INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Mapping parallel algorithms into hypercubes

White, William Warren, Ph.D.
The Ohio State University, 1989
MAPPING PARALLEL ALGORITHMS INTO HYPERCUBES

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

William Warren White, B.A., M.S., M.S.

*****

The Ohio State University

1989

Dissertation Committee

Prof. Ten-Hwang Lai

Prof. Richard Parent

Prof. Neelamegam Soundararajan

Adviser’s Approval

Adviser

Department of Computer and Information Science
VITA

September 13, 1957 ...................... Born - Louisville, Kentucky

1979 .............................................. B.A. in Mathematics, The University of Louisville, Louisville, Kentucky

1983 .............................................. M.S. in Mathematics, The Ohio State University, Columbus, Ohio

1985 .............................................. M.S. in Computer and Information Science, The Ohio State University, Columbus, Ohio

PUBLICATIONS


FIELDS OF STUDY

Major Field: Mathematics
   Studies in Combinatorics and Graph Theory, with Thomas Dowling.

Major Field: Computer and Information Science
   Studies in Hypercube Embedding, with Ten-Hwang Lai.
Table of Contents

VITA ........................................................................................................................................ ii
LIST OF FIGURES .................................................................................................................. v
LIST OF TABLES ..................................................................................................................... vii

CHAPTER PAGE
I Introduction 1
  1.1 Embedding into the Hypercube 1
  1.2 Basic Definitions 1 1
  1.3 Overview of Thesis 1 8

II Embedding Quadtrees into the Hypercube 10
  2.1 Preliminary Quadtree Results 11
  2.2 A Unit-Congestion, Optimal-Expansion Embedding 13
  2.3 Algorithm Analysis 17
  2.4 A Unit-Congestion, Unit-Expansion Algorithm 18
  2.5 Algorithm Analysis 22
  2.6 Summary 25

III Embedding Pyramids into the Hypercube 27
  3.1 Preliminary Pyramid Results 28
  3.2 An Optimal-Dilation Embedding 31
    3.2.1 The Algorithm 31
    3.2.2 Correctness 35
    3.2.3 Dilation 41
    3.2.4 Congestion 42
  3.3 A Congestion-Two Embedding 51
    3.3.1 The Algorithm 52
    3.3.2 Correctness 58
    3.3.3 Dilation 59
    3.3.4 Congestion 61
  3.4 Simulation Results 63
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$Q_2$</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>$f_2(Q_2)$ (and its unused Y-extension)</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>The recursive definition of $f_k$</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>$g_1(Q_1), q'_1(Q_1), q''_1(Q_1)$, and subgraph $A_1$</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>$A_{k-1}$</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>The images of $q_{k-1}(Q_{k-1}), q'<em>{k-1}(Q</em>{k-1}), q''<em>{k-1}(Q</em>{k-1})$, and $A_{k-1}$ under $p_{11}, p_{10} \circ \theta_{2k-3}, p_{01} \circ \theta_{2k-2}$, and $p_{00} \circ \theta_{2k-3,2k-2}$</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>$q_k(Q_k), q'_k(Q_k), q''_k(Q_k)$, and $A_k$</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>$P_2$</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>$g_1(P_1)$</td>
<td>31</td>
</tr>
<tr>
<td>10</td>
<td>$P_1(k-1,x_1,x_2)$ and its image under $g_1, g^H_1, g'_1, g''_1$</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>$g_2(P_2)$</td>
<td>36</td>
</tr>
<tr>
<td>12</td>
<td>$h_1(P_1)$</td>
<td>52</td>
</tr>
<tr>
<td>13</td>
<td>The image of $P_1$ under $h'_1$, $h^H_1$, and $h''_1$</td>
<td>55</td>
</tr>
<tr>
<td>14</td>
<td>The recursive definition of $h_k$</td>
<td>57</td>
</tr>
</tbody>
</table>
15  $a_1(P_1)$ and $b_1(P_1)$ .............................................. 69
16  $a_2(P_2)$ ................................................................. 71
17  $b_2(P_2)$ ................................................................. 72
18  Algorithm MAXDEG ....................................................... 85
19  Algorithm MAXDEG/MAXSUBTREE ................................. 87
20  Algorithm MINDIL ......................................................... 89
### List of Tables

1. Embedding $g_k$ Simulation Results - Static Routing .................................. 64
2. Embedding $h_k$ Simulation Results - Static Routing ................................ 65
3. Embedding $g_k$ Simulation Results - Dynamic Routing ............................. 66
4. Embedding $h_k$ Simulation Results - Dynamic Routing ............................. 67
5. MAXDEG Simulation Results ........................................................................ 86
6. MAXDEG/MAXSUBTREE Simulation Results .............................................. 88
7. Maximum Congestion of Greedy Simulations .............................................. 88
8. MINDIL Simulation Results ........................................................................ 90
9. Maximum Congestion of MINDIL Simulations ............................................ 90
10. Tree Partitioning Simulation Results .......................................................... 91
11. Maximum Congestion of Partitioning Simulations ....................................... 91
12. Arbitrary Tree Embedding Simulation Results ........................................... 92
13. Maximum Congestion of Arbitrary Embedding Simulations ........................ 92
14. MAXDEG Simulation Results on General Graphs ......................................... 94
15. Maximum Congestion of MAXDEG Simulations on General Graphs ........... 94
16 MINDIL Simulation Results on General Graphs ........................................ 95
17 Maximum Congestion of MINDIL Simulations on General Graphs . 95
18 Partitioning Simulation Results on General Graphs ......................... 96
19 Maximum Congestion of Partitioning Simulations on General Graphs 96
CHAPTER I

Introduction

In recent years, the hypercube network has achieved a marked popularity in the field of parallel computing. The Caltech Cosmic Cube project [47] demonstrated many of the practical advantages of implementing a hypercube network. Hastad, et al., [24] and Becker and Simon [6] have examined the pronounced fault tolerance displayed by the hypercube. The network's flexibility and efficiency are also reflected in its simple expandability, as well as a high degree of scalability [19]. The hypercube permits fast communication at a reasonable cost.

1.1 Embedding into the Hypercube

Several data transfer algorithms have been devised specifically for the hypercube by Saad and Schultz [44] and Ho and Johnsson [25], but the hypercube has also exhibited a strong propensity for adapting standard algorithms that were designed for more classical data structures and sequential architectures. Such adaptations permit the solutions of substantial problems to make good use of the hypercube's
inherent advantages.

One means by which algorithms designed for other topological structures can be implemented on the hypercube is the graphical embedding of the other network architecture into the hypercube. When such embeddings are efficiently accomplished (i.e., without wasting too many hypercube processors and without seriously modifying the structure of the network to make it “fit” in the hypercube), then the hypercube can effectively implement algorithms that are particularly suited for another network. Berman and Snyder [7] have examined the general problem of mapping a parallel algorithm with one communication structure into a parallel architecture with a different configuration. The problem of determining the existence of an adjacency-preserving embedding of a given graph into the hypercube has been shown to be NP-complete by Krumme, et al., [33]. Wagner and Corneil [50] have proven that this problem is still NP-complete when the source graphs are restricted to trees. Kim and Lai [32] have shown that even if the adjacency-preserving restriction is relaxed, obtaining a unit-congestion embedding of a graph into the hypercube is NP-complete.

Efficient embeddings into the hypercube have been developed for several simple configurations. Rectangular meshes have been embedded by Chan and Saad [12], Ma [36], and Ho and Johnsson [26, 28]. Algorithms for embedding binary trees into hypercubes have been developed by Wu [53], Bhatt and Ipsen [8], and Bhatt, et al. [9]. These embeddings take advantage of the symmetric properties of
the embedded networks and the hypercube. Unfortunately, these structures lend themselves so easily to embedding in the hypercube that the algorithms developed for them contribute little to the development of embedding approaches for more sophisticated structures which do not fit naturally into the hypercube.

In this thesis, we attempt to develop a methodology for embedding more complex types of graphs into the hypercube. We begin by exploring the difficulties associated with two classes of graphs which do not easily lend themselves to hypercube embedding: quadtrees and pyramids. The lessons learned from these nontrivial algorithms are then applied to the evaluation of specific heuristic methods for attacking the intractable problems of embedding general trees and graphs into the hypercube.

The quadtree algorithmic structure has been used extensively in the fields of computer graphics and image processing [5, 22, 30, 46]. This graph's basic structure inhibits easy embedding in the hypercube. As a result, our development of an efficient embedding algorithm for the quadtree gives us greater insight into the overall problem of hypercube embedding.

We have also developed algorithms to efficiently embed a pyramid structure in the hypercube. Several advanced algorithms that naturally lend themselves to the pyramid structure are examined by Stout [49], Miller and Stout [37], and Chang, et al. [13]. These algorithms can be effectively implemented on the hypercube by means of our embeddings.
In [48], Stout mentions a substantially different mapping of the vertices of a pyramid into a hypercube so that both expansion and dilation are optimized. The congestion associated with this mapping is not ascertained, however, since the images of edges are not specified. One of the principal contributions of this thesis is a detailed mathematical analysis of our embedding algorithms. By formally exploring the different aspects of our algorithms' efficiency, we are better able to ascertain the advantages and pitfalls of various embedding methods. In particular, we thoroughly examine the congestion associated with our embeddings, since this measurement is an extremely important gauge of efficiency.

In fact, our emphasis on congestion as an embedding efficiency measurement represents the second major contribution of this thesis. Previous embedding work has stressed the minimization of the dilation to which the source graph is subjected. First-generation hypercubes used a store-and-forward communication technology in which long-distance communications were more likely to cause delays. The development of the newer direct-connect hypercube technologies (e.g., the iPSC/2 [3, 4, 17, 41]), with their reliance on circuit-switching to establish communication between nodes, has drastically reduced the significance of dilation when gauging an embedding algorithm's efficiency. Consequently, we can now focus attention on the more complicated congestion cost gauge. The low congestion values obtained by our quadtree and pyramid embeddings indicate that these algorithms are well-suited for the current generation of hypercubes.
The comparative complexity of the quadtree and pyramid structures results in an evaluation of the efficiency of our embeddings that is more complicated than that required for the simpler embeddings of binary trees and rectangular meshes. The fact that very efficient embeddings do exist for these more sophisticated configurations serves to enhance the reputation of the hypercube as a powerful and versatile network.

