Yale University
Department of Computer Science

Minimum Expansion Embeddings of Meshes in Hypercubes

David S. Greenberg

YALEU/DCS/TR-535
August 1987

This work was supported in part by the National Science Foundation under Grant MIP-8601885

Minimum Expansion Embeddings of Meshes in Hypercubes

David S. Greenberg

Abstract

We show how to embed every 2-dimensional mesh into a hypercube with expansion 1 and dilation 5. Previous embeddings with expansion 1 and $O(1)$ dilation have only applied to restricted cases.

1 Introduction

Problems in which the optimal use of one resource requires suboptimal use of another are common in Computer Science. Classic examples include tradeoffs between time and space, between chip area and time [Th79], and for parallel computing between number of processors and runtime.

This paper examines an expansion/dilation tradeoff for graph embeddings [HMR83]. The motivation is that we wish to program algorithms for communication graphs natural to their structure (eg. binary trees for divide and conquer algorithms or meshes for physical differential equations) but to run these programs on physical networks chosen for general-purpose parallel computation (eg. a hypercube or FFT). Thus we are left with the problem of mapping the communication graph onto the connection graph.

This work was supported in part by the National Science Foundation under Grant MIP-8601885

In particular, given a communication graph, M, the task is to determine the 'best' embedding, σ, from M onto a family of connection graphs, $\mathcal{H}$. The resources we measure are ***expansion*** and ***dilation.*** Expansion is the ratio of the number of nodes in H, the chosen sized connection graph, to the number of nodes in the smallest connection graph in the family having at least as many nodes as M. Dilation equals $\max_{i,j} d(\sigma(m_i), \sigma(m_j))$ where m_i and m_j are neighbors in M and $d(x, y)$ is the length of the shortest path between x and y in H. Expansion represents the fraction of processors utilized and dilation correlates to the time per communication (in number of connection link transmissions in H per edge transmission in M.)

This paper examines the case where the communication graph is a mesh and the connection graph is a hypercube. Earlier papers have explored the tradeoffs for the cases where the communication graph is a tree and the connection graph is a hypercube [BCLR86] and where both graphs are 2-D meshes [AlRo82].

Some meshes, such as the 4×8 mesh, are subgraphs of the hypercube of the same size and therefore have the best possible embeddings. For other meshes, however, such as the 5×11 mesh, either expansion or dilation must be nonoptimal. Any 2-D mesh can be embedded with expansion 2 and dilation 1 using a gray code. Many meshes can be embedded with expansion 1 and dilation 2 but no method is known which embeds all meshes with expansion 1 and dilation 2. This paper shows that for a cost of dilation 5 the expansion can be reduced to its minimum value. In particular we give an algorithm for embedding with dilation 5 ***any*** N node mesh into the smallest hypercube containing at least N nodes.

Section 2 briefly reviews gray codes. Section 3 presents key properties of hypercubes, section 4 describes the Reflection Embedding, and section 5 introduces ***percolation*** to improve the Reflection Embedding of section 4. The final section gives some applications and possible extensions.

2 Gray Codes for Mesh Embeddings

Definition 1 The *n-dimensional hypercube*, H_n, is a set of 2^n vertices labelled from 0 to $2^n - 1$ and an edge between any two vertices whose labels differ in exactly one bit position. All the edges between vertices differing in the ith bit are said to be in the ith *dimension.*

One nice property of a hypercube is that if all the edges in dimension i are removed and the ith bit of each label is ignored two half sized hypercubes are formed. Thus hypercubes can be recursively split into smaller hypercubes.

The recursive structure of the hypercube leads to the class of reflected gray codes.

Definition 2 An nth order *reflected gray code* is a sequence of dimensions defined inductively as:

$$\begin{aligned} G_0 &= 0 \\ G_n &= G_{n-1} \circ n \circ (G_{n-1})^R \\ \text{where } S^R &= \text{the sequence } S \text{ in reverse order} \\ \text{and } \circ &= \text{string concatenation.} \end{aligned}$$

The *flexed sequence*, $\tilde{G}$, is the sequence in which the order of dimensions is reversed. Thus:

$$\begin{aligned} \tilde{G}_0 &= n \\ \tilde{G}_i &= \tilde{G}_{n-i} \circ n - i \circ (\tilde{G}_{n-i})^R \end{aligned}$$

The ith dimension along the nth order sequence will be denoted as $G_n(i)$.

Example 1

$$
\begin{aligned}
G_4 &= 0102010301020104010201030102010 \\
\tilde{G}_4 &= 4342434143424340434243414342434 \\
G_4(6) &= 1
\end{aligned}
$$

Definition 3 Given a hypercube, a reflected gray code, and a distinguished vertex, the *gray code ordering* is the numbering of the vertices of the hypercube formed by assigning 0 to the distinguished vertex and assigning $i+1$ to the vertex reached by crossing dimension $G(i+1)$ from the vertex assigned value i. (See figure 1).

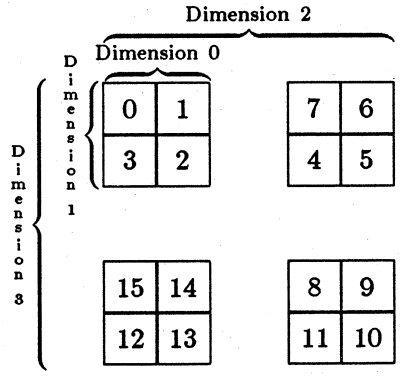

Figure 1: Gray Code Ordering of a 4-cube

An interesting property of the reflected gray code which we will use later is that between any two nodes whose gray code orderings differ by 2^i there is a path of distance two in the hypercube. For example although 8 is a distance 2^3 from 0 along G_4 it can be reached by crossing dimensions 2 and 3.

