Embedding Meshes into Small Boolean Cubes

Ching-Tien Ho and S. Lennart Johnsson
YALEU/DCS/TR-791
May 1990

To appear in the Proceedings of the Fifth Distributed
Memory Computing Conference, Charleston, S.C., April 1990.

Embedding Meshes into Small Boolean Cubes

Ching-Tien Ho*
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
Ho@ibm.com

S. Lennart Johnsson†
Department of Computer Science
Yale University
New Haven, CT 06520
Johnsson@cs.yale.edu, Johnsson@think.com

Abstract

The embedding of arrays in Boolean cubes, when there are more array elements than nodes in the cube, can always be made with optimal load-factor by reshaping the array to a one-dimensional array. We show that the dilation for such an embedding is at most $\lceil \log_2(\frac{3}{2}\frac{2^n}{\max(\ell_0,\ell_1,\cdots,\ell_{k-1})}) \rceil$ for the embedding of an $\ell_0 \times \ell_1 \times \cdots \times \ell_{k-1}$ array in an n-cube. Dilation one embeddings can be obtained by splitting each axis into segments and assigning segments to nodes in the cube by a Gray code. The load-factor is optimal if the axis lengths contain sufficiently many powers of two. The congestion is minimized, if the segment lengths along the different axes are as equal as possible, for the cube configured with at most as many axes as the array. A further decrease in the congestion is possible if the array is partitioned into subarrays, and corresponding axis of different subarrays make use of edge-disjoint Hamiltonian cycles within subcubes. The congestion can also be reduced by using multiple paths between pairs of cube nodes, i.e., by using "fat" edges.

1 Introduction

Arrays are frequently used data structures, and many computations imply local interaction in a Cartesian space corresponding to the array. Modeling the array elements by nodes in a graph and connecting adjacent array elements defines a mesh with the same number of axes, and length of axes, as the array. In the following we refer to this induced mesh and the array interchangeably. Arrays are subgraphs of Boolean cubes. Using the subgraph property for the embedding of arrays requires that each array axis is assigned a unique subcube that contains at least as many nodes as there are elements along the axis [17]. For a large number of axes the number of cube nodes used for the embedding may be a very small fraction of the total number of nodes. A variety of techniques [8,19,15,2,6,5,7,9] have been devised to improve the utilization of the address space for the embedding of arrays with axes of a length not equal to a power of two. Any embedding with an increased utilization of cube nodes does not preserve adjacency for all mesh nodes. The subgraph property, and the techniques for increased utilization of the address space can also be used when the number of array elements exceeds the number of nodes in the Boolean cube.

Partitioning an array into a number of subarrays of equal size yields perfect load balance, if the number of subarrays is equal to the number of nodes in the Boolean cube. Some, or all of the axes are split into segments with one segment per node. Embedding the segments by a binary-reflected Gray code preserves adjacency. If the product of the axis lengths contains more powers of two than the number of dimensions in the cube, then several partitionings, or *factorings* of the axes are possible, all of which yields perfect load balance. Making the length of the segments along the various axes as equal as possible minimizes the maximum number of mesh edges mapped to a cube edge, the congestion, with the cube configured as an array with at most as many edges as the array. If the number

*This work was done while the author was with the Department of Computer Science, Yale University.

†The author is also with Thinking Machines Corp., 245 First Street, Cambridge, MA 02142. This work was supported by ONR Contract No. N00014-86-K-0310.

of powers of two contained in the product of the axis lengths is less than the number of dimensions in the Boolean cube, then good load balance may either be obtained by splitting an axis that is of odd length after all powers of two are factored out, or by leaving some nodes unused. Splitting an axis of odd length always results in load imbalance. For arrays of a high dimension the splitting of several axes may lead to a load-factor[1] that is higher than if some nodes are left unused. Note that axis splitting may be desirable with respect to congestion, even if load balanced splittings exist.

For instance, an 8×9 logic array can be allocated to the nodes in a 3-cube with one 1×9 subarray per node. The load-factor is optimal, but nine mesh edges are mapped to each used cube edge. By mapping the 8×9 logic array to a 4×2 array of cube nodes, instead of the 8×1 array of nodes, the maximum number of mesh edges mapped to a cube edge is five. In this mapping some nodes receive a 2×4 logic array, some a 2×5 logic array. Optimal load-factor can be achieved by moving one of the elements in the 2×5 subarray adjacent to a 2×4 subarray into the cube node holding that subarray. The communication load between the two nodes involved in this transaction increases, but the maximum load does not.

If the product of the axis lengths does not contain sufficiently many factors of two in order to assign subarrays of equal size to all nodes in the cube, then it may be preferable with respect to the load-factor to reduce the number of segments, and leave some nodes unused. We give some bounds on the load imbalance, and the maximum distance between adjacent array elements, the dilation, when some of the axes are partitioned into a number of segments that is not a power of two. The bounds are based on embedding the segments along the different axes by the techniques in [8,19,15,2,6,5,7,9] instead of a binary-reflected Gray code embedding.