In this thesis, we apply our analysis of congestion and our evaluation of the complexities inherent in quadtree and pyramid embedding to the issue of developing heuristics for the intractable problems of embedding general trees and graphs into the hypercube. The formal mathematical techniques which we employ to thoroughly gauge the merits of embeddings of specific classes of graphs cannot be readily used with more general graphs. Instead, we perform simulations of algorithmic operations applied to random graphs which have been embedded into the hypercube by different heuristic algorithms. The statistical results obtained in this way are then analyzed with respect to our more formal results for quadtrees and pyramids. This analysis yields the third major contribution of this thesis: the recommendation of specific heuristic algorithms for embedding general trees and graphs into the hypercube.

Our final major contribution in this thesis is the presentation of a nontraditional suggestion for task allocation on the hypercube: embedding multiple algorithmic structures into a single hypercube without resorting to the relegation of separate
tasks to distinct subcubes. The subcube-assignment technique proves easier to implement [14], but often results in the waste of a significant number of inactive hypercube processors. By strategically embedding several graphs into the hypercube, the utilization of processors can sometimes be optimized. If these embeddings of multiple graphs experience little overlap or mutual congestion, then this novel technique can be implemented without fear of communication impairment.

1.2 Basic Definitions

Define $H_n$ to be the hypercube with $V(H_n) = \{0,1\}^n$ and $E(H_n) = \{(x,y): x, y \in V(H_n) \text{ and } x \text{ and } y \text{ differ in exactly one bit}\}$. For $x \in V(H_n)$, if $1 \leq i \leq n$, let $x^i$ denote the element of $V(H_n)$ differing from $x$ in the $i$-th bit (counting from the right). Similarly, $x^{ij}$ differs from $x$ in the $i$-th and $j$-th bits, and so on.

For $x \in V(H_n)$ and $b \in \{0,1\}$, let $bx$ denote the element of $V(H_{n+1})$ that has $(n+1)$-st bit value $b$ and bit values identical to $x$ in bits 1 through $n$. More generally, for $b_1, b_2, \ldots, b_{i-1}, b_i \in \{0,1\}$, $b_ib_{i-1} \ldots b_2b_1x$ denotes the element of $V(H_{n+i})$ with $(n+i)$-th bit value $b_i$ and bit values identical to $x$ in bits 1 through $n$.

An embedding $h$ of a graph $G$ into a graph $G'$ is a map of the vertices of $G$ into the vertices of $G'$ in a one-to-one fashion, combined with a map of the edges of $G$ into the simple paths of $G'$ so that if $e = (u,v) \in E(G)$, then $h(e)$ is a simple path of $G'$ with endpoints $h(u)$ and $h(v)$.
The routing technique employed in the transmission of messages in current hypercube technologies is static. When we analyze the algorithms in this thesis, we shall assume that the static routing occurs along the path specified by our embedding. When the embedding does not specify paths, we assume a standard bitwise left-to-right static routing from the low-address node to the high-address node. Also, when we consider dynamic routing, we employ the same bitwise left-to-right technique.

We now define three gauges of the cost of a graph embedding $h : G \rightarrow G'$. The expansion of $h$ is the ratio of the size of $V(G')$ to the size of $V(G)$, i.e., $|V(G')|/|V(G)|$. The expansion gauges how much of $G'$ is not directly used in the embedding of $G$; the closer the expansion value is to one, the more efficient the embedding's utilization of $G$.

For $e \in E(G)$, the dilation of $e$ under embedding $h : G \rightarrow G'$ is the length of the path $h(e)$ in $G'$. The maximum dilation of $h$ is

$$Dil(h) = \max_{e \in E(G)} \{\text{length of } h(e)\}. \quad (1)$$

The average dilation of $h$ is

$$\overline{Dil}(h) = \frac{1}{|E(G)|} \sum_{e \in E(G)} \{\text{length of } h(e)\}. \quad (2)$$

Each of these measurements reflects the degree to which the structure of $G$ is "stretched" by $h$. In this case, the closer the maximum (or average) dilation is to one, the smaller the degree of deformation of $G$ by $h$. 
Finally, for $e' \in E(h(G)) \subseteq E(G')$, the congestion of $e'$ is the number of edges in $G$ with images including $e'$, i.e., the congestion of $e'$ is $|\{e \in E(G) : e' \text{ is in path } h(e)\}|$. The maximum congestion of $h$ is

$$Cong(h) = \max_{e' \in E(h(G))} \{\text{congestion of } e' \text{ under } h\};$$

(3)

the average congestion of $h$ is

$$\overline{Cong}(h) = \frac{1}{|E(h(G))|} \sum_{e' \in E(h(G))} \{\text{congestion of } e' \text{ under } h\}. \quad (4)$$

These measurements gauge the traffic flow through the edges of $h(G)$. Once again, congestion values closer to one indicate a more efficient embedding.

Although small values of each of these cost gauges are desired, we will assume here that an optimal expansion value is crucial when embedding pyramids into hypercubes. The exponential size difference between $H_n$ and $H_{n+1}$ is the basis for this assumption. The newer hypercube technologies [41] are not drastically affected by higher values of dilation, but the negative effects of congestion [24] have not been effectively counteracted.

1.3 Overview of Thesis

We now present a brief overview of the contents of this thesis. In Chapter 2, we examine the problem of embedding a complete quadtree into the hypercube. The structure of the quadtree prohibits a unit-dilation, optimal-expansion embedding, but, in view of the advent of the direct-connect hypercube technology, we argue
that our unit-congestion, optimal-expansion embedding is, in fact, optimal. We also introduce an algorithm which embeds three quadtrees into a single hypercube with unit expansion and, significantly, unit congestion. We present this algorithm, which is a significant departure from the traditional subcube-assignment task-allocation technique, as an excellent alternative to that more wasteful approach.

In Chapter 3, we address the more complex problem of embedding the pyramid structure into the hypercube. Obtaining either optimal congestion or optimal dilation while maintaining optimal expansion is presented as a very difficult problem. Two algorithms are developed to handle this problem with varying degrees of success with respect to congestion and dilation. Simulations of the implementations of these algorithms on the pyramid yield significant results regarding the value of low-congestion embeddings. We conclude this chapter with the presentation of a unit-expansion embedding of multiple pyramids into the hypercube.

Applying our previous results to the general problem of embedding trees and graphs into the hypercube, we analyze the performance of two major heuristic embedding approaches in Chapter 4. Greedy methods are shown to be very successful when employed on tree structures. A less structure-dependent partitioning heuristic is shown to be more effective for embedding general graphs.

We conclude in Chapter 5 with a discussion of the main points of this thesis and some suggestions for further research in the areas of hypercube embedding and task allocation.
CHAPTER II

Embedding Quadtrees into the Hypercube

Previous work on the embedding of specific algorithmic structures into the hypercube structure has concentrated on the expansion and dilation of the embedding. The need for optimal dilation is drastically reduced with the new direct-connect hypercube technology, however. Instead, a change in emphasis to the optimization of congestion is called for.

Fortunately, the hypercube embeddings of binary trees [12, 26, 36] and rectangular meshes [8, 9, 53] maintain unit congestion and optimal expansion. Unfortunately, the embeddings of these structures into the hypercube are so straightforward (i.e., both structures are practically subgraphs of the hypercube) that examining them does little to illustrate the change in emphasis on the various embedding efficiency gauges, or to assist in the future development of a general approach to embedding in the hypercube.

In this chapter, we examine the hypercube embedding of a structure which does not conform naturally to the hypercube structure: the complete quadtree. In
addition to the inherent value of such an embedding, which enables us to implement
quadtrees on the hypercube, the approaches used to guarantee unit
congestion and optimal expansion are instructive for the development of general
embedding policies.

2.1 Preliminary Quadtree Results

The complete quadtree of height $k$, $Q_k$, is defined as the graph with vertex set
$V(Q_k) = \{(i, t) : 0 \leq i \leq k, 0 \leq t \leq 4^i - 1\}$ and edge set $E(Q_k) = \{((i, t), (i +
1, t')) : 0 \leq i \leq k - 1, [\frac{t}{4^i}] = t\}$. The vertex $(0, 0)$ is called the root of $Q_k$. For
$0 \leq i \leq k - 1$, each vertex $(i, t)$ is called the parent of its offspring, the vertices
$(i + 1, 4t), (i + 1, 4t + 1), (i + 1, 4t + 2)$, and $(i + 1, 4t + 3)$. Finally, each vertex $(k, t)$
is called a leaf of $Q_k$. Note that $|V(Q_k)| = \frac{1}{3}(4^{k+1} - 1)$ and $|E(Q_k)| = \frac{3}{2}(4^k - 1)$.

Figure 1 illustrates $Q_2$.

For $k \geq 1$, $2^{2k} < \frac{1}{3}(4^{k+1} - 1) < 2^{2k+1}$, indicating that any hypercube into
which $Q_k$ is embedded must have at least $2k + 1$ dimensions. The expansion of
any embedding $h : Q_k \rightarrow H_{2k+1}$ would be approximately 1.5, indicating that such
an embedding would effectively use two-thirds of the hypercube processors.

The maximum vertex degree of \( Q_1 \) is 4, so an optimal-expansion embedding into \( H_3 \) is impossible with unit congestion. However, for \( k \geq 2 \), the maximum vertex degree of \( Q_k \) is 5, yielding no contradiction to the possibility of a unit-congestion embedding into \( H_{2k+1} \).

Before presenting our first algorithm, we present a significant result regarding dilation.