Lemma 1 *$\forall i, 0 < i < n$, if the gray code orderings or nodes u, v differ by 2^i then u and v are distance 2 apart in the hypercube.*

Proof: The proof follows directly from the observation that a length 2^i sequence is a length 2^{i-1} sequence followed by a single bit change followed by the *same* length 2^{i-1} sequence in reverse followed by another single bit. Thus every dimension crossed for the first length 2^{i-1} sequence is crossed back in the second and only the two other bits remain changed. ∎

Definition 4 A *d-dimensional mesh*, M_d, of size $l_1 \times l_2 \times \cdots \times l_d$ is a set nodes labelled $v = (v_1, v_2, \ldots, v_d)$ where $1 \leq v_j \leq l_j$ and and an edge between nodes v and w iff $\sum_{j=1}^{d} |v_j - w_j| = 1$. (In other words nodes are neighbors if their labels differ by one in a single dimension.) All edges between nodes which differ only in v_i are said to be in the ith *axis*.

Definition 5 An *embedding* of graph G into graph H is an injective mapping from the vertices of G to the vertices of H.

Definition 6 A *d-dimensional gray code embedding (dDGC)* of a $l_1 \times l_2 \times \cdots \times l_d$ mesh into a $n = \sum_1^d \log(\overline{l_i})$ dimensional hypercube[1] maps node $v = (v_1, v_2, \ldots, v_d)$ of the mesh to the hypercube slot $\sigma(v) = \text{cat}_{i=1}^{d} G_{\log(\overline{l_i})}(v_i)$. (Where cat is the concatenation of the $\log(\overline{l_i})$-bit values.)

A 1-D mesh is just a Hamiltonian path and thus is embedded with optimum expansion and dilation via a one dimensional gray code embedding. For higher dimension meshes the multi-dimensional gray code does not always use minimum expansion. In our earlier example of a 5×11 mesh a 2-D gray coding uses 3 bits for the first dimension and 4 for the second and thus a hypercube of $2^7 = 128$

[1] We will often want to refer to the highest power of two smaller than X or the smallest power of two larger than X. These two values will be written respectively as $\underline{X}$ and $\overline{X}$.

nodes. But this mesh has only 55 nodes and would fit in a 6-cube of 64 nodes. In general for a $W \times L$ mesh the 2-D gray code will give minimum expansion iff $WL > \underline{W}\overline{L}$.

In fact, for some meshes, dilation 1 requires nonoptimal expansion regardless of the embedding used. For completeness we include the following theorem which has been independently discovered by, among others, [BrSc85,Chan86,HoJo87]:

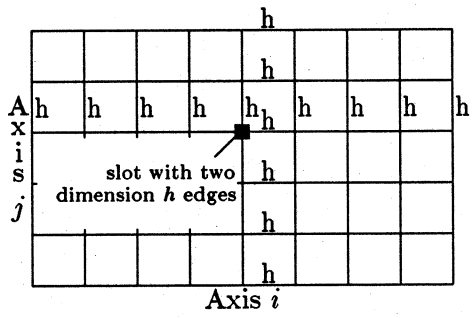

Figure 2: Conflict when Edges in Two Mesh Axes Correspond to Edges in one Hypercube Dimension

Theorem 1 (Meshes in Cubes with Dilation 1)
If an $l_1 \times l_2 \times \cdots \times l_d$ is embedded in an n-cube with dilation 1 then $n \geq \sum_1^d \log(\overline{l_i})$. Furthermore there exists an embedding such that $n = \sum_1^d \log(\overline{l_i})$.

Proof: The crux of the proof is that all hypercube edges along any hypercube dimension must correspond to mesh edges along a single mesh axis. This fact relies on the fact that any two hypercube edges on opposite sides of a 4-cycle must be along the same dimension. (As the cycle is traversed two dimensions are crossed and then the same two dimensions must be crossed back.)

Now consider any two mesh axes i and j (see figure 2). If a hypercube edge along dimension h corresponds to a mesh edge along axis i then a whole 'column'

of mesh edges will correspond to hypercube edges in dimension h since each will be opposite another in a 4-cycle. If some other edge along dimension h corresponds to a mesh edge along axis j a similar 'row' of edges would correspond to dimension h hypercube edges. But the 'row' and 'column' intersect thereby assigning two h dimension hypercube edges to a single hypercube slot which is impossible. Thus all edges in a hypercube dimension correspond to a single mesh axis.

Thus each mesh axis of length l_i is assigned at least $\log(\overline{l_i})$ hypercube dimensions disjoint from those used for other mesh axes. Clearly the entire embedding will require at least $\sum_1^d \log(\overline{l_i})$ hypercube dimensions.

When a gray code is used along each mesh axis (for example a dDGC) exactly $\log(\overline{l_i})$ hypercube dimensions are used for a mesh axis of length l_i and $n = \sum_1^d \log(\overline{l_i})$.

Corollary 1 Infinitely many d-dimensional meshes require expansion 2^{d-1}.

Proof:

If $\forall i, l_i = 2^d + 1$ then $\sum_1^d \log(\overline{l_i}) = d(d+1)$. Thus a $2^{d(d+1)}$ node hypercube is used. But $\prod l_i = (2^d+1)^d \leq 2^{d^2+1}$ so a 2^{d^2+1} node hypercube would be large enough and the expansion is 2^{d-1}.∎

3 Combinatorial Lemmas

Definition 7 A *reflected pebbling* is a pebbling of a hypercube according to the following rules. Choose a gray code sequence, G, for the hypercube. Initially place a pebble on each of the first p nodes of the hypercube according to the gray code ordering defined by G. On the ith round move pebbles across dimension $\tilde{G}(i-1)$. Thus on round i whether a pebble is placed on a given node depends on the prior round's pebbling of the node's image across dimension $\tilde{G}(i-1)$. This image node is called the node's *partner* for round i.

