Mapping Concurrently-Communicating Subcubes in a Hypercube Multicomputer

Bing-rung Tsai and Kang G. Shin
Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
Email: {iast,kgshin}@eecs.umich.edu

Abstract

This paper considers the problem of mapping concurrently-communicating subcubes within a hypercube multicomputer so as to minimize inter-subcube communication traffic. Our objective is to minimize the total communication bandwidth required. Some important mathematical properties of subcube mappings are derived. Methods are proposed to modify existing optimization algorithms for finding optimal mappings. A subset of all possible mappings, called parallel mappings, *are found to possess some desirable properties. For some special case, optimal parallel mappings are also proved to be optimal among all mappings.*

1 Introduction

Subcube allocation — the problem of finding a subcube in a large target hypercube — has been studied extensively [1, 2] under the assumption that incoming subcube requests are independent. The commonly-used objective of subcube allocation is to minimize hypercube fragmentation. In certain applications, it may be necessary to cluster task modules into small groups, and each group is assigned to a subcube so as to minimize the distance of intra-group (or intra-subcube) communications. There can still be inter-group (or inter-subcube) communications, which may become a major performance bottleneck if these communicating modules/subcubes are not carefully placed within the hypercube. For example, the embedding of TMR modules into the hypercube, as discussed in [3], requires each TMR to be embedded into a 2-dimensional subcube, Q_2. So, a task composed of communicating modules is embedded into a set of communicating Q_2's. If a pair of Q_2's communicate frequently with each other but are placed far apart, then a large amount of inter-subcube communication will result, which may in turn degrade intra-subcube communication performance, as both inter- and intra- subcube communications use the same network. We will in this paper consider the problem of mapping communicating modules/subcubes in a hypercube by minimizing inter-subcube communication traffic.

The paper is organized as follows. Section 2 introduces basic notation and assumptions and formally states the optimization problem. In Section 3, we derive some mathematical properties of the objective function. Methods are introduced to modify existing optimization algorithms to solve the problem. A special class of mappings, called *parallel mappings*, are found to be useful because of their unique properties. Section 4 deals with sub-optimal mappings found with various heuristic algorithms. Through simulations, we show that when only sub-optimal mappings are considered, parallel mappings outperform non-parallel ones for most of the time. The paper concludes with Section 5.

2 Preliminaries

An n-dimensional hypercube, Q_n, consists of 2^n nodes which are connected in the form of a Boolean cube network. Each node is assigned a unique n-bit address, and two nodes are adjacent if and only if their addresses differ in exactly one bit position. We will henceforth use lower-case Greek letters to denote subcube addresses. Let Σ be a ternary symbol set $\{0, 1, *\}$, where $*$ represents don't care. Since each node in a Q_n is represented by n address bits, every subcube of the Q_n can be uniquely represented by a sequence of n ternary symbols in Σ, called the *address* of the corresponding subcube.

The work reported in this paper was supported in part by the Office of Naval Research under grants N00014-92-J-1080 and N00014-91-J-1115. Any opinions, findings, and recommendations in this paper are those of the authors and do not reflect the views of the ONR.

The Hamming distance between two subcubes $\alpha = a_0 a_1 ... a_{n-1}$ and $\beta = b_0 b_1 ... b_{n-1}$ of a Q_n is defined as $H(\alpha, \beta) = \sum_{i=0}^{n-1} h(a_i, b_i)$, where $h(a_i, b_i) = 1$ if $(a_i, b_i \in \{0,1\} \wedge a_i \neq b_i)$, and $h(a_i, b_i) = 0$ otherwise. For example, $H(00*, *11) = 1$, and $H(00*, 11*) = 2$.