**Lemma 2.1.1** For \( k \geq 2 \), if \( h : Q_k \rightarrow H_{2k+1} \) is an embedding, then \( \text{Dil}(h) > 1 \).

**Proof.** We model our proof on that of a similar result for binary trees by Bhatt and Ipsen [8].

If \( \text{Dil}(h) = 1 \), then \( Q_k \) would be a subgraph of \( H_{2k+1} \) and, in addition to the edge connecting it to its parent, each leaf would be incident with \( 2k \) edges of the hypercube. Thus, \( 2k \cdot 4^k \) edges exist in \( E(H_{2k+1}) - E(Q_k) \) such that each such edge is incident with a leaf of \( Q_k \). Let us call the other endpoint of each of these edges a receptor (following the terminology of Bhatt and Ipsen).

Since \( H_{2k+1} \) has no odd cycles [45], no leaf or grandparent of a leaf may be a receptor. Thus, the number of receptors is at most \( 2^{2k+1} - 4^k - 4^{k-2} \).

The number of receptors that can accept \( 2k + 1 \) of the edges emanating from the leaves is \( |V(H_{2k+1})| - |V(Q_k)| = \frac{1}{3}(2^{2k+1} + 1) \). The root is the only potential receptor which can accept \( 2k - 3 \) edges. Finally, the remaining \( \frac{1}{3}(13 \cdot 4^{k-2} - 4) \)
receptors can accept $2k - 4$ edges.

Therefore, the number of edges that can be accepted by receptors is at most

$$\frac{1}{3}(2^{2k+1} + 1)(2k + 1) + (1)(2k - 3) + \frac{1}{3}(13 \cdot 4^{k-2} - 4)(2k - 1) = \left(\frac{15}{8}k - \frac{5}{12}\right)4^k - \frac{3}{4}k + 2.$$ 

A simple induction shows that this number is strictly less than $2k \cdot 4^k$. So, the number of edges which can be accepted by receptors is less than the number of edges incident to the leaves, contradicting the assumption that $Q_k$ is a subgraph of $H_{2k+1}$.

Note that this lemma indicates that a unit-dilation, optimal-expansion embedding of a complete quadtree into a hypercube is impossible. Consequently, the earlier approaches to graph embeddings would dictate the development of a non-optimal algorithm to embed $Q_k$. The advent of the iPSC/2 technology allows us to deemphasize dilation as a consideration. Furthermore, our multiple-embedding approach to task allocation will give us a new perspective on the expansion gauge.

2.2 A Unit-Congestion, Optimal-Expansion Embedding

We now present a recursive algorithm to embed $Q_k$ in $H_{2k+1}$ with unit congestion. Define embedding $f_2 : Q_2 \rightarrow H_5$ as illustrated in Figure 2. (For clarity, our illustrations of embedded graphs will take the shape of their original pre-images, with vertices labelled using hypercube binary representations.) The congestion of $f_2$ is obviously 1 and its dilation is 2. Note that there is a Y-shaped extension of the embedded quadtree which consists of three vertices in $V(H_5) - f_2(V(Q_2))$ and
three edges in $E(H_5) - E(f_2(Q_2))$; this extension will be used to implement the bottom-up recursion of our algorithm.

Assume now that $f_{k-1} : Q_{k-1} \to H_{2k-1}$ is a unit-congestion embedding with $\alpha = f_{k-1}(0,0)$. Also, assume that $\alpha^{2k-3}, \alpha^{2k-3,2k-2}, \alpha^{2k-3,2k-1} \in V(H_{2k-1}) - f_{k-1}(V(Q_{k-1}))$ and $(\alpha, \alpha^{2k-3}), (\alpha^{2k-3}, \alpha^{2k-3,2k-2}), (\alpha^{2k-3}, \alpha^{2k-3,2k-1}) \in E(H_{2k-1}) - E(f_{k-1}(Q_{k-1}))$.

Let $Q_{k-1}(1,t)$ denote the subtree of $Q_k$ with height $k - 1$ and root $(1,t)$. For $0 \leq t \leq 3$, define $\phi_t : Q_{k-1}(1,t) \to Q_{k-1}$ to be the “natural” embedding which maps vertex $(i,t')$ to $(i - 1, t' - t \cdot 4^{i-1})$. Finally, for any two binary values $b_1,b_2$, define the prefix function $p_{b_2b_1} : H_{2k-1} \to H_{2k+1}$ so that $p_{b_2b_1}(x) = b_2b_1x$ and $p_{b_2b_1}(x,x') = (b_2b_1x, b_2b_1x')$ for any vertex $x$ and edge $(x,x')$ in $H_{2k-1}$.

Using this notation, we may present our recursive algorithm for the embedding $f_k : Q_k \to H_{2k+1}$:
Algorithm

1. If \( k = 2 \), embed \( Q_k \) in \( H_{2k+1} \) using \( f_2 \).

2. If \( k > 2 \), use \( f_{k-1} \) to define \( f_k \) as follows:

   a) For each vertex \((1, t)\), embed \( Q_{k-1}(1, t) \) in \( H_{2k+1} \) using the mapping:

      \[
      \begin{align*}
      p_{00} & \circ f_{k-1} \circ \phi_0 & \text{if } t = 0, \\
      p_{01} & \circ f_{k-1} \circ \phi_1 & \text{if } t = 1, \\
      p_{10} & \circ f_{k-1} \circ \phi_2 & \text{if } t = 2, \\
      p_{11} & \circ f_{k-1} \circ \phi_3 & \text{if } t = 3.
      \end{align*}
      \]

   b) Embed the root \((0, 0)\) and its incident edges so that:

      \[
      \begin{align*}
      f_k(0, 0) & = 00\alpha^{2k-3}, \\
      f_k((0, 0), (1, 0)) & = (00\alpha^{2k-3}, 00\alpha), \\
      f_k((0, 0), (1, 1)) & = (00\alpha^{2k-3}, 01\alpha^{2k-3}, 01\alpha), \\
      f_k((0, 0), (1, 2)) & = (00\alpha^{2k-3}, 10\alpha^{2k-3}, 10\alpha), \\
      f_k((0, 0), (1, 3)) & = (00\alpha^{2k-3}, 00\alpha^{2k-3,2k-2}, 10\alpha^{2k-3,2k-2}, \\
      & \quad 11\alpha^{2k-3,2k-2}, 11\alpha^{2k-3}, 11\alpha).
      \end{align*}
      \]

The algorithm is illustrated in Figure 3. Note that the vertices \(00\alpha^{2k-3,2k-1}, 01\alpha^{2k-3,2k-1}, 10\alpha^{2k-3,2k-1} \in V(H_{2k+1}) - f_k(V(Q_k)) \) and that the edges \((00\alpha^{2k-3}, 00\alpha^{2k-3,2k-1}), (00\alpha^{2k-3,2k-1}, 01\alpha^{2k-3,2k-1}), (00\alpha^{2k-3,2k-1}, 10\alpha^{2k-3,2k-1}) \in E(H_{2k+1})\).
Figure 3: The recursive definition of $f_k$
$E(f_k(Q_k))$, indicating that the Y-shaped extension of the embedded quadtree continues recursively. This verifies the correctness of our algorithm.

### 2.3 Algorithm Analysis

The recursive nature of our algorithm allows us to easily verify the following theorems.

**Theorem 2.3.1** $\text{Dil}(f_k) = 5$. $\text{Dil}(f_k) = (47 \cdot 4^{k-1} - 20)/(2 \cdot 4^{k+1} - 8) \to \frac{47}{32}$ as $k \to \infty$.

**Proof.** A simple induction shows that exactly $3 \cdot 4^{k-1} + \frac{1}{3}(4^{k-2} - 1)$ edges receive dilation one under $f_k$; exactly $2 \cdot 4^{k-1} + \frac{2}{3}(4^{k-2} - 1)$ edges receive dilation two; and exactly $\frac{1}{3}(4^{k-2} - 1)$ edges receive dilation five. □

By sacrificing low dilation at each non-leaf node, our algorithm separates offspring to a sufficient extent to prevent problems in traffic flow, as indicated by the following theorem.

**Theorem 2.3.2** $\text{Cong}(f_k) = 1$.

**Proof.** The distinct two-bit prefixes of the images of different $Q_{k-1}(1,t)$ subtrees guarantee that their edge sets are disjoint. The nature of the Y-extension of $f_{k-1}(Q_{k-1})$ prevents any intersection between the edge sets of any $f_k(Q_{k-1}(1,t))$ and the images of the edges adjacent to the root $(0,0)$. Our result then follows by induction. □
Thus, $f_k$ embeds $Q_k$ into $H_{2k+1}$ with optimal expansion, unit congestion, and small dilation. It is worthwhile to note that a variation of our algorithm using complements allows us to embed an additional height-$(k - 1)$ quadtree into $H_{2k+1}$ without increasing congestion or dilation. Embedding both $Q_k$ and $Q_{k-1}$ into $H_{2k+1}$ reduces expansion from 1.5 to 1.2.

### 2.4 A Unit-Congestion, Unit-Expansion Algorithm

To achieve both unit congestion and unit expansion, we now present an algorithm which embeds three copies of $Q_k$ into $H_{2k+2}$ without using any hypercube edge more than once. Such an approach represents a significant departure from previous task allocation strategies on the hypercube [14], which rely on subcube assignments for separate tasks. The unit congestion of our algorithm prevents the disjoint quadtrees from inhibiting each other's performance, while the unit expansion of our algorithm is a marked improvement over the subcube approach to task allocation.