Definition 8 Any subcube spanned by the smallest i dimensions is a *primary i-subcube.* There are 2^{n-i} primary i-subcubes in an n dimensional hypercube.

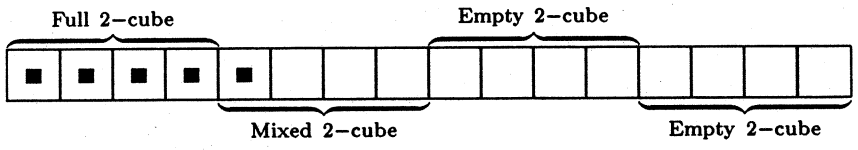

Figure 3: Initial Pebbling of a 4-cube

Lemma 2 (Reflected Pebbling Lemma)
Given an n-dimensional hypercube and a reflected pebbling of p pebbles:

1. *If $p = 2^i$ (the pebbles fill a primary i-subcube) then every node is pebbled exactly once after 2^{n-i} rounds.*
2. *After each round the maximum difference in the number of times two neighboring nodes in the hypercube have been pebbled is* 1.
3. *After each round the maximum difference in the number of times any two nodes have been pebbled is n.*

Proof:

1. Any dimension higher than i maps each primary i-subcube onto some other primary i-subcube, whereas any dimension smaller than or equal to i maps each primary i-subcube onto itself. The sequence of $2^{n-i}-1$ dimensions of $\tilde{G}$ prior to the first occurrence of dimension i form a complete gray code of the dimensions higher than i. Therefore the subcube containing the pebbles must have been mapped to every other i-subcube exactly once after 2^{n-i} rounds.

2. First consider two nodes which differ in the highest dimension. On even numbered rounds $\tilde{G}$ specifies the highest dimension and thus the two nodes are partners. Thus each node is pebbled on the even round iff the other was pebbled on the odd round and they must have been pebbled the same number of times after even rounds. The odd rounds can then make them differ by at most one.

   Next consider nodes which differ in a lower dimension, i, and the primary i-subcube containing them. After the first round if the primary i-subcubes are examined in gray code order there are 0 or more subcubes filled with pebbles followed by 0 or 1 partially filled subcube followed by empty subcubes. (See figure 3). Since the partners of one primary subcube are always another primary subcube there will always be at most one subcube which is partially pebbled. In a round in which one of the full or empty subcubes is mapped to our subcube both nodes are pebbled or both are not pebbled respectively. Thus any differences in pebble numbers are the same as if only the partially pebbled primary i-subcube were pebbled at the start. Furthermore, when the dimensions larger than i are ignored in $\tilde{G}$ the remaining sequence in $\tilde{G}$ induces a reflected pebbling on this primary i-subcube. Thus examining only the rounds on which the partially pebbled subcube is mapped to the subcube containing the two nodes yields a new reflected pebbling on the subcube. In this pebbling the two nodes differ in the highest dimension so the previous case applies.

3. The original pebbles can be broken up into at most n sets by choosing first all the pebbles in the largest i-subcube with $2^i \leq p$ and then successively the the pebbles in the largest i-subcube with 2^i less than or equal to the remaining number of pebbles. Since at most one subcube

of each size is chosen there are at most n chosen sets. Considered individually each set and the subcube containing it fulfills the conditions of part 1 and thus causes a pebble to be placed on each node before causing a second pebble on any node. Thus each is responsible for a difference of at most one in the number of pebbles placed on any two nodes. Since there are at most n sets there can be at most a discrepancy of n.∎

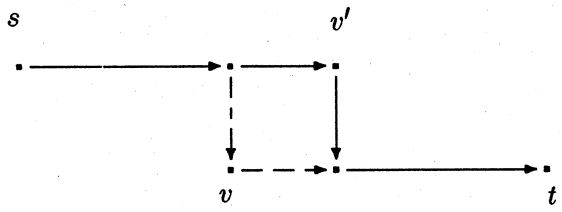

Figure 4: Alternate Shortest Path Through v

Lemma 3 *If a shortest path, P, between nodes s and t in a hypercube contains two neighbors of v then v is on a shortest path between s and t.*

Proof: The two neighbors of v are a distance 2 apart. (See figure 4). Thus since P is a shortest path it must take distance 2 to get from one neighbor to the other. Then the path from s to t which follows P to the first neighbor, visits v, visits the second neighbor, and then follows P to s has the same length as P and is also a shortest path.∎

Lemma 4 *In an n dimensional hypercube forall sets* $\{h_1, \ldots, h_{n-1}\}$ *of* $n-1$ home *nodes, there exist* $n-1$ *pairwise disjoint pairs of nodes* $p_1, \ldots, p_{n-1}$ *such that both nodes in* p_i *are neighbors of* h_i *and no home node is included in any pair.*

Proof: We start with the fact that nodes a distance 2 apart in the hypercube share 2 common neighbors while nodes any other distance apart share no neighbors. Then we show that sequentially for $i = 1$ to $n-1$, h_i can claim two neighbors as pair p_i while ensuring that p_i does not contain nodes which are both neighbors of h_j for $j > i$.

Inductively for each i we show that there are more possible pairs to choose from than pairs which are disqualified by containing two neighbors of some $h_j, j > i$.

