random distance away.

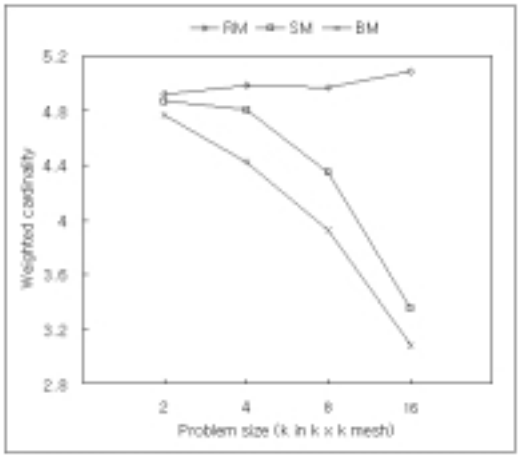

Figure 6: Weighted cardinality on the network of 256 nodes and 75 switches having 75% connectivity.

Cray T3D with PVM is quoted as having a startup time of 3 μsec whereas IBM SP-2 with MPI has a startup time of 35 μsec [21], where the startup time includes the software overheads for allocating buffers, copying messages, and initializing the router and DMA. Chien analyzed the router delay for various routing algorithms using a 0.8 micron gate array technology [22]. Based on that study and contemporary VLSI technology, the following default performance parameters are assumed: communication startup time (t_s) of 2 μsec, link propagation delay (t_p) of 20 $nsec$, and switch (router) delay (t_r) of 300 $nsec$. In addition, the network interface delay is assumed to be almost the same as the switch delay for our evaluation.

For all experiments, unless otherwise stated, the cluster system is assumed to consist of 256 nodes (processors) and 75 switches, and the network is interconnected with 8-port switches having 75% *connectivity*. The connectivity, or *connection ratio*, r of k-port switches is defined as the ratio of the average number of connected ports over k [20].

5.2 Results and Discussion

In this subsection, the simulation results of the proposed schemes are presented for the three different

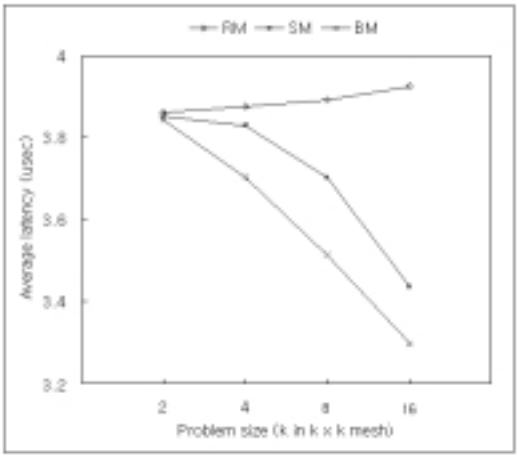

Figure 7: Communication latency with no distant communication, case-**(i)**.

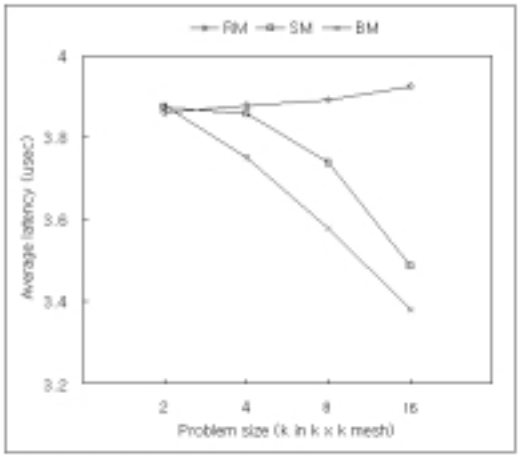

Figure 8: Communication latency with 10% distant communication, case-**(ii)**.

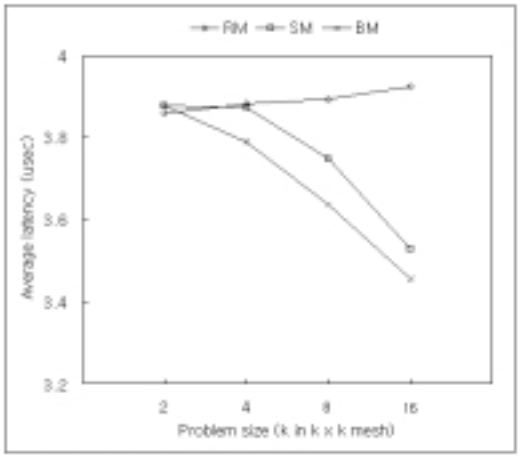

Figure 9: Communication latency with 20% distant communication, case-**(iii)**.

communication patterns in the virtual topology. The weighted cardinality is shown in Figure 6 for the three different mapping strategies. Notice from the figure that the two proposed mapping schemes show improvement over RM, while the improvement becomes larger as the size of mesh increases. This is intuitively obvious since the effect of mapping becomes more significant as the size of virtual topology increases. Also, as expected, BM significantly outperforms other mapping schemes.

The communication latency is evaluated for the three different communication patterns in terms of the average latency between communicating nodes. Figure 7 shows the simulation results of the case when each node communicates only with its adjacent nodes, *i.e.*, case**(i)**. Figure 8 and Figure 9 are for the cases **(ii)** and **(iii)**, respectively.