Define three embeddings $q_1, q'_1, q''_1$ of $Q_1$ into $H_4$ as illustrated in Figure 4. Note that $q_1$ has dilation 2, while $q'_1$ and $q''_1$ each have dilation 1. Each embedding has congestion 1, and no element of $E(H_4)$ is used in distinct embeddings. Furthermore, $A_1$, the subgraph of $H_4$ indicated in Figure 4, has the property that none of its edges occur in $q_1(Q_1), q'_1(Q_1)$, or $q''_1(Q_1)$, and its apex 1111 is the only vertex of $H_4$ that does not occur in those images. The existence of this subgraph will prove useful in our recursion.
Assume now that $q_{k-1}, q'_{k-1},$ and $q''_{k-1}$ are three unit-congestion embeddings of $Q_{k-1}$ into $H_{2k}$ such that $q_{k-1}(0,0) = 0000(1)^{2k-4}$, $q'_{k-1}(0,0) = 0110(1)^{2k-4}$, and $q''_{k-1}(0,0) = 1001(1)^{2k-4}$. Also, assume that $1111(1)^{2k-4}$ is the only vertex of $H_{2k}$ not occurring in $q_{k-1}(Q_{k-1})$, $q'_{k-1}(Q_{k-1})$, or $q''_{k-1}(Q_{k-1})$, and that none of the edges of $A_{k-1}$, the subgraph of $H_{2k}$ illustrated in Figure 5, occur in these images.

For $\{i_1, i_2, \ldots, i_j\} \subseteq \{1, 2, \ldots, 2k\}$, define embedding $\theta_{i_1, i_2, \ldots, i_j} : H_{2k} \rightarrow H_{2k}$ so that $\theta_{i_1, i_2, \ldots, i_j}(x) = x^{i_1, i_2, \ldots, i_j}$ and $\theta_{i_1, i_2, \ldots, i_j}(x, y) = (x^{i_1, i_2, \ldots, i_j}, y^{i_1, i_2, \ldots, i_j})$ for any vertex $x$ and edge $(x, y)$ in $H_{2k}$. For any two binary values $b_1, b_2$, define the prefix function $p_{b_1 b_2} : H_{2k} \rightarrow H_{2k+2}$ in the same fashion as before. Also, recall the definition of the “natural” embeddings $\phi_t$, $0 \leq t \leq 3$, from the previous sections.
Using this notation, we now present our recursive algorithm for the three embeddings $q_k$, $q'_k$, and $q''_k$ from $Q_k$ to $H_{2k+2}$:

**Algorithm**

1. If $k = 1$, use $q_1, q'_1$, and $q''_1$.

2. If $k > 1$, use $q_{k-1}, q'_{k-1}$, and $q''_{k-1}$ to define $q_k, q'_k$, and $q''_k$ as follows:

   a) On $Q_{k-1}(1,0)$, $q_k = p_{01} \circ \theta_{2k-2} \circ q'_{k-1} \circ \phi_0$.

   On $Q_{k-1}(1,1)$, $q_k = p_{10} \circ \theta_{2k-3} \circ q''_{k-1} \circ \phi_1$.

   On $Q_{k-1}(1,2)$, $q_k = p_{01} \circ \theta_{2k-2} \circ q''_{k-1} \circ \phi_2$.

   On $Q_{k-1}(1,3)$, $q_k = p_{01} \circ \theta_{2k-2} \circ q''_{k-1} \circ \phi_3$.

   $q_k(0,0) = 000011(1)^{2k-4}$.

   $q_k((0,0),(1,0)) = (000011(1)^{2k-4},000010(1)^{2k-4},010010(1)^{2k-4})$.

   $q_k((0,0),(1,1)) = (000011(1)^{2k-4},000001(1)^{2k-4},100001(1)^{2k-4})$.

   $q_k((0,0),(1,2)) = (000011(1)^{2k-4},001011(1)^{2k-4},001001(1)^{2k-4})$, 

**Figure 5: $A_{k-1}$**
001101(1)\textsuperscript{2k-4}, 011101(1)\textsuperscript{2k-4}.

\(q_k((0,0),(1,3)) = (000011(1)\textsuperscript{2k-4}, 000111(1)\textsuperscript{2k-4}, 000110(1)\textsuperscript{2k-4}, 000100(1)\textsuperscript{2k-4}, 010100(1)\textsuperscript{2k-4}).

b) On \(Q_{k-1}(1,0)\), \(q'_k = p_{00} \circ \theta_{2k-3,2k-2} \circ q'_{k-1} \circ \phi_0\).

On \(Q_{k-1}(1,1)\), \(q'_k = p_{11} \circ q''_{k-1} \circ \phi_1\).

On \(Q_{k-1}(1,2)\), \(q'_k = p_{00} \circ \theta_{2k-3,2k-2} \circ q_{k-1} \circ \phi_2\).

On \(Q_{k-1}(1,3)\), \(q'_k = p_{10} \circ \theta_{2k-3} \circ q_{k-1} \circ \phi_3\).

\(q'_k(0,0) = 011011(1)\textsuperscript{2k-4}.

\(q'_k((0,0),(1,0)) = (011011(1)\textsuperscript{2k-4}, 011010(1)\textsuperscript{2k-4}, 001010(1)\textsuperscript{2k-4}).

\(q'_k((0,0),(1,1)) = (011011(1)\textsuperscript{2k-4}, 011001(1)\textsuperscript{2k-4}, 111001(1)\textsuperscript{2k-4}).

\(q'_k((0,0),(1,2)) = (011011(1)\textsuperscript{2k-4}, 011111(1)\textsuperscript{2k-4}, 011110(1)\textsuperscript{2k-4}, 011100(1)\textsuperscript{2k-4}, 001100(1)\textsuperscript{2k-4}).

\(q'_k((0,0),(1,3)) = (011011(1)\textsuperscript{2k-4}, 111011(1)\textsuperscript{2k-4}, 111010(1)\textsuperscript{2k-4}, 111000(1)\textsuperscript{2k-4}, 101000(1)\textsuperscript{2k-4}).

c) On \(Q_{k-1}(1,0)\), \(q''_k = p_{11} \circ q''_{k-1} \circ \phi_0\).

On \(Q_{k-1}(1,1)\), \(q''_k = p_{00} \circ \theta_{2k-3,2k-2} \circ q''_{k-1} \circ \phi_1\).

On \(Q_{k-1}(1,2)\), \(q''_k = p_{10} \circ \theta_{2k-3} \circ q_{k-1} \circ \phi_2\).

On \(Q_{k-1}(1,3)\), \(q''_k = p_{11} \circ q_{k-1} \circ \phi_3\).

\(q''_k(0,0) = 100111(1)\textsuperscript{2k-4}.

\(q''_k((0,0),(1,0)) = (100111(1)\textsuperscript{2k-4}, 100110(1)\textsuperscript{2k-4}, 110110(1)\textsuperscript{2k-4}).

\(q''_k((0,0),(1,1)) = (100111(1)\textsuperscript{2k-4}, 100101(1)\textsuperscript{2k-4}, 000101(1)\textsuperscript{2k-4}).

\( q_k^r((0,0),(1,2)) = (100111(1)^{2k-4}, 001111(1)^{2k-4}, 001111(1)^{2k-4}, 001111(1)^{2k-4}, 101110(1)^{2k-4}, 101110(1)^{2k-4}) \).

\( q_k''((0,0),(1,3)) = (100111(1)^{2k-4}, 100011(1)^{2k-4}, 100010(1)^{2k-4}, 100000(1)^{2k-4}, 110000(1)^{2k-4}) \).

In essence, our algorithm first maps four copies of \( q_{k-1}(Q_{k-1}), q'_{k-1}(Q_{k-1}), q''_{k-1}(Q_{k-1}) \), and \( A_{k-1} \) into \( H_{2k+2} \) using \( p_{11}, p_{10} \circ \theta_{2k-3}, p_{01} \circ \theta_{2k-2}, \) and \( p_{00} \circ \theta_{2k-3,2k-2} \), as illustrated in Figure 6. These images are then rearranged to form the height-\( k \) quadtrees illustrated in Figure 7. Note that the vertex \( 111111(1)^{2k-4} = 1111(1)^{2k-2} \) does not appear in \( q_k(Q_k), q'_k(Q_k), \) or \( q''_k(Q_k) \). Also, \( A_k \), the unused subgraph of \( H_{2k+2} \), corresponds to the format of its predecessor in \( H_{2k} \). Our algorithm's correctness is consequently verified.

### 2.5 Algorithm Analysis

Define \( \text{Dil}(q_k, q'_k, q''_k) \) and \( \overline{\text{Dil}}(q_k, q'_k, q''_k) \) to be, respectively, the maximum and average dilations of the embedding that maps three copies of \( Q_k \) into \( H_{2k+2} \), one with \( q_k \), another with \( q'_k \), and a third with \( q''_k \). Similarly, \( \text{Cong}(q_k, q'_k, q''_k) \) is the congestion of that embedding. The recursive nature of our algorithm yields the following results:

**Theorem 2.5.1** \( \text{Dil}(q_k, q'_k, q''_k) = 4; \overline{\text{Dil}}(q_k, q'_k, q''_k) = (7 \cdot 4^{k-1} - 3)/(4^k - 1) \rightarrow 7/4 \) as \( k \rightarrow \infty \).
Figure 6: The images of $q_{k-1}(Q_{k-1})$, $q'_{k-1}(Q_{k-1})$, $q''_{k-1}(Q_{k-1})$, and $A_{k-1}$ under $p_{11}$, $p_{10} \circ \theta_{2k-3}$, $p_{01} \circ \theta_{2k-2}$, and $p_{00} \circ \theta_{2k-3,2k-2}$. 
Figure 1: 

\[ \mathcal{V} \]

\[ (\mathcal{V}, \mathcal{V}_b) \]

\[ (\mathcal{V}, \mathcal{V}_b) \]

\[ (\mathcal{V}, \mathcal{V}_b) \]

\[ (\mathcal{V}, \mathcal{V}_b) \]
Proof. An analysis of Step 2 of our algorithm reveals that half of the edges in the top height-$(k-1)$ subtree of each copy of $Q_k$ receive dilation 2, while the other half receive dilation 4. Of the remaining edges, the definitions of $q_1$, $q_1'$, and $q_1''$ indicate that two-thirds receive dilation 1, and one-third receive dilation 2. Of the total of $4^{k+1} - 4$ edges, then, $\frac{1}{2}(4^k - 4)$ receive dilation 4, $\frac{1}{2}(3 \cdot 4^k - 4)$ receive dilation 2, and $2 \cdot 4^k$ receive dilation 1. □

The following theorem indicates that, as with our previous algorithm, sacrificing low dilation prevents congestion problems with each of our new embeddings. However, by not sacrificing dilation too much, our algorithm prevents congestion problems between the individual embeddings.