In the hypercube h_1 has n neighbors. Suppose x of its neighbors are in fact home nodes and thus cannot be in p_1. There still remain $\binom{n-x}{2}$ possible pairs of neighbors from which to choose p_1. The x home nodes which are neighbors of h_1 cannot share neighbors with h_1. The remaining $n-x-2$ home nodes could each share some one pair of nodes with h_1. Since $n - x \geq 2$, $\binom{n-x}{2} > n-x-2$ and thus the number of possible pairs is greater than the number which contain two neighbors and a pair can be chosen.

Now assume the first i home nodes have claimed their pairs under the induction hypothesis. Then suppose h_{i+1} has x home neighbors and $y \leq i$ neighbors were claimed by the previous i home nodes. Thus there are at least $n-x-i$ neighbors from which to choose. Each of the $n-x-i-2$ home nodes which are not neighbors and have index greater than i can disqualify one pair. Since $\binom{n-x-i}{2} > n-x-i-2$ there remains at least one pair of neighbors to become p_{i+1}.∎

Lemma 5 (Disjoint Path Lemma) *Given an n-dimensional hypercube,* $p \leq n$ *sources, and p sinks there exist n node disjoint paths from sources to sinks.*

Proof: The proof is by induction on the number of dimensions in the hypercube.

When $n = 2$ the lemma is true by inspection.

Assuming the lemma is true for $n - 1$ we show it true for n. Choose the source/sink pair, (s, t), with minimum Hamming distance between s and t. Since there are at least 2 sources the distance s to t must be less than n. Thus there exists a half cube containing both s and t. Choose any such half cube and call it the upper half cube, the dimension not in the half cube's span i, and its image across dimension i the lower half cube.

Connect s to t via any shortest path. Note that since s and t are a closest pair no shortest path between s and t can pass through any other source or sink.

For each source or sink other than s and t in the upper half cube create a surrogate source or sink at the node's image across dimension i or at the image of one of its neighbors. By the induction hypothesis paths can be found from the sources originally in the lower half cube and the surrogate sources to the original sinks and surrogate sinks. These paths can then be extended from the surrogates to the original source via the dimension i edge at the surrogate and possibly one edge in the upper half cube.

Thus we must guarantee that no surrogate source is placed on an original source, that no surrogate sink is placed on an original sink, and that the possible extra path node in the upper half cube does not lie on the path between s and t.

When a source's (or sink's respectively) image across dimension i is not a source (sink) the surrogate can be placed on the image and the path in the upper subcube will contain only the source (sink) itself and thus will not intersect the path from s to t.

When the sources's image is occupied by a source we will displace the surrogate to the image of a neighbor. The existence of a neighbor which does not lie on the s to t path relies on two facts:

By lemma 4 each of the possibly $n - 2$ sources can pick two neighbors in the $n - 1$ dimensional half cube as potential surrogates disjoint from those chosen by

the other sources. (s needs no surrogate and any source looking for a surrogate neighbor represents two sources, one from each half cube so at most $n-2$ sources need surrogate neighbors.)

Since by lemma 3, two neighbors of a source cannot both be on the path between s and t at least one of the 2 neighbors chosen is an acceptable surrogate.

Thus every source and sink can be assigned a surrogate either at its image across dimension i or at the image of a neighbor which does not lie on the s to t path.

Therefore the $n-1$ paths extended from the lower subcube plus the path between s and t form the desired set of n paths.∎

4 The Two-phase Embedding Strategy

In this section we will discuss only two dimensional meshes. We will also concentrate only on those meshes which are not embedded with minimum expansion by 2DGC so throughout the section M will be a $W \times L$ mesh where $WL \leq \underline{W}\overline{L}$ and $W \leq L$.

Once the requirement of dilation 1 is removed there is no clear strategy comparable to 2DGC. We use a two phase strategy which allows us, nonetheless, to use some of the properties of the 2DGC. First the guest mesh is embedded into an intermediate mesh which is a convenient size for 2DGC and then the intermediate is embedded into the cube.

The intermediate mesh, M_1, which we will use will be $W_1 \times L_1 = \underline{W} \times \overline{L}$. Clearly the 2DGC gives a bijective, dilation 1 map from M_1 to a hypercube. Thus if we can embed M into M_1 with dilation δ and optimum expansion we can embed M into a hypercube with the same dilation and expansion by composing the two embeddings.

Several authors have examined the problem of embedding one mesh into another mesh [AlRo82,KoAt86]. Some of their techniques are of use to us. In particular we will make use of line compression.

4.1 Line Compression

The idea of line compression (LC) is to take one W-node column of the guest mesh and compress it into $\lceil W/W_1 \rceil$ W_1-node columns of the host mesh. In general, however, LC can cause row neighbors to be greatly separated. In fact the distance between row neighbors equals the number of host columns needed for one guest column. However, $\underline{W}$ is always by definition at least half W. Thus only two columns of M_1 will be needed for each column of M. We will call line compression into two columns LC2.

Thus our basic building block will be an embedding of one guest column into two host-mesh-sized columns. (See figure 5) Let $D = W - W_1$ and $S = W_1 - D$. Then each block will have D rows in which two columns are filled and S rows in which a single column is filled. If we think of these two column blocks as new intermediates then our remaining task is split into:

1. How is a column of M mapped to a block? Which of the rows of the block contain two elements and which one, etc. We will call those rows containing two elements *doublets* and those with only one element *singlets*.

2. How is a block mapped into M_1?

In deciding how to solve these new tasks we will want to keep track of three quantities:

1. What is the maximum dilation within a block? This will correspond to the dilation of edges between adjacent rows of M.