For one-dimensional arrays it is trivial to determine the segment lengths for an optimal load-factor. Adjacency is preserved by mapping the segments to nodes by a binary-reflected Gray code. For higher dimensional arrays partitioning each axis independently no longer yields an optimal load-factor, in general. But, by reshaping arrays with multiple axes to arrays with a single axis, optimal segment lengths with respect to the load-factor are easily determined. Embedding the segments by a binary-reflected Gray code preserves adjacency within the one-dimensional array, but adjacency within the original array is not guaranteed. We show that the maximum distance between adjacent array elements when embedded with this strategy is bounded by the logarithm of the size of the array divided by the product of the load-factor and the length of the longest axis. The dilation, and congestion, may be improved by reshaping the original array to fewer axes, but not necessarily a single axis. But, an optimal load-factor is no longer guaranteed.

The congestion is minimized by using all edges of the Boolean cube. In a $2n$-cube there exist n edge-disjoint Hamiltonian paths. This property can be used to partition the array into subarrays, and by using a different embedding for each subarray. This technique was used in [22] for matrix multiplication. The dilation is not affected, but the load-factor may change. The existence of multiple paths between pairs of nodes in a Boolean cube can be used to create "fat-edges" [16] between adjacent cube nodes. The fat-edge technique may increase the dilation, but it does not change the load-factor.

Several techniques for equalizing the load by moving grid points between adjacent cube nodes have been devised [14,12,13,10,3,4]. Techniques such as simulated annealing, and neural networks have been used, in particular for irregular discretizations of continuous domains. Embedding meshes into small hypercubes by techniques similar to the ones used here have been considered by Ellis et al. [9]. The results presented in this paper apply to different schemes for defining segments, such as *cyclic* and *consecutive* allocation [20].

2 Graph decomposition

Let $\mathcal{V}(G)$ be the node set and $\mathcal{E}(G)$ the edge set of a graph G. $|S|$ is the cardinality of a set S. The embedding function, $\varphi : G \rightarrow H$, maps each node in G to a node in H. The *dilation* of the embedding φ is the maximum distance of $\varphi(i)$ and $\varphi(j)$ for all $(i,j) \in \mathcal{E}(G)$. The *load-factor* is the maximum number of nodes in G that are mapped

[1] All technical terms are defined in Section 2.

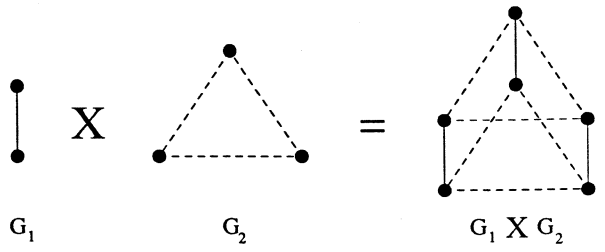

Figure 1: A product graph and two types of edges in the product graph.

to a node in H. Clearly, the optimal load-factor is $\lceil |\mathcal{V}(G)|/|\mathcal{V}(H)| \rceil$. The *congestion* is the maximum number of edges in G that are mapped to the same edge in H. $\lceil x \rceil_2$ denotes $2^{\lceil \log_2 x \rceil}$. The (Cartesian) *product* graph $G_1 \times G_2$ of two graphs G_1 and G_2 is defined as

$$\mathcal{V}(G_1 \times G_2) = \{[u_i, v_i] \mid \forall u_i \in \mathcal{V}(G_1), v_i \in \mathcal{V}(G_2)\},$$

$$\mathcal{E}(G_1 \times G_2) = \{([u_i, v_i], [u_i, v_j]) \mid \forall u_i \in \mathcal{V}(G_1),$$

$$(v_i, v_j) \in \mathcal{E}(G_2)\} \bigcup$$

$$\{([u_i, v_i], [u_j, v_i]) \mid \forall v_i \in \mathcal{V}(G_2), (u_i, u_j) \in \mathcal{E}(G_1)\}.$$

We will refer to the first form of edges as type-G_2 edges and the second form as type-G_1 edges. Figure 1 shows a product graph in which solid edges are of type-G_1 and dashed edges of type-G_2. $G_1 \times G_2$ can be derived by replacing each vertex of G_1 by G_2 and replacing each edge of G_1 by a set of edges connecting corresponding vertices of G_2. Note that the product operator "$\times$" is commutative and associative. Also, $|\mathcal{V}(G_1 \times G_2)| = |\mathcal{V}(G_1)| * |\mathcal{V}(G_2)|$ and $|\mathcal{E}(G_1 \times G_2)| = |\mathcal{V}(G_1)| * |\mathcal{E}(G_2)| + |\mathcal{V}(G_2)| * |\mathcal{E}(G_1)|$.