**Theorem 2.5.2** $Cong(q_k) = Cong(q'_k) = Cong(q''_k) = Cong(q_k, q'_k, q''_k) = 1$.

Proof. As with the congestion proof for our first algorithm, this result follows from the disjoint nature of the prefix functions and a simple induction on $k$. □

### 2.6 Summary

In this chapter, we have presented a unit-congestion, optimal-expansion embedding of a complete quadtree into the hypercube. The maximum dilation of this embedding is five. The allowance of such relatively high dilation values in order to ensure low congestion represents a radical departure from the embedding techniques employed on the earlier store-and-forward communication technology.
Under the restrictions of that technology, an optimal (i.e., unit-dilation, optimal-expansion) embedding of the quadtree into the hypercube would be impossible. Thus, we have illustrated the increased capacity for efficient algorithm implementation on the new direct-connect technology.

We have also explored the novel task allocation strategy of embedding several structures into a single hypercube, with no regard for the relegation of separate tasks to disjoint subcubes. Our algorithm embeds three height-$k$ quadtrees into a single hypercube with dimension $2k + 2$. Using the traditional task-allocation approach would have required the use of three dimension-$(2k + 1)$ hypercubes, with the resultant waste of over $2^{2k+1}$ hypercube nodes. Our algorithm effectively utilizes all but one hypercube node with no mutual interference between the individual embedded quadtrees. Although such results would undoubtedly be difficult to achieve with other algorithmic structures, the impressive results with quadtrees certainly indicate that such an avenue of approach merits consideration.

The quadtree, then, may be added to the list of algorithmic structures which can be optimally embedded into the hypercube. Although it does not lend itself to such an embedding as easily as binary trees or rectangular meshes, the sacrifice of optimal dilation does result in the necessary maneuverability to permit an embedding that is optimal on all important counts.

In the next chapter, we will examine the possibility of embedding an algorithmic structure for which a lack of optimality appears inevitable: the pyramid.
CHAPTER III

Embedding Pyramids into the Hypercube

The results in the previous chapter illustrate that an algorithmic structure that does not naturally lend itself to embedding in the hypercube may, in fact, be embeddable with optimal expansion and optimal congestion. With the advent of the newer direct-connect hypercube technology, the presence of non-unit dilation in that embedding may be deemed insignificant.

In this chapter, the substantially more complex problem of embedding a pyramid into the hypercube is addressed. The structure of the pyramid is drastically different from that of the hypercube, effectively inhibiting its being embedded optimally with respect to all three of the pertinent cost gauges.

We begin this chapter by examining the inherent limitations of embedding pyramids into hypercubes. We then present an algorithm which embeds a pyramid into a hypercube with optimal expansion and optimal dilation. This algorithm merits attention when the implementation of pyramid algorithms takes place on the older store-and-forward hypercubes, in which high dilation is extremely detrimental. A
second algorithm is presented next; it embeds pyramids into hypercubes at a higher
dilation cost, but a lower congestion cost, thus proving more valuable on the newer
hypercubes. Simulations of the implementations of these two embeddings are then
examined, and the relative merits of each algorithm are then discussed. We con­
clude this chapter with an extension of the second algorithm to improve expansion
by embedding three disjoint pyramids into a hypercube.

3.1 Preliminary Pyramid Results

Define $M_k$ to be a $2^k \times 2^k$ mesh with vertex set

$$V(M_k) = \{(x_1, x_2) : 0 \leq x_1, x_2 \leq 2^k - 1\} \quad (5)$$

and edge set

$$E(M_k) = \{((x_1, x_2), (x_1', x_2')) : |x_1 - x_1'| + |x_2 - x_2'| = 1\}. \quad (6)$$

Define $P_k$ to be the pyramid structure with

$$V(P_k) = \bigcup_{i=0}^{k} \{(i, x_1, x_2) : (x_1, x_2) \in V(M_i)\} \quad (7)$$

and

$$E(P_k) = \bigcup_{i=1}^{k} \{((i, x_1, x_2), (i, x_1', x_2')) : ((x_1, x_2), (x_1', x_2')) \in E(M_i)\} \cup \bigcup_{i=1}^{k} \{((i, x_1, x_2), (i - 1, \lfloor \frac{x_1}{2} \rfloor, \lfloor \frac{x_2}{2} \rfloor)) : (x_1, x_2) \in V(M_i)\}. \quad (8)$$

Intuitively, $P_k$ contains $M_0$ through $M_k$, with each vertex in $M_i$ having four
offspring in $M_{i+1}$. $V(P_k)$ can be separated into levels; specifically, $(i, x_1, x_2) \in$
$V(P_k)$ is on level $i$. The subgraph generated by the level $k$ vertices of $P_k$ is called the base of the pyramid. The vertex on level 0 of $P_k$ is called the apex of the pyramid. $P_k$ is said to have height $k$. $P_2$ is illustrated in Figure 8.

Note that $|V(P_k)| = \frac{1}{3}(4^{k+1} - 1)$ and $|E(P_k)| = 4^{k+1} - 2^{k+2}$.

**Lemma 3.1.1** Any embedding of $P_k$ into a hypercube requires at least a $(2k + 1)$-dimensional hypercube and, hence, an expansion of at least $(6 \cdot 4^k)/(4^{k+1} - 1) \approx 1.5$.

**Proof.** $|V(P_k)| = \frac{1}{3}(4^{k+1} - 1)$ and $|V(H_{2k})| = 2^{2k} < \frac{1}{3} (4^{k+1} - 1) < 2^{k+1} = |V(H_{2k+1})|$. $|V(H_{2k+1})|/|V(P_k)| = (6 \cdot 4^k)/(4^{k+1} - 1)$. □

**Lemma 3.1.2** Any embedding of $P_k$ into a hypercube must have maximum dilation greater than one.

**Proof.** Saad and Schultz [45] showed that the hypercube contains no odd cycles. The result follows, then, from the existence of triangles in $P_k$. □

**Lemma 3.1.3** For $1 \leq k \leq 3$, if $h : P_k \to H_{2k+1}$ is an embedding, then $\text{Cong}(h) \geq 2$.

**Proof.** The maximum degree of a vertex of $P_k$ is 4, 7, and 9 for $k = 1, 2, \text{and} 3$, respectively. In each case, this value exceeds the value of $2k + 1$. The result then follows. □

Although the proof above cannot be extended to values of $k$ greater than 3, the lemma does indicate that it would be difficult, if not impossible, to develop a recursive algorithm which embeds $P_k$ in $H_{2k+1}$ with unit congestion.
Figure 8: $P_2$
3.2 An Optimal-Dilation Embedding

In this section, we present a recursive algorithm that embeds the pyramid into a hypercube with optimal expansion, maximum dilation two, and maximum congestion three. As shown in the previous section, the achievement of maximum dilation two is optimal. When implementing pyramid algorithms on a store-and-forward hypercube, the embedding algorithm in this section is advantageous.

3.2.1 The Algorithm

Define embedding $g_1 : P_1 \rightarrow H_3$ as illustrated in Figure 9. Note that five edges of $P_1$ receive dilation two, while the other three edges receive dilation one. Two edges of $g_1(P_1)$, specifically, $(000,010)$ and $(001,101)$) receive congestion two; the remaining nine edges of $g_1(P_1)$ receive unit congestion. The structure of this
pyramid, including the relative positions of the dilated and congested edges, plays a significant role in our algorithm.

More generally, let \( P_i(k - 1, x_1, x_2) \) denote the subpyramid of \( P_k \) with height one and apex \((k - 1, x_1, x_2)\). For \( v \in V(H_{2k+1}) \) and \( 1 \leq p, q, r \leq 2k + 1 \), let \( H_3(v; p, q, r) \) be the 3-dimensional subcube of \( H_{2k+1} \) generated by \( \{v, v^p, v^q, v^r, v^{pq}, v^{pr}, v^{qr}, v^{pqr}\} \). \( P_i(k - 1, x_1, x_2) \) may be embedded into \( H_3(v; p, q, r) \) by any of the four embeddings \( g_1, g_1^H, g_1^V, \) and \( g_1^{VH} \) illustrated in Figure 10. The positions of corresponding dilated and congested edges in these embeddings exhibit the mirrored quality examined in Section 3.

Intuitively, we shall use a top-down approach to recursively construct this embedding. After having embedded the top \( P_{k-1} \)-subpyramid of \( P_k \) using \( g_{k-1} \), we will use \( g_1 \) and its reflections to embed each of the height-one subpyramids arrayed at the base of \( P_k \).

For this purpose, we should first observe that the edges of \( P_k \) are composed of four disjoint sets:

- \( A_k = \{(i, x_1, x_2), (j, x_1', x_2') \in E(P_k) : 0 \leq i, j \leq k - 1 \} \), the edges in the top \( P_{k-1} \)-subpyramid of \( P_k \);

- \( B_k = \{(k, x_1, x_2), (k - 1, \lfloor \frac{k}{2} \rfloor, \lfloor \frac{k}{2} \rfloor) \} \), the edges joining the base of \( P_k \) to the top \( P_{k-1} \)-subpyramid;

- \( C_k = \{(k, x_1, x_2), (k, x_1', x_2') \in E(P_k) : \lfloor \frac{k}{2} \rfloor = \lfloor \frac{x_1}{2} \rfloor, \lfloor \frac{k}{2} \rfloor = \lfloor \frac{x_2}{2} \rfloor \} \), the edges
Figure 10: $P_1(k - 1, x_1, x_2)$ and its image under $g_1$, $g_1^H$, $g_1^V$, and $g_1^{ VH}$
in the base of some \( P_k(k - 1, u_1, u_2) \);

- \( D_k = \{((k, x_1, x_2), (k, x_1', x_2')) \in E(P_k) - C_k\} \) the edges in the base of \( P_k \)

which are not in \( C_k \).