We will assume that the sizes of communicating subcubes are known *a priori*, and communication events among these subcubes occur within a small *time window* [4, 5], i.e., messages are exchanged almost *concurrently*, as in the FFT computation. A *weighted task graph* G will be used to represent the communication behavior among the subcubes within the time window. $G = (V, E)$, where V is the set of vertices each denoting a subcube, and $E = \{(v_i, v_j, w_{ij})\}$ the set of weighted, directed edges from v_i to v_j, where w_{ij} denotes the weight on the edge, and represents the length of the message from v_i to v_j.

We will first discuss a simple case where all subcubes are of the same dimension, using the subcube communication model defined for uniform-size subcubes as in [6, 7]. Given (v_i, v_j, w_{ij}), $w_{ij} > 0$, suppose $\phi_i = a_n a_{n-1} ... a_1$ is the subcube address which v_i is mapped to, and $\phi_j = b_n b_{n-1} ... b_1$ is the subcube address v_j is mapped to. We define an *instance* of subcube communication as each node in ϕ_i sends a message of length w_{ij} to another node in ϕ_j. Since we are now dealing only with uniform-sized subcubes of dimension d (the general case of non-uniform sized subcubes are discussed in [8]), the number of messages sent is 2^d in each instance of communication. These messages are routed by the algorithm **Eq-subcube-route** proposed in [6], where a 1-to-1 mapping function is found between source and destination nodes, and a message between each source-destination pair is routed through a shortest path. Also, all messages in an instance of subcube communication are routed through edge-disjoint paths. From [7], we get the sum of lengths of these paths as $T(\phi_i, \phi_j) = M(\phi_i, \phi_j) 2^d$, where $M(\phi_i, \phi_j)$ is defined as $M(\phi_i, \phi_j) = \sum_{i=1}^{n} m(a_i, b_i)$, where

$$m(a_i, b_i) = \begin{cases} 1 & \text{if } a_i = \bar{b_i} \neq * \\ 0 & \text{if } a_i = b_i \\ 1/2 & \text{otherwise.} \end{cases}$$

Therefore, we define the *bandwidth* of such an instance of subcube communication to be $w_{ij} T(\phi_i, \phi_j)$.

In [4, 5] we have shown that for concurrently-communicating modules, the total bandwidth of a mapping is a good indicator of run-time performance. A mapping with a smaller total bandwidth almost always has better run-time performance, regardless of the underlying switching methods. We will henceforth use the *total bandwidth*, denoted by Φ, as our objective function. Formally, we have the following problem definition.

Given G and a target hypercube of dimension $n \geq \log(2^d \mid V \mid)$, i.e., it is large enough to accept all subcubes in V, our goal is to find a mapping of these subcubes into the target hypercube so that the total communication bandwidth of all communication instances is minimized. A mapping problem is described by a three tuple (G, d, n), and we want to minimize $\Phi = \sum_{v_i, v_j \in V} w_{ij} T(\phi_i, \phi_j)$, where ϕ_i and ϕ_j are the addresses of subcubes v_i and v_j mapped to, respectively.

3 Mathematical Properties

In this section we derive some mathematical properties which are important when finding optimal subcube mappings.

As in [7], we define the *frontier subcube* of α towards β, denoted by $\sigma_{\alpha \to \beta} = c_n c_{n-1} ... c_0$ such that $c_i = b_i$ if $a_i = * \wedge b_i \in \{0, 1\}$, and $c_i = a_i$ otherwise. For example, if $\alpha = 00**$ and $\beta = 1*1*$ then $\sigma_{\alpha \to \beta} = 001*$. $\sigma_{\alpha \to \beta}$ contains all the nodes in α which are closest to β, i.e., the Hamming distance of each node in this subcube of α to β is exactly $H(\alpha, \beta)$.