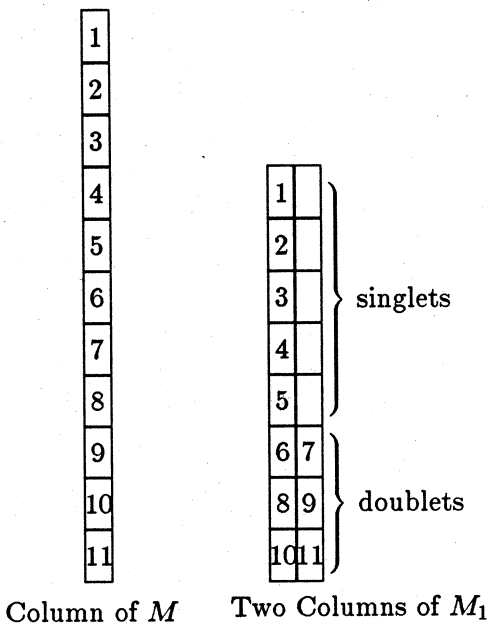

Figure 5: One Column of LC2

2. What is the maximum dilation of edges between adjacent blocks? This will correspond to the dilation of edges between adjacent columns of M.

3. Were we able to fit all the blocks into M_1? Were we forced to map two nodes of M onto one node of M_1 and if so how often did we do so?

4.2 Reflection Method Embedding

The Reflection Method Embedding (RE) will utilize the fact that the 2DGC embedding of M_1 into the hypercube maps columns and rows of M_1 into subcubes. Thus the edges of the subcubes can be used to help spread the doublets out evenly across rows without spoiling the dilation.

The embedding can be visualized as follows. First map each column of M into two columns of M_1 so that dilation within the column is small. If the same map is used for each column then doublets will be mapped to the same rows by each column and thus $2 \cdot L$ columns will be needed. Since $L_1 \leq 2L$ this will often be too many columns. However, if the doublets were spread evenly across the rows we might be able to pack the blocks to the left thereby saving columns.

The dispersal of the doublets is achieved by reflecting each block around a hypercube dimension. The sequence of dimensions used for reflection follow a flexed gray code and thus the Pebbling Lemma gives us bounds on how uniformly the doublets have been allocated.

Reflection Method A $W \times L$ mesh, M, is embedded into a $\overline{WL}$ node hypercube via a $\underline{W} \times \overline{L}$ intermediate intermediate mesh M_1 as follows: (See figure 6)

Let $\tilde{G}(i) =$ be the ith element in the flexed gray code $\tilde{G}_{\log(\underline{W})}$ and $S = 2\underline{W} - W$.

1. Use LC2 embedding for each column of nodes of M into two successive column slots of M_1.

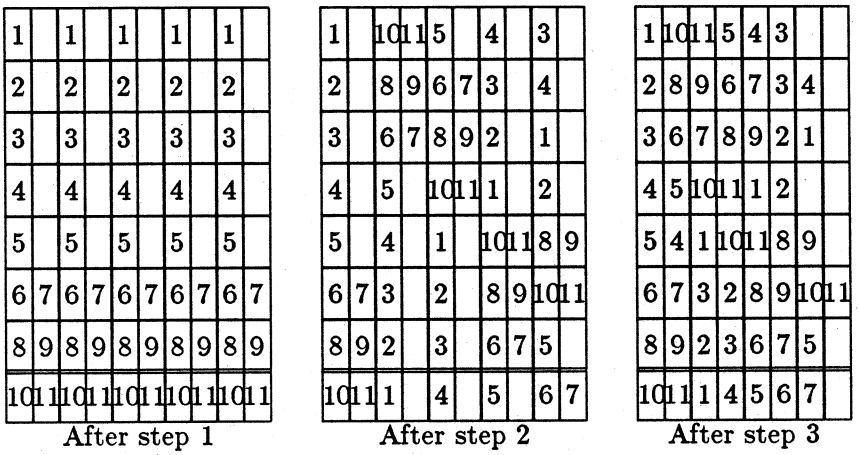

Figure 6: The Reflection Method Embedding

For all $(r,c), 1 \leq r \leq W, 1 \leq c \leq L$, map node (r,c) of M to slot (r_1, c_1) of M_1 as follows:

$$r_1 = \begin{cases} r & \text{if } r \leq S \\ S + \lceil \frac{r-S}{2} \rceil & \text{otherwise} \end{cases}$$

$$c_1 = \begin{cases} 2c-1 & \text{if } r \leq S \text{ or } r - S \text{ is odd} \\ 2c & \text{otherwise} \end{cases}$$

2. Reflect each block about the appropriate hypercube dimension.

   $\forall c > 1, \forall r$ change r_1 to $r_1 \oplus \tilde{G}(c-1)$.

3. Pack all nodes to the left.

   Define $\mathrm{DB}(r', c)$ = the number of doublets placed in row r' of M_1 after $c-1$ blocks have been placed.

   $\forall c > 1, \forall r$ change c_1 to $c + \mathrm{DB}(r_1, c)$.

4. Reflect any nodes assigned slots outside of M_1 back into M_1. This may cause more than one node to be assigned the same slot. (Strictly speaking this makes the Reflection Method not an embedding but in the next section this will be rectified.)

   $\forall(r,c)$ such that $c_1 > \overline{L}$ change c_1 to $c_1 - \overline{L}$.

5. If either node in a doublet is now a distance greater than 3 from its neighbor in the previous column swap the slots of the two nodes in the doublet.