Lemma 1 *Let φ_i be an embedding function which maps a graph G_i into a graph H_i with load-factor f_i, dilation d_i and congestion c_i, $i \in \{1, 2\}$. Then, there exists an embedding function which maps the graph $G_1 \times G_2$ into the graph $H_1 \times H_2$ with load-factor $f = f_1 f_2$, dilation $d = \max(d_1, d_2)$ and congestion $c = \max(f_1 c_2, f_2 c_1)$. Furthermore, for the graph $H_1 \times H_2$, the congestion of a type-H_1 edge increases by at most a factor of f_2, and the congestion of a type-H_2 edge increases by at most a factor of f_1.*

Proof: We define the new embedding function by applying φ_1 to each copy of H_1 and applying φ_2 to each copy of H_2. Consider a node $u_i \in H_1$ and a node $v_i \in H_2$. There are at most f_1 nodes in G_1 that map to node u_i. Similarly, there are at most f_2 nodes in G_2 that map to node v_i. The corresponding product node in $H_1 \times H_2$, $[u_i, v_i]$ contains at most $f_1 f_2$ nodes in $G_1 \times G_2$.

For the dilation, each type-G_1 edge in $G_1 \times G_2$ is mapped to a copy of H_1 by φ_1 with dilation d_1. Similarly, each type-G_2 edge is mapped to a copy of H_2 by φ_2 with dilation d_2.

For the congestion, consider an edge incident to node $[u_i, v_i]$. It is either a type-H_1 edge or a type-H_2 edge. The congestion is $\leq f_1 c_2$ for the former edges by the definition of graph products. Similarly, edges of the latter type has congestion $\leq f_2 c_1$. ∎

Lemma 2 *Let φ be an embedding function which maps an $\ell_0 \times \ell_1 \times \cdots \times \ell_{k-1}$ mesh M into an n-cube with load-factor f, dilation d, and congestion c_i for any cube edge to which edges of axis i is mapped. Then, there exists an embedding function $\tilde{\varphi}$ that maps an $\ell_0 \ell'_0 \times \ell_1 \ell'_1 \times \cdots \times \ell_{k-1} \ell'_{k-1}$ mesh $\tilde{M}$ into an n-cube with load-factor $\tilde{f} = f \prod_{i=0}^{k-1} \ell'_i$, dilation $\tilde{d} = d$, and congestion $\tilde{c} \leq \max_{i=0}^{k-1} \{(c_i \prod_{j=0}^{k-1} \ell'_j)/\ell'_i\}$.*

Proof: With respect to the congestion, we show that the congestion for the edges of axis 0 is $\leq c_0 \prod_{i=1}^{k-1} \ell'_i$ without loss of generality. Consider the following two facts:

1. By Lemma 1 and the embedding function φ for the mesh M, an $\ell_0 \ell'_0 \times \ell_1 \times \ell_2 \times \cdots \times \ell_{k-1}$ mesh M_1 can be embedded into an n-cube with load-factor $f \ell'_0$, dilation d, and congestion c_0 of the edges of axis 0.

2. An $\ell'_1 \times \ell'_2 \times \cdots \times \ell'_{k-1}$ mesh M_2 can be embedded into a 0-cube (i.e., one-node cube) with load-factor $\prod_{i=1}^{k-1} \ell'_i$, dilation 0, and congestion 0.

From Lemma 1, the mesh $M_1 \times M_2$ can be embedded into an n-cube with load-factor $f \prod_{i=0}^{k-1} \ell'_i$, dilation d, and congestion for the edges of the axis 0 being $c_0 \prod_{i=1}^{k-1} \ell'_i$. Since the mesh $\tilde{M}$ is a subgraph

of the mesh $M_1 \times M_2$ [23], [18], the load-factor, dilation and congestion for the edges of axis 0 also hold for the mesh $\tilde{M}$. ∎

The properties of the load-factor in this lemma was also observed independently in [9].

3 Embeddings based on axis splittings

If the product of the axis lengths contains at least as many powers of two as there are dimensions in the Boolean cube, then adjacency can be preserved with an optimal load-factor. We state it formally in the following theorem.

Theorem 1 *An $\ell_0 2^{n_0} \times \ell_1 2^{n_1} \times \cdots \times \ell_{k-1} 2^{n_{k-1}}$ mesh M can be embedded into an $(\sum_{i=0}^{k-1} n_i)$-cube with dilation one, congestion $(\prod_{i=0}^{k-1} \ell_i)/\min_i\{\ell_i\}$, and optimal load-factor.*

Proof: Simply apply Gray code embedding to φ in Lemma 2 in mapping a $2^{n_0} \times 2^{n_1} \times \cdots \times 2^{n_{k-1}}$ mesh into a $(\sum_{i=0}^{k-1} n_i)$-cube with load-factor one, dilation one and congestion one. ∎

For example, a $100 \times 100 \times 100$ mesh can be embedded in up to a 6-cube with dilation one and optimal load-factor, either with *consecutive* or *cyclic* partitioning.