Subcubes α and β are said to be *parallel* with each other, denoted as $\alpha \parallel \beta$, if $\mid \sigma_{\alpha \to \beta} \mid = d = \mid \alpha \mid = \mid \beta \mid$. Note that in the degenerate case, all Q_0's (individual nodes) are parallel with one another. It follows that if $\alpha \parallel \beta$, $T_{\alpha\beta} = 2^d H(\alpha, \beta)$. Under a *parallel mapping* all communicating subcubes are mapped to subcube addresses parallel with one another. In a parallel mapping, the expression for Φ can be rewritten as $\Phi = \sum_{v_i, v_j \in V} w_{ij} H(\phi_i, \phi_j) 2^d$. Therefore, if we only consider parallel mappings, minimizing the total bandwidth is equivalent to minimizing $\sum_{v_i, v_j \in V} w_{ij} H(\phi_i, \phi_j)$. Note that if all subcubes are parallel, we can ignore the "don't cares", in subcube addresses when calculating their Hamming distances. The optimization problem for (G, d, n) is then reduced to finding optimal mappings for $(G, 0, n-d)$. This is just the optimization problem we treated in [4].

Parallel mappings also have the advantage that, even with the simplest (fixed-order) e-cube routing algorithm, all inter-subcube messages are routed through links that are never used for intra-subcube messages. For example, in Fig. 1(a), with e-cube routing, all inter-subcube messages are routed through links of the form $ab*$ or $*ab$, $a, b \in \{0, 1\}$, but never through links $a * b$ which are used only for intra-subcube messages. But in the non-parallel mapping shown in Fig. 1(b), when both v_0's send messages to v_3's, fixed-order routing will either route a message through the link between the nodes which v_0's are mapped to, or the link between the nodes which v_3's are mapped to. This situation can only be avoided by introducing a more complex routing algorithm.

Note that for arbitrary, but not necessarily parallel α and β, given the frontier subcube $\sigma_{\alpha\rightarrow\beta}$, each of α and β can be partitioned into parallel subcubes of the same size as $\sigma_{\alpha\rightarrow\beta}$. So the computation of $T(\alpha, \beta)$ can be broken down into the evaluation of T's between parallel subcubes of dimension $\mid \sigma_{\alpha\rightarrow\beta} \mid$ within α and β. If $\mid \sigma_{\alpha\rightarrow\beta} \mid = 0$, then the evaluation of T degenerates into the case of evaluating the Hamming distances between many pairs of individual nodes in α and β. As an example, consider several Q_2's in a Q_4. The two subcube addresses $01**$ and $00**$ are parallel, and $e^{01**,00**} = 2$. As for the two addresses $00**$ and $*1*1$, $e^{00**,*1*1} = 1$, and $T(00**, *1*1)$ can be expressed as $T(00*1, 01*1) + T(00*0, 11*1)$ or $T(00*1, 11*1) + T(00*0, 01*1)$. As for $00**$ and $**00$, $e^{00**,**00} = 0$, so $T(00**, **00)$ can only be expressed as in the definition of T, i.e., the sum of Hamming distances between individual nodes in these two subcubes.

As a result, all mappings for the problem (G, d, n) can be expressed as parallel mappings for (G^{d-f}, f, n), $f \le f^*$, where G^{d-f} is some graph constructed from G (to be explained below), and f^* is the dimension of the *greatest common frontier*(GCF) subcube, which can be calculated by counting the number of common positions in which $*$'s appear in all subcube addresses. For example, we have a problem of $(G, 2, 4)$ with G given in Fig. 2, and we have a non-parallel mapping $v_0 \rightarrow 0**0$, $v_1 \rightarrow *1*1$, $v_2 \rightarrow *0*1$ and $v_3 \rightarrow 1**0$. Then this mapping can be expressed as parallel mappings of either 16 Q_0's or 8 Q_1's, which are smaller or equal to the size of the GCF subcube of the three Q_2's. G^{d-f} is just 2^{d-f} *disjoint* copies of G:

- $V^{d-f} = \{v_{i,x} \mid v_i \in V,\ 0 \le x < 2^{d-f}\}$
- $E^{d-f} = \{(v_{i,x}, v_{j,x}, w_{ij}) \mid (v_i, v_j, w_{ij}) \in E, 0 \le x < 2^{d-f}\}$.