Lemma 6

For any bit position $1 \leq b \leq \log(\underline{W})$, and indices $1 \leq r_1 \leq \underline{W}$, $1 \leq c \leq L$

$$|\mathrm{DB}(r_1, c) - \mathrm{DB}(r_1 \oplus b, c)| \leq 1$$

Proof: Apply the Reflection pebbling lemma with the rows receiving doublets corresponding to the nodes receiving pebbles.∎

Corollary 2 If nodes u and v are neighbors within a column of the mesh then their slots are at most a distance 3 apart in the hypercube.
$\mathrm{Dist}(\mathrm{RE}(r+1,c), \mathrm{RE}(r,c)) \leq 3$.

Proof: Starting at $\mathrm{RE}(r,c)$ the row for $\mathrm{RE}(r+1,c)$ is distance 1 away since the row assigned the two nodes differs in one dimension of $\tilde{G}$. That the distance in this row to $\mathrm{RE}(r+1,c)$ is less than two is clear from the following observations along with Lemma 6. The row $r+1$ node is mapped to column $(c-1)+\mathrm{DB}(r+1,c)+1$ or $(c-1)+\mathrm{DB}(r+1,c)+2$ and the row r node is mapped to $(c-1)+\mathrm{DB}(r,c)+1$ or $(c-1)+\mathrm{DB}(r,c)+2$.∎

Corollary 3 If nodes u and v are neighbors within a row of the mesh then their slots are at most a distance 3 apart in the hypercube.
$\text{Dist}(\text{RE}(r, c+1), \text{RE}(r, c)) \leq 3$.

Proof: Once again an edge exists to the correct row since the columns differ by one gray code dimension. The distance within the row is less than two in each of the following two cases. If row r corresponds to a singlet then the difference in columns will be $c + \text{DB}(r_1 \oplus b, c+1) + 1 - ((c-1) + \text{DB}(r_1, c) + 1) \leq 2$. If it corresponds to a doublet then column c node falls in column $(c-1)+\text{DB}(r_1, c)+1$ or $(c-1)+\text{DB}(r_1, c)+2$ and the column $c+1$ node falls column $c+\text{DB}(r' \oplus b, (c+1))$ or $c+\text{DB}(r' \oplus b, (c+1))+2$. The maximum distance along the intermediate mesh row between placements is thus 4. However, by lemma 1 there is always a length 2 path in the hypercube between any two slots a distance 2^i apart on the gray code ordering. Thus if the distances along the row would be 3 and 3 or 2 and 4 depending on the order of the elements in the doublet the latter is chosen in rule 5 and the hypercube distance is 2 for both. ∎

Lemma 7 Only the last $\log \underline{W'} - 1$ columns can contain slots to which two nodes have been mapped.
$|\text{DB}(r_1, c) - \text{DB}(r_2, c)| \leq \log(\underline{W})$.

Proof: Again a simple application of the Reflection pebbling lemma. Before step 4 the number of columns receiving nodes in any row equals the number of singlets placed in the row (which for all rows equals the number of columns in the guest mesh) plus the number of doublets placed in the row. But by the lemma the number of doublets varies by at most $\log \underline{W}$ and thus the difference between the number of the last column in which every row has a guest node embedded and the first column in which no row has a guest node embedded is also at most $\log \underline{W}$. If the end of the intermediate mesh were at a column in which every slot received a

node and more nodes remained then the guest mesh would have more nodes than the intermediate mesh contradicting our assumption. Thus the doubling back must begin in the area where some slots did not receive nodes, leaving fewer than $\log \underline{W}$ columns to reflect back and cause two nodes to be mapped to a single slot. Therefore only the last fewer than $\log \underline{W}$ columns can contain slots to which two nodes have been mapped.∎

Theorem 2 *The Reflection Method Embedding of a $W \times L$ mesh to a $\log \underline{W}\underline{L}$ dimension hypercube has dilation 3 and causes two nodes to be mapped to one hypercube slot only in a $\log(\underline{W}) + \log(\log(\underline{W}))$ dimensional subcube.*

This next section describes the percolation technique which allows the slots containing two guest nodes to offload one node to the slots containing no guest nodes. Thus the Reflection Method becomes a 1-1 embedding.

5 Percolation

Sometimes, as in the Reflection Method above, a mapping has small dilation but leaves some slots containing two nodes and others containing none. One way of resolving these conflicts is to *percolate* one of the nodes from the two node slot over to an empty slot. The defining property of a percolation is that no node is moved a distance greater than 1 in the hypercube. In this way the dilation is increased by at most 2. (If two nodes were d apart before the percolation and each moves across a different dimension the dilation would be $d + 2$.)

Suppose there is one slot containing two nodes and one slot which is empty. If the slots are neighbors then one of the nodes can be moved from the overfull slot to the empty slot and no node has moved a distance greater than 1. However, if the overfull slot is further than distance 1 from the empty slot one of the nodes in the overfull slot cannot be moved directly to the empty slot. Instead we pick a

simple path between the overfull slot and the empty slot. One of the nodes in the overfull slot is moved to the second slot on the path, the node previously in that slot is moved to the third slot, etc. until the node in the penultimate slot is moved into the empty slot. (See figure 7). Thus each node is moved at most a distance 1.

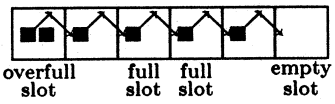

Figure 7: Percolation of a single source

When there are multiple overfull slots (sources) and multiple empty slots (sinks) a set of vertex disjoint paths between sources and sinks is necessary.

Definition 9 A *percolation* of a set of source vertices to a set of sink vertices in a graph is a set of vertex disjoint paths connecting each source vertex to a sink vertex.