It can be seen in Figure 7 that for a virtual topology of a 16×16 mesh, SM and BM result in 15% and 20% shorter communication latency compared to RM, respectively. Interestingly, the three plots display almost the same trends as the plot for the comparison of weighted cardinality. This is because the weighted cardinality represents the degree of closeness between the routing topology and virtual topology after mapping. Less weighted cardinality implies that the average routing distance between adjacent processes is small, while the communication latency between them is linearly proportional to this value. Also notice that the improvement effect of BM will be more significant as communication between nonadjacent processes prevails.

6 Conclusion

In this paper, the mapping problem for switch-based cluster systems with irregular networks was addressed. Since the mapping problem is computationally NP-complete in nature, two different heuristic mapping algorithms, having different cost-performance tradeoff, were proposed. The simulation results show that for a virtual topology of a 16×16 mesh, the two proposed mapping schemes result in about $15 \sim 20\%$ shorter communication latency compared to random mapping. The efficiency of the mapping schemes becomes more significant as the size of the virtual topology increases. Based on the observed results, we believe that the proposed algorithms may be more beneficial when they

are applied to metacomputing and cluster of cluster systems, where communication costs are an order of magnitude different depending on the relative position of the processor nodes.

We are currently investigating heuristic algorithms that tackle the mapping problem for different virtual and processor topologies. We also plan to implement the proposed schemes in an MPI environment to study its cost-performance tradeoff.

References

[1] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*, Prentice-Hall Inc., NJ, 1999.

[2] G. F. Pfister, *In Search of Clusters*, 2nd Edition, Chapter 5, Prentce-Hall, Inc., NJ, 1998.

[3] H. Chen and P. Wyckoff, "Simulation Studies of Gigabit Ethernet versus Myrinet Using Real Application Cores," *Proc. of the 4th Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing*, Jan. 2000.

[4] M. D. Schroeder, *et.al.*, "Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links," *SRC Research Report*, No.59, Digital Equipment Corporation, April 1990.

[5] A. M. Mainwaring, B. N. Chun, S. Schleimer, and D. S. Wilkerson, "System Area Network Mapping," *Proc. of the Annual Symposium on Parallel Algorithms and Architectures*, 1997.

[6] W. Qiao and L. M. Ni, "Adaptive Routing in Irregular Networks Using Cut-Through Switches," *Proc. of the International Conference on Parallel Processing*, 1996.

[7] F. Silla, J. Duato, A. Sivasubramaniam and C. R. Das, "Virtual Channel Multiplexing in Networks of Workstations with Irregular Topology," *Proc. of the International Conference on High Performance Computing*, pp. 147-154, December 1998.

[8] J. Duato, S. Yalamanchile, and L. Ni, *Interconnection Networks: An Engineering Approach*, pp. 175-226, IEEE Computer Society, Los Alamitos, CA, 1997.

[9] T. Hatazaki, "Rank Reordering Strategies for MPI Topology Creation Functions," *Lecture Notes in Computer Science*, Vol. 1497, pp. 188-195, 1998.

[10] S. H. Bokhari, "On the Mapping Problem," *IEEE Transactions on Computers*, Vol. C-30, No. 3, pp. 207-214, March 1981.

[11] O. Krämer and H. Mühlenbein, "Mapping Strategies in Message-Based Multiprocessor Systems," *Parallel Computing*, Vol. 9, pp. 213-225, 1989.

[12] Sandia National Laboratories, Chaos, http://www.cs.sandia.gov/HPCCIT/chaos.html, Dec. 2000.

[13] Université Bordeaux, SCOTCH, http://www.labri.u-bordeaux.fr/Equipe/ALiENor/membre/pelegrin/scotch/, Dec. 2000.

[14] C. Glass and L. Ni, "The Turn Model for Adaptive Routing," *Journal of the ACM*, Vol. 41 No. 4, September 1994.

[15] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, Version 1.1, June 12, 1995.

[16] R. Calkin, R. Hempel, H.-C. Hoppe, and P. Wypior, "Portable programming with the PARMACS message-passing library," *Parallel Computing*, Vol. 20, pp. 614-632, 1994.

[17] R. M. Butler and E. L. Lusk, "Monitors, Messages, and Clusters: the p4 Parallel Programming System," *Parallel Computing*, 1994.

[18] K. Li and K.-H. Cheng, "Job Scheduling in a Partitionable Mesh Using a Two-Dimensional Buddy System Partitioning Scheme," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 413-422, Oct. 1991.

[19] A. Pothen, "Graph partitioning Algorithms with Applications to Scientific Computing," *Technical Report*, TR-97-03, Old Dominion University, 1997.

[20] R. Kesavan, K. Bondalapati, and D. K. Panda, "Multicast on Irregular Switch-Based Networks with Wormhole Routing," *Proc. of the International Symposium on High Performance Computer Architecture*, 1997.

[21] P. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann, San Francisco, CA, 1997.

[22] A. A. Chien, "A Cost and Speed Model for k-ary n-Cube Wormhole Routers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 2, pp. 150-162, Feb. 1998.