Therefore, if the value of f is known and if an optimal parallel mapping for (G^{d-f}, f, n) is an optimal mapping for (G, d, n), then the optimization algorithm for $(G, 0, n)$ can be applied to solve (G, d, n). However, in general, the value of f in an optimal mapping of (G, d, n) is not known, so we have to consider the worst case of $f = 0$ and construct G^{d-0}, which will be denoted as G^d for simplicity.

Note that any mapping for (G, d, n) can be expressed as some mapping for $(G^d, 0, n)$, but the converse is not true. In other words, the set containing all mappings for (G, d, n) is a subset of the set containing all mappings for $(G^d, 0, n)$. Therefore, if we find an optimal mapping for $(G^d, 0, n)$, it could be useless since it may not be a *valid* mapping for (G, d, n), i.e., copies of certain v_i are not mapped into a Q_d. For example, let us consider $(G, 1, 3)$ with G given in Fig. 2 and its corresponding G^1 is just two identical G's. Fig. 3 shows an optimal mapping for $(G^1, 0, 3)$, but this mapping is not valid for $(G, 1, 3)$ since copies of v_0 are not mapped into a Q_1.

It is possible to avoid this problem by modifying G^d into G^d_X. We will add some extra edges to E^d, which are of the form $\{(v_{ix}, v_{i(x+1)}, X) \mid 0 \le x < 2^d - 1\}$, and if $d > 1$, another edge $\{(v_{i(x+2^d-1)}, v_{ix}, X)\}$ is added. X is some sufficiently large number and its appropriate value is calculated by the method below.

When these edges are added, Φ of any mapping for $(G^d_X, 0, n)$ should be the value of Φ of some mapping for $(G^d, 0, n)$ plus M such that $M \ge \mid V \mid 2^d X$. The value X is chosen so that an optimal mapping found will always map all copies of v_i into a Q_d, $\forall v_i \in V$. And if $M \ge \mid V \mid 2^d X + X$, the mapping can never be optimal and is not valid for (G, d, n). So, we must have $\mid V \mid 2^d X + X + \Phi(opt(G^d, 0, n)) > \mid V \mid X 2^d + \Phi(wst(G, d, n))$, where $opt(G^d, 0, n)$ denotes the optimal mapping for $(G^d, 0, n)$ and $wst(G, d, n)$ the worst mapping for (G, d, n). So we must have $X > \Phi(wst(G, d, n)) - \Phi(opt(G^d, 0, n))$. We can substitute any upper-bound for $\Phi(wst(G, d, n))$ and any lower-bound for $\Phi(opt(G^d, 0, n))$ to obtain the value of X needed.

For example, to prevent an optimization algorithm from finding an invalid mapping as in Fig. 3, we construct G^1_X as in Fig. 4. An upper-bound of $\Phi(wst(G, 1, 3))$ is calculated by assuming the worst possible case that any pair of v_i and v_j are mapped 3 hops away from each other, which is the farthest distance in a Q_3. A lower-bound of $\Phi(opt(G^1, 0, 3))$ is calculated by assuming any pair of v_i and v_j are mapped adjacent to each other. So $X > 3 * 2(3 + 2 + 1) - 1 * 2(3 + 2 + 1) = 24$.

Therefore the optimization algorithm for $(G, 0, n)$ can find an optimal mapping for (G, d, n) by constructing G_X^d and apply the algorithm to $(G_X^d, 0, n)$. Since the optimization problem is NP-hard [4], the computation cost is much higher than the case of finding an optimal parallel mapping, where only the problem $(G, 0, n-d)$ needs to be considered. In what follows, we will show that for some special cases of G, an optimal parallel mapping is indeed optimal. Therefore, the optimization process can be greatly simplified.