Even if the length of an axis is not divisible by a power of two partitioning each axis into a number of segments that correspond to some such power allows for an efficient embedding of the segments by a binary-reflected Gray code. Adjacency is preserved. The load-factor is minimized if $\Pi_{i=0}^{k-1} \lceil \ell_i/2^{n_i} \rceil$ is minimized, where $\sum_{i=0}^{k-1} n_i = n$. The optimal load-factor is $\lceil \Pi_{i=0}^{k-1} \ell_i/2^n \rceil$.

The load-factor obtained by creating as many subarrays as there are nodes in the Boolean cube by partitioning each axis, may not yield the best load-factor among all possible partitionings of individual axis. A better load-factor may actually be obtained by leaving some cube nodes unused. The following theorem gives a bound on the load-factor for some such partitionings.

Theorem 2 *An $\ell_0 \times \ell_1 \times \cdots \times \ell_{k-1}$ mesh M can be embedded in an n-cube with load-factor optimal within a factor of two, and dilation at most two for $k = 2$, at most 7 for $k = 3$ and at most $4k + 1$ for $k > 3$, if there exist values ℓ'_i and ℓ''_i such that $\ell_i = \ell'_i \ell''_i$ for all $0 \leq i < k$, and $\lceil \log_2 \prod_{i=0}^{k-1} \ell'_i \rceil = n$.*

Proof: Apply the embedding techniques devised for one element per node to the array $\ell'_0 \times \ell'_1 \times \cdots \times \ell'_{k-1}$. ∎

For instance, embedding a 35×27 array into a 6-cube can be made by factoring it into a 7×9 array to be embedded in the 6-cube by the techniques in [5,6,18], and a local 5×3 array in each cube node. The dilation is two and the load-factor is 15 (optimal). All nodes assigned array elements have the same number of elements assigned to them.

4 Embeddings by combining axes.

In the embedding of the 8×9 mesh in a 3-cube, the optimal load-factor could be obtained by moving a single node between a pair of partitions. The dilation remained minimal, but the congestion increased. One technique for determining which nodes of the logic array should be allocated to the same cube node for an optimal load-factor is to reshape the array into a one-dimensional array, partition this array for optimal load-factor, and embed the partitions by a binary-reflected Gray code. In general, the dilation will increase through such an embedding, and so will the congestion. As a compromise, the array can be reshaped into an array with fewer axes, and a binary-reflected Gray code used for each axis in the new logic array. Such a strategy does not guarantee optimal load-factor, however. Theorem 3 establishes an upper bound on the dilation for reshaping an array to a one-dimensional array that is partitioned for optimal load-factor and embedded by a binary-reflected Gray code.

Let $G(i)$ be the binary-reflected Gray code [25] of i and $Hamming(i, j)$ be the Hamming distance between i and j. The following lemma gives a bound

on the Hamming distance between two Gray coded integers as a function of the absolute difference in their values. The bound is due to Yuen [27]. For convenience, we include the proof below.

Lemma 3 [27]
Hamming$(G(i), G(j)) \leq \lceil \log_2(3d/2) \rceil$, *if* $|i-j| = d$.

Proof: We prove instead that if *Hamming*$(G(i), G(j)) = m$, then $|i-j| > 2^m/3$. First observe that $G(i) \oplus G(j) = i \oplus \lfloor \frac{i}{2} \rfloor \oplus j \oplus \lfloor \frac{j}{2} \rfloor = G(i \oplus j)$; thus the Gray code of $\ell = i \oplus j$ has m bits equal to one. Let $k_1 < k_2 < \cdots < k_m$ be the positions of the m ones in $G(\ell)$. Then $\ell_k = 1$, $k_{m-1} < k \leq k_m$; $\ell_k = 0$, $k_{m-2} < k \leq k_{m-1}$; and so forth. Note that $k_m \geq m$.

Without loss of generality assume $i > j$. Then, given $\ell = i \oplus j$, $i - j$ is clearly minimized if

$$i_k = \begin{cases} 1, & k = k_m, \\ 0, & \text{otherwise}, \end{cases}$$

$$j_k = \begin{cases} 1, & \ell_k = 1 \text{ and } k \neq k_m, \\ 0, & \text{otherwise}. \end{cases}$$

It follows that

$$\begin{aligned} i - j &= 2^{k_m - 1} - 2^{k_m - 2} - \cdots \\ &\quad - 2^{k_{m-1}} - 2^{k_{m-2}-1} - \cdots \\ &= 2^{k_{m-1}} - 2^{k_{m-2}-1} - \cdots. \end{aligned}$$