Note that \( A_k = \bigcup_{i=1}^{k-1} (B_i \cup C_i \cup D_i) \), where \( B_i, C_i, \) and \( D_i \) are defined in an analogous fashion.

For each vertex \( v \) in the base of \( P_k \), let \( \alpha^i(v) \) denote the ancestor of \( v \) on level \( i \) of \( P_k \) (i.e., \( \alpha^i(k, x_1, x_2) = (i, \lfloor x_1/2^{k-i} \rfloor, \lfloor x_2/2^{k-i} \rfloor) \)). Also, for \( a, b \in \{0, 1, \ldots, m - 1\} \), define \( \{a\}_m = \{x : x \mod m = a\} \) and \( \{a, b\}_m = \{a\}_m \cup \{b\}_m \).

Using this notation, we may present our recursive algorithm for the embedding

\[ g_k : P_k \rightarrow H_{2^k+1} : \]

**Algorithm 1**

1. If \( k = 1 \), embed \( P_k \) in \( H_{2k+1} \) using \( g_1 \).

2. If \( k > 1 \), use \( g_{k-1} \) to define \( g_k \) as follows:

   a) Embed the top \( P_{k-1} \)-subpyramid of \( P_k \) in \( H_{2k+1} \) using \( p \circ g_{k-1} \),

   where \( p : H_{2k-1} \rightarrow H_{2k+1} \) such that \( p(x) = 00x \).

   b) For each vertex \( u = (k - 1, x_1, x_2) \) on level \( k - 1 \) of \( P_k \), embed the height-one subpyramid \( P_1(u) \) in \( H_3(00g_{k-1}(u); 2k - 1, 2k, 2k + 1) \)

   using the mapping:

   \[ g_1 \quad \text{if } x_1, x_2 \in \{0\}_2 \]
c) The remaining unembedded edges are in the base of $P_k$, but not part of any of the height-one subpyramids arrayed in the base, i.e., the edges of $D_k$.

For any edge $e = (u, v) \in D_k$, let $e' = (u', v') = (\alpha^{k-1}(u), \alpha^{k-1}(v))$.

If $g_k(e') = (g_k(u'), g_k(u')^i, g_k(v'))$ for some $i \in \{1, \ldots, 2k+1\}$, then embed $e$ in $H_{2k+1}$ using $g_k(e) = (g_k(u), g_k(u)^i, g_k(v))$.

Otherwise, if $g_k(e') = (g_k(u'), g_k(v'))$, then let $g_k(e) = (g_k(u), g_k(v))$.

The algorithm is illustrated in Figure 11.

3.2.2 Correctness

We can now validate the correctness of this algorithm by verifying that the vertex mapping is one-to-one and that the edge mappings in step 2(c) of the algorithm are well-defined.

Lemma 3.2.1 Under the mapping $g_k$, $V(P_k)$ is mapped as follows:

\[ g_k(i, x_1, x_2) = \begin{cases} 
 00g_{k-1}(i, x_1, x_2) & \text{if } 0 \leq i \leq k - 1 \\
 11g_{k-1}(u) & \text{if } x_1, x_2 \in \{0, 3\}_4 \\
 10g_{k-1}(u) & \text{if } x_1 \in \{0, 3\}_4, x_2 \in \{1, 2\}_4 \\
 01(g_{k-1}(u))^{2k-1} & \text{if } x_1 \in \{1, 2\}_4, x_2 \in \{0, 3\}_4 \\
 10(g_{k-1}(u))^{2k-1} & \text{if } x_1, x_2 \in \{1, 2\}_4 
\end{cases} \]
Figure 11: $g_2(P_2)$
where \( u = \alpha^{k-1}(k, x_1, x_2) = (k - 1, \left\lfloor \frac{x_1}{2} \right\rfloor, \left\lfloor \frac{x_2}{2} \right\rfloor) \).

**Proof.** The result for \( 0 \leq i \leq k - 1 \) follows directly from step 2(a) of the algorithm. The result for base vertices follows from step 2(b) and the definition of \( g_1, g_1^V, g_1^H, \) and \( g_1^{YH} \).

For \( u, v \in V(P_k) \), let \( S(u, v) \) denote the set \( \{ i : 1 \leq i \leq 2k + 1, g_k(u) \text{ differs from } g_k(v) \text{ in bit } i \} \).

The following lemma illustrates the validity of step 2(c) of the algorithm by showing that the bit differences between the images of endpoints of edges in \( D_k \) are inherited from their immediate ancestors.

**Lemma 3.2.2** If \( e = (u, v) \in D_k \) and \( e' = (u', v') = (\alpha^{k-1}(u), \alpha^{k-1}(v)) \), then \( S(u, v) = S(u', v') \) and \( |S(u, v)| \leq 2 \).

**Proof.** By definition, \( D_k = \{(k, x_1, x_2), (k, x_1 + 1, x_2) : x_1 \in \{1\}_2 \} \cup \{(k, x_1, x_2), (k, x_1, x_2 + 1) : x_2 \in \{1\}_2 \} \). Lemma 3.2.1 shows, then, that \( g_k(u) \) and \( g_k(v) \) have identical two-bit prefixes and therefore differ in exactly the same bits as \( g_{k-1}(u') \) and \( g_{k-1}(v') \). Since \( g_k(u') = 00g_{k-1}(u') \) and \( g_k(v') = 00g_{k-1}(v') \), the first part of the lemma follows.

The definitions of the embeddings \( g_1, g_1^V, g_1^H, \) and \( g_1^{YH} \) indicate that edges in any \( C_i (1 \leq i \leq k) \) are mapped to paths of length at most two. To prove a similar result for \( D_k \), we will assume it for \( D_{k-1} \) and proceed inductively (the inductive base, \( D_2 \), is illustrated in Figure 11). Using the notation of the lemma, \( g_k(u) \) and
$g_k(v)$ differ in exactly the same bits as $g_{k-1}(u')$ and $g_{k-1}(v')$. Since $(u', v')$ is either in $C_{k-1}$ or $D_{k-1}$, the second part of the lemma follows.

The following lemma demonstrates that $g_k$ is, in fact, an embedding of $P_k$ into $H_{2k+1}$.

**Lemma 3.2.3** For $u, v \in V(P_k)$, if $u \neq v$, then $g_k(u) \neq g_k(v)$.

**Proof.** Lemma 3.2.1 indicates that we need only prove that no two vertices in the base of $P_k$ have the same image under $g_k$; the result then follows by induction.

Applying Lemma 3.2.1 and a simple induction, we may conclude that if $u'$ and $v'$ are distinct vertices in the base of $P_{k-1}$, then $g_{k-1}(u') \neq (g_{k-1}(v'))^{2k-1}$. Using this result, Lemma 3.2.1 shows that no two base vertices of $P_k$ can receive the same image under $g_k$ with prefix 10. Similar arguments can be made regarding the other possible prefixes of the images of base vertices of $P_k$.

In order to examine the dilation and congestion under $g_k$, we need to analyze the embedding's effect on the four subsets $A_k, B_k, C_k, D_k \subseteq E(P_k)$, as defined in Section 4.1.1. We accomplish this by specifying the bit differences which occur in the images of elements of each of these subsets.

**Lemma 3.2.4**

(i) If $e \in A_k$, then $g_k(e) = 00g_{k-1}(e)$. 
(ii) If $e = (u, v) = ((k - 1, \lfloor \frac{k}{2} \rfloor), (k, x_1, x_2)) \in B_k$, then

$$g_k(e) = \begin{cases} (0g_{k-1}(u), 01g_{k-1}(u), 11g_{k-1}(u)) & \text{if } x_1, x_2 \in \{0, 3\}_4 \\ (00g_{k-1}(u), 01g_{k-1}(u), 01(g_{k-1}(u))^{2k-1}) & \text{if } x_1 \in \{0, 3\}_4, \\ (00g_{k-1}(u), 01g_{k-1}(u), 10(g_{k-1}(u))^{2k-1}) & \text{if } x_1 \in \{1, 2\}_4, \\ (00g_{k-1}(u), 00(g_{k-1}(u))^{2k-1}, 10(g_{k-1}(u))^{2k-1}) & \text{if } x_1, x_2 \in \{1, 2\}_4. \\ \end{cases}$$

(iii) If $e = (u, v) = ((k, x_1, x_2), (k, x_1 + 1, x_2)) \in C_k$, then

$$g_k(e) = \begin{cases} (g_k(u), (g_k(u))^{2k-1}) & \text{if } x_1 \in \{0\}_2, x_2 \in \{1, 2\}_4 \\ (g_k(u), (g_k(u))^{2k-1}, (g_k(u))^{2k-1,2k+1}) & \text{if } x_1 \in \{0\}_4, x_2 \in \{0, 3\}_4 \\ (g_k(u), (g_k(u))^{2k+1}, (g_k(u))^{2k+1,2k-1}) & \text{if } x_1 \in \{2\}_4, x_2 \in \{0, 3\}_4. \\ \end{cases}$$

(iv) If $e = (u, v) = ((k, x_1, x_2), (k, x_1, x_2 + 1)) \in C_k$, then

$$g_k(e) = \begin{cases} (g_k(u), (g_k(u))^{2k}) & \text{if } x_1 \in \{0, 3\}_4, x_2 \in \{0\}_2 \\ (g_k(u), (g_k(u))^{2k}, (g_k(u))^{2k,2k+1}) & \text{if } x_1 \in \{1, 2\}_4, x_2 \in \{0\}_4 \\ (g_k(u), (g_k(u))^{2k+1}, (g_k(u))^{2k+1,2k}) & \text{if } x_1 \in \{1, 2\}_4, x_2 \in \{2\}_4. \\ \end{cases}$$

**Proof.** (i) follows from step 2(a) of the algorithm; (ii)-(iv) follow from step 2(b) and the definition of $g_1, g_1^Y, g_1^H, g_1^{YH}$.