We define a *sub-mapping* of a mapping for (G, d, n) as a set of node addresses in a Q_n which collectively contain a mapping for $(G, 0, n)$. Each mapping for (G, d, n) can be *partitioned* into 2^d such sub-mappings, and the node addresses which the copies of v_i mapped to must form a Q_d to make the mapping valid for (G, d, n). Note that we can partition a parallel mapping such that each of these 2^d sub-mappings lies within a Q_{n-d}, and each sub-mapping is a mapping for $(G, 0, n-d)$. Furthermore, an optimal parallel mapping for (G, d, n) can be partitioned into 2^d sub-mappings, each of which is an optimal mapping for $(G, 0, n-d)$. But this is not true in the case of non-parallel mappings. For example, in Fig. 1(a) we have an optimal parallel mapping for $(G, 1, 3)$ and in Fig. 1(b) a non-parallel mapping, with G given in Fig. 2. In the parallel mapping, each sub-mapping lies within a Q_2 and is an optimal mapping for $(G, 0, 2)$. In the non-parallel mapping, sub-mappings are not mappings for $(G, 0, 2)$.

The following proposition is stated without giving the proof, which is trivial.

Proposition 1: For a problem (G, d, n), if there exists a mapping better than an optimal parallel mapping, then the mapping can be partitioned into sub-mappings and there must be a sub-mapping whose Φ is smaller than that of $opt(G, 0, n-d)$, the optimal mapping for $(G, 0, n-d)$.

We will show that, if G is a star-like graph, i.e., all edges in G directed into or out from just one vertex, then there is no such sub-mapping.

Lemma 1: If G is a star, an optimal parallel mapping for a problem (G, d, n) must also be optimal among all mappings.

Proof: An optimal parallel mapping can be considered as 2^d copies of the optimal mapping of the problem $(G, 0, n-d)$, each denoted as $opt(G, 0, n-d)$, and the 2^d nodes that a given v_i mapped to form a Q_d. The Φ value of this mapping can be expressed as $2^d \Phi_{0,n-d}^{opt}$, where $\Phi_{0,n-d}^{opt}$ is the Φ value of $opt(G, 0, n-d)$. Suppose there is a non-parallel mapping with a smaller Φ. This mapping can also be partitioned into 2^d copies of sub-mappings. There must exist at least one sub-mapping whose Φ is smaller than $\Phi_{0,n-d}^{opt}$. To satisfy this condition, this sub-mapping must be some mapping for $(G, 0, m)$, $n-d < m \leq n$.

Without loss of generality, let us assume v_0 to be the central vertex of the star in G and is always mapped to the address $0^{n-d}*^d$. In the optimal parallel mapping, consider the sub-mapping in $*^{n-d}0^d$ where v_0 is mapped to 0^n. Since G is a star, the value of Φ is determined only by the distance of v_i's to v_0. If there is a mapping for $(G, 0, m)$ with a smaller Φ than $opt(G, 0, n-d)$, then some v_i's in $opt(G, 0, n-d)$ must be re-mapped to the subcube $0^{n-d}*^d$. These addresses v_i's originally mapped to and 0^n must form a Q_d; otherwise the copies of v_0 cannot be re-mapped into a Q_d. However, if the original addresses which these v_i's mapped to can form a Q_d, then re-mapping these v_i's into $0^{n-d}*^d$ cannot lower Φ. Otherwise, the original sub-mapping cannot be optimal in $(G, 0, n-d)$. □

We are still unable to prove the general case of arbitrary task graphs. However, while enumerating many low-dimensional cases, we could not find any non-parallel mapping better than optimal parallel mappings. It is our conjecture that optimal parallel mappings are indeed optimal.

4 Heuristic Mapping Strategies

In Section 3, we have discussed the mathematical properties of subcube mappings, and discussed a strategy to find an optimal mapping using a modified version of existing optimization algorithms developed for a simpler mapping problem. However, the complexity of these optimization algorithms remains exponential, and hence for large problem sizes, we need some heuristic algorithms to find good sub-optimal mappings.