Consequently, if $k_{m-1} \geq m$, we have

$$i - j > 2^{k_{m-1}} - 2^{k_{m-2}} \geq 2^{k_{m-1}-1} \geq 2^{m-1} > 2^m/3.$$

On the other hand, if $k_{m-1} = m - 1$, then $k_i = i$, $i \leq m - 1$, thus,

$$\begin{aligned} i - j &= 2^{m-1} - 2^{m-3} - 2^{m-5} - \cdots \\ &> 2^{m-1} - 2^{m-3}(1 + \frac{1}{4} + \frac{1}{16} + \cdots) \\ &= 2^{m-1} - \frac{4}{3}(2^{m-3}) = 2^m/3. \quad \blacksquare \end{aligned}$$

Note, that the Hamming distance in Lemma 3 is a tight upper bound for a binary-reflected Gray code, but it is not tight for any Gray code. The tight lower bound for the Hamming distance in Lemma 3 using *any* Gray code is $\lfloor \log_2 d \rfloor + 1$, except for $d = 4^i + 1$, $i \geq 1$, in which case it is $\lfloor \log_2(d+1) \rfloor + 1$. Recently, Madhavapeddy and Sudborough [24] defined a Gray code, called window code, that achieves the lower bound. However, the window code is defined only when d is given (i.e., different d's might yield different window codes) and the improvement of the Hamming distance over the binary-reflected Gray code is either zero or one, depending on d.

The following theorem gives an upper bound on the dilation of arrays reshaped into a one-dimensional array partitioned for optimal load-factor, then embedded by a binary-reflected Gray code. If the window code is used, then the bound on the dilation is modified accordingly.

Theorem 3 *An* $\ell_0 \times \ell_1 \times \cdots \times \ell_{k-1}$ *mesh can be embedded into an n-cube with optimal load-factor and dilation* $\leq \lceil \log_2(\frac{3}{2} \frac{2^n}{\max(\ell_0, \ell_1, \cdots, \ell_{k-1})}) \rceil$.

Proof: Assume $\ell_0 \geq \ell_1 \geq \cdots \geq \ell_{k-1}$ without loss of generality. Define a function f that maps each node $(x_0, x_1, \cdots, x_{k-1})$, $0 \leq x_i < \ell_i$, in the logic array to a distinct node in a one-dimensional array through reshaping where the shortest axis is labeled first, and the longest axis labeled last.

$$\sum_{i=0}^{k-1} (x_i \prod_{j=i+1}^{k-1} \ell_j).$$

The range of indices is 0 to $(\prod_{i=0}^{k-1} \ell_i) - 1$. The optimal load-factor is $\beta = \lceil (\prod_{i=0}^{k-1} \ell_i)/2^n \rceil$. Segments of length β, or $\beta - 1$ in the reshaped array are embedded using a binary-reflected Gray code. A node with index y in the reshaped array is mapped to node $G(\lfloor y/\beta \rfloor)$ in the Boolean cube. Let $z_1 = f(x_0, x_1, \cdots, x_{k-1})$ and $z_2 = f(x_0, x_1, \cdots, x_i + 1, \cdots, x_{k-1})$ be the indices of two nodes adjacent in the original array. Then, $z_2 - z_1$ is maximized when $i = 0$, and $z_2 - z_1 \leq \prod_{j=1}^{k-1} \ell_j \leq \beta 2^n/\ell_0 \leq \beta d$. From the assumption of the theorem $\ell_0 \geq 2^n/d$ and $z_2 - z_1 \leq \beta d$. Hence, $\lfloor z_2/\beta \rfloor - \lfloor z_1/\beta \rfloor \leq d$, and the proof follows from lemma 3. $\blacksquare$

As a corollary, dilation one and optimal load-factor is achieved when $\max(\ell_0, \ell_1, \cdots, \ell_{k-1}) \geq 2^n$. When $\max(\ell_0, \ell_1, \cdots, \ell_{k-1}) \geq 2^{n-1}$, dilation two and optimal load-factor is attained. Figure 2 shows an embedding of a 9×7 mesh into a 4-cube with dilation two and optimal load-factor.

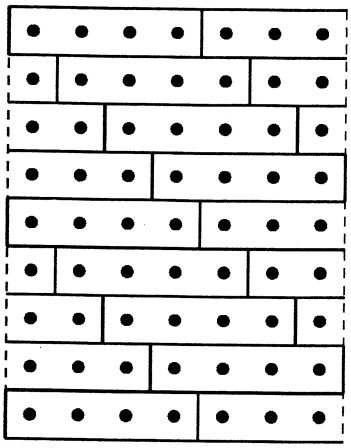

Figure 2: Embedding a 9×7 mesh into a 4-cube with dilation two and optimal load-factor.

Theorem 3 can be applied to any reshaping of an array. Subsets of axes may be reshaped into an array with fewer axes. The theorem gives a bound on the dilation for each new axis, and lemma 1 can be used to determine the dilation, load-factor, and congestion for the reshaped array. The reshaping technique can also be used in combination with factoring out a mesh with axis lengths being powers of two.

For instance, in case of embedding the 8×9 mesh into a 4×2 array, the problem is reduced to the embedding of a 2×9 array in a 1-cube. Applying theorem 3 to this problem yields optimal load-factor.