Percolation of the Reflection Embedding The percolation will proceed in two phases. (See figure 8). In the first phase we find disjoint paths along the rows of M_1 which connect each source (respectively sink) to an intermediate source (sink) so that the intermediate sources (sinks) are evenly distributed among the columns. We ensure that the paths for the sources pass through slots in one $\underline{W}/2$ column region (call it R_1) of M_1 and those for the sinks pass through slots in a distinct region (call it R_3). In the second phase we find disjoint paths along the columns which connect intermediate sources to intermediate sinks. These paths pass through slots in a third $\underline{W}/2$ column region (call it R_2). Thus every source will be connected to a sink via a path that first travels along a path containing slots in R_1 which takes it to a new column then along a path in R_2 which takes it

to the row of its partner sink and finally along a path in R_3 which takes it to the sink's column and thus to the sink's slot.

We next describe where these regions are and how the disjoint paths are found. The identity of the regions is simple. $W \leq L$ implies $\underline{W} \leq 2\overline{L}$ and thus there are at least four subcubes spanning $\underline{W}/2$ columns. The rightmost four will be used as path regions. Label them from left to right R_1, R_2, R_3, and R_4. Since $\underline{W} \geq 2$ implies $\log \underline{W} \leq \underline{W}/2$ all the sources and sinks originally lie in region R_4. Regions R_4 and R_2 are connected via cube edges to both region R_1 and region R_3.

The first phase's disjoint paths are a little more complicated to describe. Suppose the total number of sources is T. Then we wish to assign sources in region R_4 to intermediate sources in region R_1 so that every column has either $\lfloor 2T/\underline{W} \rfloor$ or $\lceil 2T/\underline{W} \rceil$ intermediate sources. Those columns having the larger number of intermediate sources will be to the left of those having the smaller number. Once the number of intermediate sources in each column is known the pairing to the original sources can be made as follows. Suppose row r contains T_r sources. For each row, r, from top to bottom assign the T_r sources to the T_r columns which have the most as yet unplaced intermediate sources by placing an intermediate source in row r of each column. (Break ties arbitrarily.)

Now consider any single row. There are $T_r \leq \log(\underline{W}) - 1$ intermediate sources in R_1 and the same number of images in R_1 of original sources from R_4 across the cube edges mentioned above. Treating the images as sources and the intermediates as sinks the Disjoint Path Lemma guarantees that disjoint paths exist between the two sets within region R_1. (The original source to intermediate source pairings may not be the same as when we assigned intermediate sources to columns but we do not care.)

The paths for the sinks are found in a similar way except region R_3 is used instead of region R_1 and there may be extra sinks left over at the end of the intermediate sink assignment.

The second phase is now simple. In each column of R_2 treat the images in R_2 of the intermediate sources from R_1 as sources and the images in R_2 of the intermediate sinks from R_3 as sinks and apply the Disjoint Path Lemma to the $\log W$ dimensional cube containing the column.

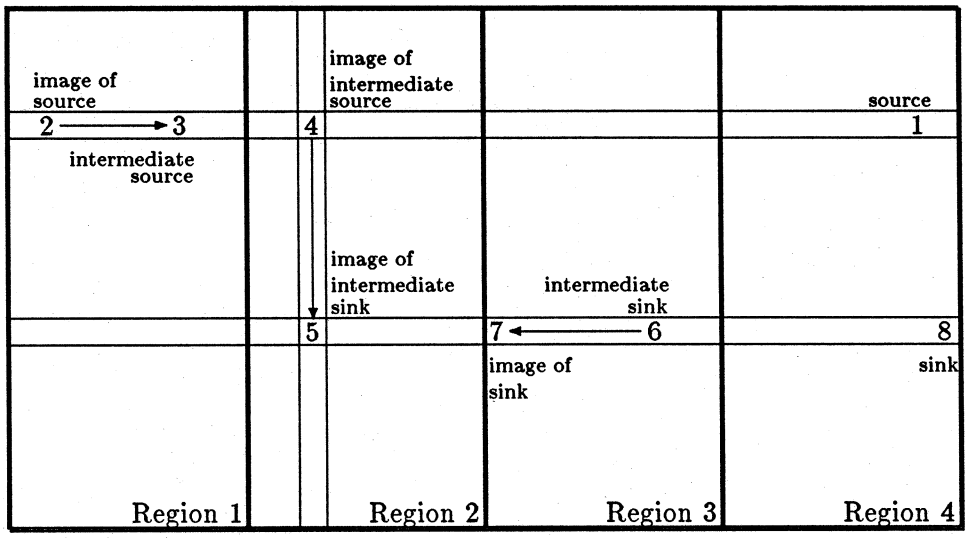

Figure 8: Overview of Percolation

Thus a set of vertex disjoint paths between each overfull slot and an empty slot are created and the percolation adjusts the Reflection Method Embedding to create a 1-1 embedding which maps at most one node of the mesh M onto each slot of the hypercube.

Theorem 3 *The Reflection Method Embedding with Percolation of a $W \times L$ mesh to a $\log W L$ dimension hypercube has dilation 5 and minimum expansion.*

Proof: Consider any two neighboring nodes in the mesh. After the Reflection Method Embedding they are both mapped to hypercube slots which are are most

distance 3 apart. One phase of percolation may push them a further 2 apart. Thus the overall embedding is dilation 5.