We have shown that an optimal parallel mapping is also optimal among all mappings for certain special cases, and it is our conjecture that optimal mappings is indeed optimal. However, this does not imply that *all* parallel mappings are better than non-parallel mappings. Therefore, two important questions arise: When we use a heuristic algorithm to find a good sub-optimal parallel mapping, will this sub-optimal parallel mapping be worse than most non-parallel mappings? Do we need to consider all possible mappings, and not just parallel mappings when looking for a good sub-optimal mapping?

To answer these questions, we investigate several heuristic methods, and compare the performance of parallel and non-parallel mappings found with each heuristic. We also confirm that optimization of mappings with respect to Φ improves several other performance parameters as well. We will again focus on the discussion of the case of uniform-size communicating subcubes, since we have shown [8] that mapping variable-size subcubes can always be reduced to a uniform-size subcube mapping problem.

The simulated annealing method [9] is shown to be an effective algorithm for finding near-optimal solutions to NP-hard task-mapping problems [10, 11]. In [11], we investigated a simulated annealing method optimization process for finding good sub-optimal mappings for the problem $(G, 0, n)$. The implementation of the simulated annealing method here is based on parameters selected with a similar criterion as in [10]. We set the *initial temperature* $T_0 = 30$, the *new temperature* $T_{new} = 0.95T$, where T is the temperature in the last iteration. The *freezing point* is set so that a move increasing the objective function by a unit value has an acceptable probability of 2^{-31}. The perturb function is given by performing random 2-opt exchanges [12] on the original mapping. Since each instance of 2-opt exchange takes approximately the same amount of computing time, the expected computing time of an optimization process can be normalized and expressed as the average number of exchanges performed. Given the above parameters, the optimization process is found to terminate after 3000 ± 500 exchanges on 90% of inputs used in producing the data presented here.

In Table 1, we compare the performance of parallel mappings (sn-p) and non-parallel mappings (sn-np) found with the simulated annealing method. For sn-p mappings, the initial mapping is a random parallel mapping, and only exchanges among parallel subcubes are allowed. For sn-np mappings, only non-parallel mappings are considered. The inputs are generated with $\mid V \mid = 25$, $prob(w_{ij} > 0)$ = CCP (Concurrent Communication Probability), i.e., the probability that v_i communicates with v_j in the time window considered. w_{ij} is set to 20 when $w_{ij} > 0$, and each subcube is of dimension $d = 3$. Each data point is obtained by averaging results from 10,000 iterations. Deviation from the mean values is found to be reasonably small ($< 3\%$).

CCP	0.1	0.2	0.4	0.6	0.8
rand	13472	25952	49280	72640	96752
sn-p	8624	19072	41216	63968	89344
sn-np	9552	21344	45536	70480	95856

Table 1: Φ of mappings found by various strategies.

It is obvious that both sn-p and sn-np mappings improve over random mappings significantly when CCP is small. sn-p mappings have more than 15% improvement over random mappings when CCP < 0.6. Also, sn-p mappings outperform sn-np mappings consistently ($> 10\%$) for all CCP values. This is even more prominent when CCP > 0.5, where the margin between sn-np and random mappings narrows down.

When w_{ij}'s are constant as in our simulations, the optimization of Φ can also improve several important performance parameters of a mapping. The *communication diameter* of a mapping is the largest Hamming distance between two nodes belonging to a pair of communicating subcubes. The *average communication distance* is the average Hamming distance of all pair of nodes involved in inter-subcube communications. In Figs. 5 and 6, we show the three performance parameters plotted against CCP for the mappings found. Obviously, sn-p mappings outperform random and sn-np mappings in all cases, except in the case of average subcube distance, where sn-np has a lower value for various CCP values. This shows that non-parallel mappings are better only in minimizing the Hamming distance between subcubes, not necessarily between nodes.