5 Reducing congestion

The mappings of a mesh into a smaller cube using the methods described above do not minimize the congestion, if the cube has at least twice as many dimensions as the mesh. For instance, for an array with two axes at most four cube dimensions are used, and for an array with three axes at most six dimensions are used. The number of edges in the array mapped to a cube edge increases rapidly with the size and dimensionality of the domain mapped to a single cube node. For a k-dimensional mesh of shape $\ell 2^{n_0} \times \ell 2^{n_1} \times \cdots \times \ell 2^{n_{k-1}}$ embedded into an $(\sum_{i=0}^{k-1} n_i)$-cube with optimal load-factor, and dilation one according to the technique in Section 3, the congestion is ℓ^{k-1}. To minimize the congestion the fact that a $2n$-cube contains n edge-disjoint Hamiltonian cycles can be used.

Lemma 4 [11] *If graphs C_1 and C_2 are Hamiltonian, then $C_1 \times C_2$ contains two edge-disjoint Hamiltonian cycles.*

Lemma 5 [1] *If the graph G contains two edge-disjoint Hamiltonian cycles, and the graph C is Hamiltonian, then the graph $G \times C$ contains three edge-disjoint Hamiltonian cycles.*

Lemma 6 [26] *A $2n$-cube contains n edge-disjoint Hamiltonian Cycles (HC's).*

Proof: We prove the lemma by induction on the dimensionality of the "even" cubes. A 2-cube is a cycle. A 4-cube contains two edge-disjoint HC's by Lemma 4. Assume that a $2k$-cube contains k edge-disjoint HC's for all $k \leq n$. We wish to show that a $(2n+2)$-cube contains $n+1$ edge-disjoint HC's.

- If $n+1$ is even, then decompose the $(2n+2)$-cube into two $(n+1)$-cubes. By the induction hypothesis, each $(n+1)$-cube contains $(n+1)/2$ edge-disjoint HC's. Consider the $(n+1)/2$ product graphs formed as the product of the ith HC in the two cubes, for $1 \leq i \leq (n+1)/2$. Each product graph contains two edge-disjoint HC's by Lemma 4. Furthermore, all these product graphs are edge-disjoint. Therefore, there exist $(n+1)$ edge-disjoint HC's in the $(2n+2)$-cube.
- If $n+1$ is odd, then decompose the $(2n+2)$-cube into an n-cube and an $(n+2)$-cube. Pair each but one HC of the n cube with a HC in the $n+2$-cube. This leaves one HC of the n-cube and two HC's of the $n+2$-cube unpaired. By Lemmas 4 and 5, there are $(n+1)$ edge-disjoint HC's in the $(2n+2)$-cube. ∎

For an $\ell_0 \times \ell_1 \times \cdots \times \ell_{k-1}$ array in which axis i is embedded in an n_i-cube, $0 \leq i < k$, $\sum_{i=0}^{k-1} n_i = n$, and ℓ_i is a multiple of $2^{n_i} n_i/2$ where n_i is even, the ith axis is partitioned into $n_i/2$ segments of the same size. Each such segment is embedded according to one of the $n_i/2$ edge-disjoint Hamiltonian

paths in the n_i-cube. The congestion of the cube edges corresponding to the ith array axis is reduced by a factor of $n_i/2$. From Theorem 1, we have the following theorem.

Theorem 4 *An $\ell_0 2^{n_0}\lfloor n_0/2 \rfloor \times \ell_1 2^{n_1}\lfloor n_1/2 \rfloor \times \cdots \times \ell_{k-1}2^{n_{k-1}}\lfloor n_{k-1}/2 \rfloor$ array can be embedded into an $(\sum_{i=0}^{k-1} n_i)$-cube with dilation one, congestion $(\prod_{i=0}^{k-1} \ell_i)/\min_i\{\ell_i\}$, and optimal load-factor.*

The congestion is minimized if n_i is chosen such that $\lceil \ell_i / \lfloor \frac{n_i}{2} \rfloor \rceil$ is minimized.

Instead of partitioning the array into $\Pi_{i=0}^{k-1} n_i/2^k$ subarrays each with its own embedding the "fat-edge" technique [16] can be used. There exist n edge-disjoint paths between any pair of nodes in an n-cube. Of these paths d are of length d and $n-d$ paths are of length $d+2$. By using more than one path between a pair of nodes, in particular by using paths of non-minimum length, the congestion can be reduced. In the fat-edge technique, the mesh embedding techniques of the previous sections can be used for the assignment of array indices to nodes in the cube. The fat-edge technique is used for routing messages, and does not affect the mapping of array indices. Not all paths between all pairs are edge-disjoint (though edges for a given pair are) and the scheduling of messages needs to account for this fact.