**Corollary 3.2.5** For any $x_1, x_2, x_1', x_2'$ with $0 \leq x_1, x_2, x_1', x_2' \leq 2^{k-1} - 1$, with $(x_1, x_2) \neq (x_1', x_2')$, $g_k(P_1(k - 1, x_1, x_2))$ and $g_k(P_1(k - 1, x_1', x_2'))$ are vertex-disjoint and edge-disjoint.

**Proof.** Lemma 3.2.4(ii)-(iv) shows that all vertices of $g_k(P_1(k - 1, x_1, x_2))$ have the same suffix of $2k - 2$ bits. Similarly, all vertices of $g_k(P_1(k - 1, x_1', x_2'))$ have the same suffix of $2k - 2$ bits. Lemma 3.2.1 shows that these two $(2k - 2)$-bit suffixes are not the same. 

\[\Box\]
Complications arise when attempts are made to analyze the image under $g_k$ of an edge $e = (u, v)$ in set $D_k$. Such an edge joins the base vertices of neighboring subpyramids of the form $P_i(k-1, x_1, x_2)$. Lemma 3.2.2 indicates that the sequence of bit differences in path $g_k(e)$ will reflect the bit differences in the corresponding path between the $g_k$-images of the ancestors of $u$ and $v$. Determining these bit differences will enable us to complete the specification of $g_k(E(P_k))$ and analyze the dilation and congestion of $g_k$.

For any edge $(u, v) \in D_k$, define

$$\sigma(u, v) = \min_{0 \leq i \leq k-1} \{ i : \alpha^i(u) \neq \alpha^i(v) \} \quad (9)$$

(i.e., $\sigma(u, v)$ is the level at which the ancestors of $u$ and $v$ have the same parent).

**Lemma 3.2.6**

(i) If $e = (u, v) = ((k, x_1, x_2), (k, x_1 + 1, x_2)) \in D_k$, then,

letting $u'' = (\sigma(u, v), x''_1, x''_2)$ be the ancestor of $u$ at level $\sigma(u, v)$ and $v'' = (\sigma(u, v), x''_1 + 1, x''_2)$ be the ancestor of $v$ at level $\sigma(u, v)$, we have:

$$g_k(e) = \begin{cases} (g_k(u), (g_k(u))^{2\sigma(u,v)-1}) & \text{if } x''_1 \in \{0\}_2, x''_2 \in \{1, 2\}_4 \\ (g_k(u), (g_k(u))^{2\sigma(u,v)-1}, (g_k(u))^{2\sigma(u,v)+1}) & \text{if } x''_1 \in \{0\}_4, x''_2 \in \{0, 3\}_4 \\ (g_k(u), (g_k(u))^{2\sigma(u,v)+1}, (g_k(u))^{2\sigma(u,v)+1, 2\sigma(u,v)-1}) & \text{if } x''_1 \in \{2\}_4, x''_2 \in \{0, 3\}_4 \end{cases}$$

(ii) If $e = (u, v) = ((k, x_1, x_2), (k, x_1, x_2 + 1)) \in D_k$, then, letting $u'' = (\sigma(u, v), x''_1, x''_2)$ be the ancestor of $u$ at level $\sigma(u, v)$ and $v'' =
(σ(u, v), x'_1, x'_2 + 1) be the ancestor of v at level σ(u, v), we have:

\[ g_k(e) = \begin{cases} (g_k(u), (g_k(u))^{2σ(u, v)}) & \text{if } x'_1 \in \{0, 3\}_4, x'_2 \in \{0\}_2 \\ (g_k(u), (g_k(u))^{2σ(u, v)}, (g_k(u))^{2σ(u, v), 2σ(u, v)+1}) & \text{if } x'_1 \in \{1, 2\}_4, x'_2 \in \{0\}_4 \\ (g_k(u), (g_k(u))^{2σ(u, v)+1}, (g_k(u))^{2σ(u, v)+1, 2σ(u, v)}) & \text{if } x'_1 \in \{1, 2\}_4, x'_2 \in \{2\}_4 \end{cases} \]

**Proof.** The proof of Lemma 3.2.2 is easily extended to show that \( S(u, v) = \{ i : 1 \leq i \leq 2k + 1, g_k(u) \text{ differs from } g_k(v) \text{ in bit } i \} = \{ i : 1 \leq i \leq 2k + 1, g_k(u'') \text{ differs from } g_k(v'') \text{ in bit } i \} \). Since \((u'', v'') \in C_σ(u, v)\) by the definition of \( σ(u, v)\), substituting \( σ(u, v) \) for \( k \) in Lemma 3.2.4(iii)-(iv) yields the desired result. \( \square \)

### 3.2.3 Dilation

The disjoint subsets \( A_k, B_k, C_k, D_k \subseteq E(P_k) \) defined in Section 4.1 may now be utilized to determine the dilation of \( g_k \).

**Theorem 3.2.7** Under embedding \( g_k \), \( \frac{7}{3}4^k - 2^{k+1} - \frac{1}{3} \) edges of \( P_k \) receive dilation two, and the remaining \( \frac{5}{3}4^k - 2^{k+1} + \frac{1}{3} \) edges receive dilation one.

**Proof.** The edge equations in Lemma 3.2.4 and Lemma 3.2.2 demonstrate that \( g_k \) yields a maximum dilation of two. Since \( |E(P_k)| = 4^{k+1} - 2^{k+2} \), we need only prove the first part of the theorem.

Inducting on \( k \), we note that the result has been shown for \( g_1 \). Assuming the result for \( g_{k-1} \), we shall prove it for \( g_k \).
For \( X \subseteq E(P_k) \), define \( \psi^k(X) = \{|e \in X : e \text{ receives dilation two under } g_k\} \).

Thus,

\[
\psi^k(E(P_k)) = \psi^k(A_k) + \psi^k(B_k) + \psi^k(C_k) + \psi^k(D_k)
\]

\[
= \psi^{k-1}(E(P_{k-1})) + \frac{3}{4}|B_k| + \frac{1}{2}|C_k| + \frac{1}{2}|D_k|
\]

\[
= \left(\frac{7}{3}d^{k-1} - 2^k - \frac{4}{3}\right) + \frac{3}{4}(4^k) + \frac{1}{2}(4^k - 2^k) + \frac{1}{2}(4^k - 2^k)
\]

\[
= \frac{7}{3}d^k - 2^{k+1} - \frac{1}{3}
\]

(10)

\[\square\]

The average dilation of \( g_k \) can be computed by means of Theorem 3.2.7.

**Corollary 3.2.8** \( \overline{Dil}(g_k) = (19 \cdot 2^k + 1)/(3 \cdot 2^{k+2}) \).

Note that \( \overline{Dil}(g_k) \to \frac{19}{12} \) as \( k \to \infty \).

### 3.2.4 Congestion

To analyze the congestion of this embedding, we first examine the overlap between the images of the disjoint subsets \( A_k, B_k, C_k, D_k \subseteq E(P_k) \); any edge in the hypercube that is in the image of more than one of these subsets will be shown to have a correspondingly higher congestion. Determining the degree of edge congestion that occurs within these subsets will then enable us to specify the congestion of \( g_k \).

For \( E' \subseteq E(P_k) \), let \( g_k(E') \) denote the set of edges \( \{e \in E(g_k(P_k)) : e \text{ is on path } g_k(e') \text{ for some } e' \in E'\} \).
The following lemma begins our evaluation of the congestion of $g_k$ by specifying
the overlap that takes place between the images of the various $A_i$'s, $B_i$'s, $C_i$'s, and
$D_i$'s that comprise $P_k$.

**Lemma 3.2.9** For $1 \leq i \leq k$,

(i) $g_k(A_i) \cap g_k(B_i) = \{(g_k(v), (g_k(v))^{2i-1}) : v = (i-1, x_1, x_2); 0 \leq x_1, x_2 \leq 2^{i-1} - 1\}$.

(ii) $g_k(A_i) \cap g_k(C_i) = g_k(A_i) \cap g_k(D_i) = \emptyset$.

(iii) $g_k(A_i) \cap g_k(C_i) = \emptyset$.

(iv) $g_k(B_i) \cap g_k(D_i) = \{(g_k(v), (g_k(v))^{2i-1}) : v = (i, x_1, x_2); x_1, x_2 \in \{1, 2\}_4\}$.

(v) $g_k(C_i) \cap g_k(D_i) = \{(g_k(v), (g_k(v))^{2i-1}) : v = (i, x_1, x_2); x_1 \in \{2, 5\}_8; x_2 \in \{0, 7\}_8\}$.

Proof. The recursive nature of the algorithm allows us to limit our proof to the
case $i = k$.

(i) Substituting $k - 1$ for $k$ in Lemma 3.2.4(ii)-(iii), we obtain \{$(g_{k-1}(k-1, x_1, x_2))^{2^{k-1}-1}) : 0 \leq x_1, x_2 \leq 2^{k-1} - 1\} \subseteq g_{k-1}(B_{k-1} \cup C_{k-1})$. By Lemma 3.2.4(i), then, our desired set is a subset of $g_k(A_k)$. 

Lemma 3.2.4(ii) shows that this set is in $g_k(B_k)$ and is the maximum possible intersection of $g_k(A_k)$ and $g_k(B_k)$.

(ii) Both endpoints of an edge in $g_k(A_k)$ have prefix 00. At most one endpoint of an edge in $g_k(C_k)$ or $g_k(D_k)$ has prefix 00.

(iii) If an edge $e \in B_k$ and an edge $e' \in C_k$ have overlapping images, then Corollary 3.2.5 shows that $e$ and $e'$ are edges in the same $P_i(k-1, x_1, x_2)$. Our result then follows from step 2(b) of the algorithm and from the embedding definitions illustrated in Figure 10.