5 Concluding Remarks

In this paper, we have addressed the problem of mapping a set of communicating subcubes in a hypercube by minimizing inter-subcube communication traffic. The communication model we used was based on the one proposed in [6, 7] for routing messages between subcubes of the same size. Our objective was to minimize the total inter-subcube communication bandwidth.

Several important mathematical properties of this type of mappings were derived. Methods were proposed to modify existing algorithms to find an optimal mapping. Parallel mappings were found to have certain desirable properties, and require less computation cost to find. It was also shown that for some special cases optimal parallel mappings were indeed optimal among all mappings.

We also showed by simulations that in heuristic algorithms such as simulated annealing methods and other fast heuristics, parallel mappings still outperformed non-parallel mappings in most cases. In our simulations, optimizing the proposed objective function also led to improvements in several other performance parameters.

References

[1] M. S. Chen and K. G. Shin, "Processor allocation in an n-cube multiprocessor using gray codes", *IEEE Trans. on Computers*, vol. C-36, pp. 1396–1407, Dec. 1987.

[2] D. D. Sharma and D. K. Pradhan, "A novel approach for subcubes allocation in hypercube multiprocessors", *in Proc. of the Fourth IEEE Intl. Symposium on Parallel and Distributed Processing*, pp. 336–345, 1992.

[3] D. L. Kiskis and K. G. Shin, "Embedding triple-modular redundancy into a hypercube architecture", *in Proc. of the Third Conf. on Hypercube Concurrent Computers and Applications*, pp. 337–345, Jan. 1988.

[4] B.-R. Tsai and K. G. Shin, "Communication–oriented assignment of task modules in hypercube multicomputers", *in Proc. 12–th Int'l Conf. on Distributed Comput. Syst.*, pp. 38–45, June 1992.

[5] B.-R. Tsai and K. G. Shin, "Mapping concurrent communicating modules to mesh multicomputers equipped with virtual channels", submitted to publication, 1994.

[6] S. Padmanabhan and C. Baru, "Routing between subcubes in a hypercube", *in Proc. of the 6th Distributed Memory Computing Conference*, pp. 295–298, Apr. 1991.

[7] M. S. Chen and K. G. Shin, "Subcube allocation and task migration in hypercube multiprocessor", *IEEE Trans. on Computers*, vol. C-39, pp. 1146–1155, Sep. 1990.

[8] B.-R. Tsai, *Mapping and Scheduling of Concurrent Communication Traffic in Multicomputer Networks*, PhD thesis, The University of Michigan, 1994.

[9] P. J. M. Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, 1987.

[10] F. Ercal, J. Ramanujam, and P. Sadayappan, "Task allocation onto a hypercube by recursive min-cut bipartitioning", *in Proc. of the Third Conf. on Hypercube Concurrent Computers and Applications*, pp. 210–221, Jan. 1988.

[11] B.-R. Tsai and K. G. Shin, "Communication-oriented assignment of task modules in faulty hypercube multicomputers", *IEEE Trans. on Computers*, vol. C-36, pp. 1396–1407, May 1994.

[12] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.