6 Summary

The strategy in mapping an array to an n-cube depends upon the desired objective for the mapping: optimal load-factor, dilation, or congestion, or what combination thereof should be minimized. A good starting point in exploring alternatives is to decompose a given array M into two arrays M_1 and M_2, $M_1 \times M_2 = M$, such that all axis lengths of M_1 are powers of two, and all axis lengths of M_2 are odd numbers. For instance, an array $M(100, 200)$ is decomposed into $M_1(4, 8)$ and $M_2(25, 25)$. Then, if

1. $\log_2 |M_1| = n$: a binary-reflected Gray code embedding of each axis yields optimal load-factor and dilation one. The congestion is $|M_2|$ divided by the length of the shortest axis of M_2, unless any of the techniques in Section 5 can be used to reduce the congestion.

2. $\log_2 |M_1| > n$: only a subset of all even partitionings of the axes can be used for assigning array indices to cube nodes. In general, there are many ways to choose the partitions while maintaining an optimal load-factor and dilation one by a binary-reflected Gray code encoding of the partitions along each axis. To minimize the congestion, the total boundary area of a partition should be minimized. For instance, the congestion for the embedding of a 100×320 array into a 4-cube is minimized by configuring the n-cube as a 2×8 array resulting in partitions of the shape 50×40.

3. $\log_2 |M_1| < n$. The embedding problem is reduced to the embedding of the array M_2, $\ell_0 \times \ell_1 \times \cdots \times \ell_{k-1}$, $\ell_0 \geq \ell_1 \geq \cdots \geq \ell_{k-1}$, in an $(n - \log_2 |M_1|)$-cube.

   Assigning array elements to all nodes of the cube requires that some axis of odd length (possibly after powers of two have been factored out) be partitioned. For an optimal load-factor the array M_2 is reshaped into an array by combining j axes such that $\Pi_{i=0}^{j-2}\ell_i \leq 2^n/|M_1| \leq \Pi_{i=0}^{j-1}\ell_i$, and a binary-reflected Gray code used for the new axis.

   For minimal load-factor under the constraint of unit dilation the longest axis is partitioned recursively until the number of partitions is equal to $2^n/|M_1|$. Hence, the embedding of the array $\ell_0 \times \ell_1 \times \cdots \times \ell_{k-1}$ is divided into the tasks of embedding the arrays $\lceil \frac{\ell_0}{2} \rceil \times \ell_1 \times \cdots \times \ell_{k-1}$ and $\lfloor \frac{\ell_0}{2} \rfloor \times \ell_1 \times \cdots \times \ell_{k-1}$ into two independent cubes of dimension $n - \log_2 |M_1| - 1$. In the recursion the partitioning is always applied to the longest axis. However, if $\lceil \frac{\ell_0}{2} \rceil$ is even, then the maximum power of two should be factored out before proceeding with the recursion.

   If instead minimal congestion is sought under the constraint of unit dilation, the cube is configured such that its shape is as congruent as possible to the array shape, making the partitions having an aspect ratio as close to one as possible.

   Leaving some nodes unused by partitioning an axis of odd length into an odd number of seg-

ments may sometimes yield a lower load-factor than if an even number of partitions is generated, if the axes are partitioned independently and the maximum number of subarrays are given. The embedding of a 35×27 array in a 6-cube illustrated this fact. With 7 partitions along the first axis and 9 along the second the load-factor is 15, which is optimal. The minimum load-factor if all nodes are used is 18.

Embedding by the axis splitting technique may increase the load-factor, while the axis combining technique may reduce it. The dilation may be reduced by axis splitting, and increased by the merging of axes. The dilation is increased, in general, if the fat-edge technique is used to reduce the congestion. Using edge-disjoint Hamiltonian paths to reduce congestion is equivalent to axis splitting, but does not affect the dilation. The communication along an axis for a subarray is determined by the product of all axis segments, but the segment along the axis of communication [21]. Hence, the congestion for a given array axis in the embedding is only reduced to the extent that the segments along the other axes are reduced in defining subarrays (for exploiting the multiple paths property). Splittings of an axis for multiple path embeddings increase the communication requirements in proportion to the number of paths utilized, and do not affect the congestion along the axis being split. Reducing the congestion by the fat-edge technique leaves the partitions unaffected, and hence the communication requirements.

Acknowledgement This work has been supported in part by the Office of Naval Research under Contract N00014-86-K-0310. The authors would like to thank Alan Wagner for his helpful comments on the construction of n edge-disjoint Hamiltonian cycles in a $2n$-cube and pointing of reference [1]; Ajit Agrawal for his helpful discussion on references [1] and [11]; Michel Jacquemin for translating the reference [1] from French into English; David Greenberg for stimulating discussion on Section 5; Seshu Madhavapeddy for discussions of Lemma 3 and the window code in [24].

References

[1] Jacques Aubert and Bernadette Schneider. Decomposition de la somme cartesienne d'un cycle et de l'union de deux cycles hamiltoniens en cycles hamiltoniens. *Descrete Mathematics*, 38:7–16, 1982.