Since $\log \underline{W}\underline{L} = \log \overline{W}\overline{L}$ we are using the smallest dimension hypercube having more slots than the mesh has nodes.∎

6 Extensions and Comments

6.1 Improving the Dilation

For many meshes dilation better than dilation 5 is possible. Besides those for which dilation 1 can be achieved by 2DGC there are many meshes which can be embedded with dilation 2. In fact by using another mapping called the Translation Method we have succeeded in embedding with dilation 2 all meshes with both width and length smaller than 66.

It however remains an open question to determine whether all meshes can be embedded with dilation 2 and expansion 1 or whether, in fact, some require larger dilation.

For dilation greater than 1 *load factor* becomes an issue. To calculate load factor a single path in the hypercube is chosen for each edge in the mesh. The load factor is the maximum over all edges of the hypercube of the number of such paths using the edge. A simple choice of paths for the Reflection Embedding is for the path corresponding to the edge between a node and its right or lower mesh neighbor to (if necessary) cross the percolation edge to the start node's pre-percolation slot, cross a hypercube edge to the correct row of the destination's pre-percolation slot, cross row edges to the destination's pre-percolation slot, and then (if necessary) cross the percolation edge to the destination slot. Thus each hypercube edge may be crossed by at most 4 percolation edges (1 for each mesh edge out of the node percolated across it) and either 2 column edges or 4 row edges

for a total of 8. More complicated schemes can reduce this value somewhat.

6.2 Higher Dimensional Meshes and Toruses

As we saw earlier a d-dimensional mesh may use $d-1$ extra dimensions if d-D gray code is used. A first approach to decreasing the expansion is to pair mesh dimensions for which a cube dimension can be saved via our 2-dimensional approach. Using a multidimension gray code to fit the pairs together up to $d/2$ dimensions can be saved at a cost of dilation 5 .

If better expansion is required an extension of the 2-dimensional method would be necessary. This extension could be called Surface-Compression. As with Line Compression the idea is to embed $d-1$ dimension surfaces into two smaller $d-1$ dimension surfaces. All the constructions easily generalize to show that adjacent elements in the $(d-1)$-dimensional meshes and corresponding elements in adjacent submeshes are at most distance 5 apart in the d-D meshes. However, if the $(d-1)$-D meshes already contain length p paths they could be stretched to length $5p$. In other words a straightforward extension of the 2-D method to d dimensions leads to an minimum expansion, 5^{d-1} dilation embedding.

Another open question is whether this bound can be reduced with or without reducing the bound for the 2-dimensional case.

Often wrap-around connections between the last position and the first position in a given row or column are desired. Since the gray codes of the row positions already wrap around the reflection method automatically has column wrap around. The gray codes of the column positions can be 'stretched out' by skipping pairs of high dimension bit flips in order to make the last column close to the first column. Since the stretching occurs in two column increments an extra distance of 1 may occur on the wrap around edges.

The Percolation does not, however, stretch these edges so the overall embedding is still dilation 5.

6.3 Many-to-one Mappings

In some situations the size of the mesh may not be known at the start or the mesh may be so big that several nodes of the mesh will have to be mapped to a single node of the hypercube. In this case the polymorphic arrays of Fiat and Shamir [FiSh84] can be used. A polymorphic array guarantees that if a polymorphic array of size M is used then any array of size $M/\sqrt{5}$ can be dynamically constructed without assigning more than one element of the new array to an element of the polymorphic array and that furthermore for arbitrary sized arrays the number of elements assigned to two different polymorphic array positions will differ by at most $O(\log n)$.

These polymorphic arrays can be easily embedded using our method. Since the dimensions of the polymorphic arrays have Fibonacci and Lucas number sized lengths the 2-D gray code is often inappropriate. For example both the 5×11 and 21×47 versions (two of the four smallest sizes) would require an extra dimension with 2-D gray code.

Acknowledgements

The author is grateful to Sandeep Bhatt for his guidance and clarifications as well as to Josh Benaloh, Ching-Tien Ho, Ruben Michel, and Abhiram Ranade for many helpful discussions.

References

[AlRo82] R. Aleliunas and A. Rosenberg, "On Embedding Rectangular Grids in Square Grids", IEEE Transactions on Computers, V. c-31, 9 (September 1982), 907–913.

[BCLR86] S. Bhatt, F. Chung, T. Leighton, and A. Rosenberg, "Optimal Simulations of Tree Machines", *Proc.* 27th *IEEE Symp. on Foundations of Computer Science*, Toronto, ON (Oct. 1986), 274–282.

[BrSc85] J. Brandenburg and D. Scott, "Embeddings of Communication Trees and Grids into Hypercubes", Intel Scientific Computers Tech Report, 1985.

[Chan86] T.Chan, Personal communication, 1986.

[FiSh84] A. Fiat and A. Shamir, "Polymorphic Arrays: A Novel VLSI Layout for Systolic Computers", *Proc.* 25th *IEEE Symp. on Foundations of Computer Science*, Singer Island, FL (Oct. 1984), 37–45.

[HMR83] J. Hong, K. Mehlhorn, and A. Rosenberg, "Cost Trade-offs in Graph Embeddings, with Applications", JACM, V. 30, 4 (October 1983), 709–728.

[HoJo87] C-T. Ho, and L. Johnsson, "On the Embedding of Meshes in Boolean Cubes", *Proc. IEEE Computer Society Int. Conf. on Parallel Processing*, (1987), 188–191.

[KoAt86] S. Kosaraju and M. Atallah, "Optimal Simulations between Mesh-connected Arrays of Processors", Technical Report, Purdue University, CSD-TR-561.

[Th79] C.C. Thompson, "Area-time Complexity for VLSI", *Proc.* 11th *ACM Symp. on Theory of Computing*, (1979), 81–88.