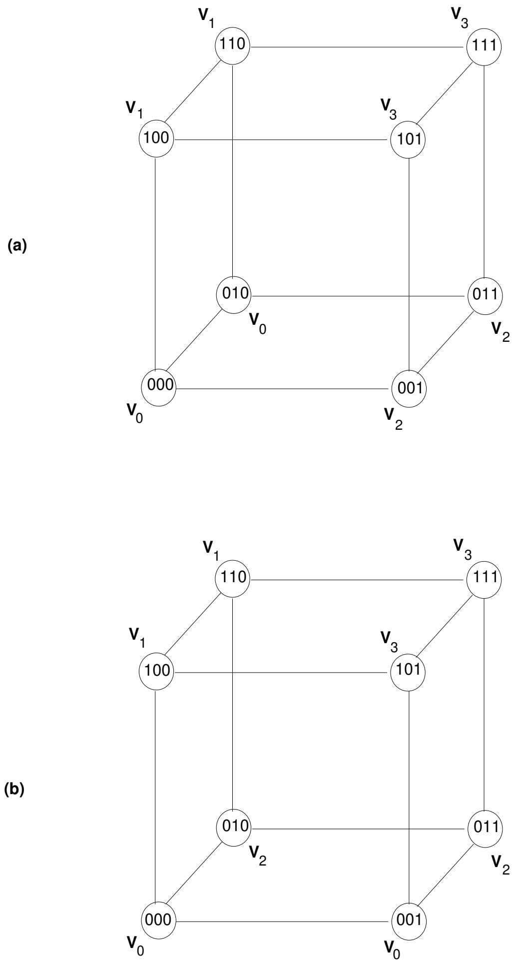

Figure 1: Two example mappings.

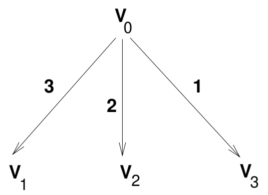

Figure 2: An example G.

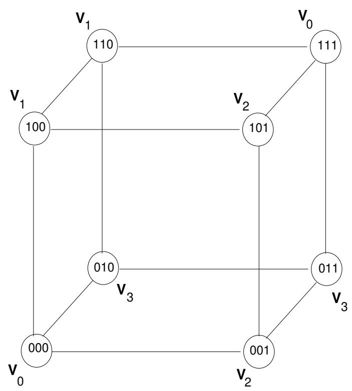

Figure 3: A mapping for $(G^1, 0, 3)$.

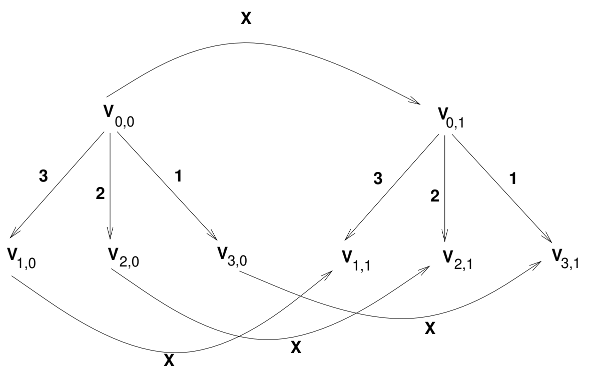

Figure 4: An example G^1_X.

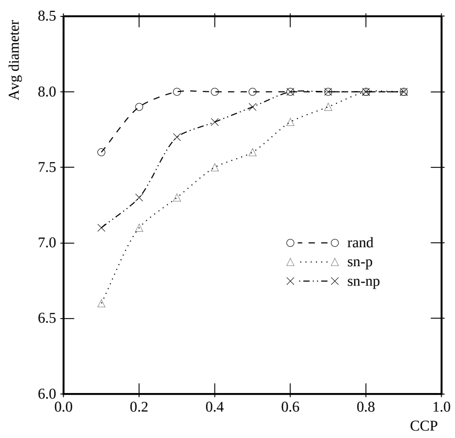

Figure 5: Average diameter versus CCP.

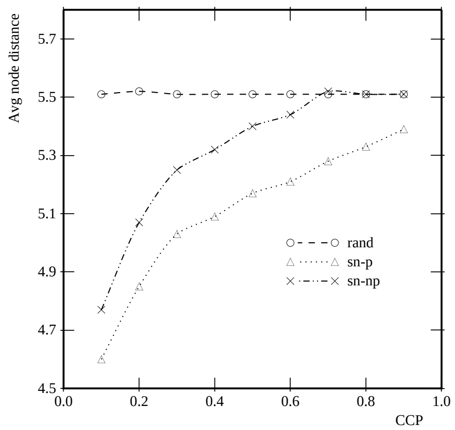

Figure 6: Average node distance versus CCP.