[2] Said Bettayeb, Zevi Miller, and I. Hal Sudborough. Embedding grids into hypercubes. In *Proceedings of '88 AWOC: VLSI, Algorithms and Architectures Conf.*, Springer Verlag's Lecture Notes in Computer Science, no 319, 1988.

[3] Shahid H. Bokhari. On the mapping problem. *IEEE Trans. Computers*, C-30:207–214, 1981.

[4] Shahid H. Bokhari. Partitioning problems in parallel, pipelined and distributed computing. *IEEE Trans. Computers*, C-37(1):48–57, January 1988.

[5] M.Y. Chan. Dilation-2 embeddings of grids into hypercubes. In *1988 International Conf. on Parallel Processing*, The Pennsylvania State University Press, 1988.

[6] M.Y. Chan. *The Embedding of Grids into Optimal Hypercubes.* Technical Report, Computer Science Dept., University of Texas at Dallas, 1988.

[7] M.Y. Chan. Embeddings of 3-dimensional grids into optimal hypercubes. In *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, Vol. I*, pages 297–299, 1990.

[8] M.Y. Chan and F.Y.L. Chin. *On Embedding Rectangular Grids.* Technical Report TR-B2-87, Center of Computer Studies and Applications, University of Hong Kong, February 1987. to appear in IEEE Trans. Computers.

[9] John Ellis, Zevi Miller, and I. Hal Sudborough. *Compressing Meshes into Small Hypercubes.* Technical Report, Dept. of Computer Science, Univ. of Texas at Dallas, 1989.

[10] J.W. Flower, Steve W. Otto, and M.C. Salama. *A Preprocessor for Irregular Finite*

Element Problems. Technical Report CCCP-292, California Inst. of Technology, Pasadena, CA, June 1986.

[11] Marsha Foregger. Hamiltonian decompositions of products of cycles. *Descrete Mathematics*, 24:251–260, 1978.

[12] Geoffrey C. Fox. *Load Balancing and Sparse Matrix Vector Multiplication on the Hypercube.* Technical Report CCCP-327, California Inst. of Technology, Pasadena, CA, July 1986.

[13] Geoffrey C. Fox and Wojtek Furmanski. *Load Balancing by a Neural Network.* Technical Report CCCP-363, California Inst. of Technology, Pasadena, CA, September 1986.

[14] Geoffrey C. Fox and Steve W. Otto. *Concurrent Computation and the Theory of Complex Systems.* Technical Report CCCP-255, California Inst. of Technology, Pasadena, CA, March 1986.

[15] David S. Greenberg. *Minimum Expansion Embeddings of Meshes in Hypercubes.* Technical Report YALEU/DCS/RR-535, Dept. of Computer Science, Yale Univ., New Haven, CT, August 1987.

[16] David S. Greenberg and Sandeep N. Bhatt. *Routing Multiple Paths in Hypercubes.* Technical Report YALEU/DCS/RR-768, Dept. of Computer Science, Yale Univ., New Haven, CT, March 1990.

[17] I. Havel and J. Móravek. B-valuations of graphs. *Czech. Math. J.*, 22:338–351, 1972.

[18] Ching-Tien Ho and S. Lennart Johnsson. Embedding meshes in Boolean cubes by graph decomposition. *Journal of Parallel and Distributed Computing*, 8(4):325–339, April 1990.

[19] Ching-Tien Ho and S. Lennart Johnsson. On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two. In *1987 International Conf. on Parallel Processing*, pages 188–191, Penn State, 1987.

[20] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987. (Tech. Rep. YALEU/DCS/RR-361, Yale Univ., New Haven, CT, January 1985).

[21] S. Lennart Johnsson. *Optimal Communication in Distributed and Shared Memory Models of Computation on Network Architectures*, page . Morgan Kaufman, 1989.

[22] S. Lennart Johnsson and Ching-Tien Ho. *Multiplication of arbitrarily shaped matrices using the full communications bandwidth on Boolean cubes.* Technical Report YALEU/DCS/RR-721, Department of Computer Science, Yale University, July 1989.

[23] Yuen-Wah E. Ma and Lixin Tao. Embeddings among toruses and meshes. In *1987 International Conf. on Parallel Processing*, pages 178–187, IEEE Computer Society, 1987.

[24] Seshu Madhavapeddy and I. Hal Sudborough. *A Note on Cyclic Binary Codes.* Technical Report Tech. Report UTDCS-6-89, Dept. of Computer Science, Univ. of Texas at Dallas, March 1989. in preparation.

[25] E M. Reingold, J Nievergelt, and N Deo. *Combinatorial Algorithms.* Prentice-Hall, Englewood Cliffs. NJ, 1977.

[26] Alan Wagner. 1988. Personal communication.

[27] C.K. Yuen. The separability of Gray code. *IEEE Trans. on Information Theory*, 668, 1974.