(iv) Let $e \in g_k(B_k) \cap g_k(D_k)$. Since $g_k$ has maximum dilation two, at least one endpoint of $e$ is in $g_k(V(P_k))$. Using Corollary 3.2.5, it can be shown that there exist $u, v, w \in V(P_k)$ such that $(u, v) \in B_k$, $(v, w) \in D_k$, $e = (g_k(v), (g_k(v))^i)$, and $e$ is on both path $g_k(u, v)$ and path $g_k(v, w)$. Lemma 3.2.4(ii) and Lemma 3.2.6 indicate that $i = 2k - 1$, and Lemma 3.2.1 shows that $g_k(v)$ does not have prefix 00.

Let $v = (k, x_1, x_2)$ and $u = (k - 1, \lfloor \frac{x_1}{2} \rfloor, \lfloor \frac{x_2}{2} \rfloor) = \alpha^{k-1}(v)$. Thus, Lemma 3.2.4(ii) indicates that $g_k(v) = 01(g_{k-1}(u))^{2k-1}$ and $x_1 \in \{1, 2\}_4$, $x_2 \in \{0, 3\}_4$.

Since $2k - 1 \in S(v, w) = \{ i : 1 \leq i \leq 2k - 1, g_k(v) \text{ and } g_k(w) \text{ differ in bit } i \}$, Lemma 3.2.6 shows that $\sigma(v, w) = k - 1$ and either (a) $S(v, w) =$
\{2k-1,2k-2\} \text{ and } [\frac{2k}{2}], [\frac{2k}{2}] \in \{1,2\}_4, \text{ or (b) } S(v,w) = \{2k-1,2k-3\} \ \text{ and } [\frac{2k}{2}] \in \{1,2\}_4, [\frac{2k}{2}] \in \{0,3\}_4.

Since \(\sigma(v,w) = k-1\), we know that \(v\) and \(w\) are in the same height-two subpyramid of \(P_k\). Therefore, under either case (a) or case (b) above, \(w = (k, x_1 \pm 1, x_2)\). By Lemma 3.2.6, then, case (a) is impossible.

Case (b) indicates that \(x_1, [\frac{x_1}{2}] \in \{1,2\}_4\) and \(x_2, [\frac{x_2}{2}] \in \{0,3\}_4\), so \(x_1 \in \{2,5\}_8\) and \(x_2 \in \{0,7\}_8\). Therefore, \(g_k(B_k) \cap g_k(D_k) \subseteq \{(g_k(v), (g_k(v))^{2k-1}) : v = (k, x_1, x_2); x_1 \in \{2,5\}_8; x_2 \in \{0,7\}_8\}\). Equality is obtained by applying Lemma 3.2.4(ii) and Lemma 3.2.6.

\(v\) An argument similar to that for part (iv) shows that \(g_k(C_k) \cap g_k(D_k) = E' \cup E''\), where \(E' = \{(g_k(v), (g_k(v))^{2k-1}) : v = (k, x_1, x_2); x_1 \in \{2,5\}_8; x_2 \in \{1,2\}_4\}\) and \(E'' = \{(g_k(v), (g_k(v))^{2k-1}) : v = (k, x_1, x_2); x_1 \in \{3,4\}_8; x_2 \in \{2,5\}_8\}\). From Lemma 3.2.1 we may conclude that \(E'' = \{(g_k(v))^{2k-1}, g_k(v)\} : v = (k, x_1, x_2); x_1 \in \{2,5\}_8; x_2 \in \{2,5\}_8\}\), so \(E'' \subseteq E'\) and our result follows. \(\square\)

Lemma 3.2.9 reveals the overlap that occurs between the images of the top \(P_{k-1}\)-subpyramid, the base of \(P_k\), and the edges joining the two. The following corollary refines that analysis by examining the overlap that occurs between edges on different levels of the pyramid and between edges joining different pairs of pyramid levels. This analysis will further assist us in the determination of the
congestion of $g_k$.

**Corollary 3.2.10** For $1 \leq i, j \leq k$,

(i) if $|i - j| > 1$, then $g_k(B_i) \cap g_k(B_j) = \emptyset$

(ii) if $i \neq j$, then $g_k(C_i) \cap g_k(C_j) = g_k(D_i) \cap g_k(D_j) = \emptyset$.

(iii) $g_k(B_i) \cap g_k(C_j) \cap g_k(D_j) = \emptyset$.

**Proof.**

(i) Without loss of generality, assume that $i \leq j - 2$. Lemma 3.2.4(i) and (ii) indicate that the endpoints of elements of $g_k(B_i)$ have value zero in bits $2i + 2$ through $2j + 1$, while at least one endpoint of any element of $g_k(B_j)$ has value one in bit $2j - 1$, $2j$, or $2j + 1$.

(ii) Without loss of generality, assume that $i < j$. Lemma 3.2.4(i) illustrates that the endpoints of elements of $g_k(C_i)$ and $g_k(D_i)$ have value zero in bits $2j$ and $2j + 1$, while Lemma 3.2.1 shows that at least one endpoint of elements of $g_k(C_j)$ and $g_k(D_j)$ has value one in either bit $2j$ or $2j + 1$.

(iii) Lemma 3.2.1 indicates that a nonempty intersection could occur only if $j \in \{i - 1, i\}$. Substituting $i - 1$ for $k$ in Lemma 3.2.9(v) indicates that the endpoints of elements of $g_k(C_{i-1}) \cap g_k(D_{i-1})$ differ in bit $2i - 3$. Substituting $i$ for $k$ in Lemma 3.2.9(i) shows that the endpoints of elements of $g_k(A_i) \cap g_k(B_i)$ differ in bit $2i - 1$ and, since $g_k(C_{i-1}) \cap$
\( g_k(D_{i-1}) \subseteq g_k(A_i) \), we can conclude that \( g_k(B_i) \cap g_k(C_{i-1}) \cap g_k(D_{i-1}) = \emptyset \). Finally, substituting \( i \) for \( k \) in Lemma 3.2.9(iii) and (iv) indicates that \( g_k(B_i) \cap g_k(C_i) \cap g_k(D_i) = \emptyset \). □

The above results indicate how much congestion takes place between the images of the disjoint sets of pyramid edges. To complete an analysis of the congestion of \( g_k \), we must examine the amount of congestion that takes place within these individual sets.

For \( E' \subseteq E(P_k) \), we say that \( e \in g_k(E') \) has congestion \( t \) within \( g_k(E') \) if \( |\{e' \in E' : e \text{ is on path } g_k(e')\}| = t \). Finally, we define \( \tau^k_t(E') = \{e \in g_k(E') : e \text{ has congestion } t \text{ within } g_k(E')\} \).

**Lemma 3.2.11** Using the definition of \( \tau^k_t(E') \) given above, we have, for \( 1 \leq i \leq k \),

(i) \( \tau^k_2(B_i) = \{(g_k(v), (g_k(v))^{2i}) : v = (i - 1, x_1, x_2); 0 \leq x_1, x_2 \leq 2^{i-1} - 1\} \);

\( \tau^k_1(B_i) = g_k(B_i) - \tau^k_2(B_i) \).

(ii) \( \tau^k_1(C_i) = g_k(C_i) \).

(iii) \( \tau^k_2(D_i) = \{(g_k(v), (g_k(v))^{2i-1}) : v = (i, x_1, x_2); x_1, x_2 \in \{2, 5\}_3\} \);

\( \tau^k_1(D_i) = g_k(D_i) - \tau^k_2(D_i) \).

**Proof.** Again, we need only prove the lemma for the case \( i = k \).

(i) From Corollary 3.2.5, we know that an edge has congestion \( t \) within \( g_k(B_k) \) only if it has congestion \( t \) within its particular \( g_k(P_1(k-1, x_1, x_2)) \).
This fact, together with step 2(b) of the algorithm, yields the desired result.

(ii) Similar to part (i).

(iii) The analysis of $g_k(D_k)$ in the proof of part (iv) of Lemma 3.2.9 demonstrates that any congested edges within $g_k(D_k)$ must be in the same height-two subpyramid of $P_k$. Step 2(c) of the algorithm then allows us to conclude that $(g_k(v), (g_k(v))^{2k-1}) = ((g_k(u))^{2k-1}, g_k(u))$ if $v = (k, x_1, x_2)$, $x_1 \in \{2\}_8$, $x_2 \in \{2, 5\}_8$, and $u = (k, x_1 + 1, x_2)$. Similar results hold for $x_1 \in \{5\}_8$, $x_2 \in \{2, 5\}_8$, and $u = (k, x_1 - 1, x_2)$. Step 2(c) of the algorithm illustrates that these are the only occurrences of congestion within $g_k(D_k)$. □

Now that the congestion between the images of the disjoint sets of pyramid edges and the congestion within each of these sets have been determined, we are able to completely specify the congestion of every edge in $g_k(P_k)$. The proof technique breaks the edges of $g_k(P_k)$ into several disjoint sets, each of which consists exclusively of edges with congestion one, two, or three.

**Theorem 3.2.12** Under embedding $g_k$, $\frac{2}{3}(4^{k-1} - 1)$ of the edges in $g_k(P_k)$ have congestion three, $\frac{1}{3}(11 \cdot 4^{k-1} - 5)$ edges have congestion two, and the remaining $4^{k+1} - 3 \cdot 2^{k+1} + 5$ edges have congestion one.
Proof. Define the following sets of edges in $g_k(P_k)$:

$$E_1 = \bigcup_{i=2}^{k-1} \{ (g_k(v), (g_k(v))^{2i-1}) : v = (i, x_1, x_2); x_1, x_2 \in \{2, 5\}_8 \};$$  \hfill (11)

$$E_2 = \bigcup_{i=1}^{k-1} \{ (g_k(v), (g_k(v))^{2i+1}) : v = (i, x_1, x_2); x_1, x_2 \in \{1, 2\}_4 \};$$  \hfill (12)

$$E_3 = \bigcup_{i=2}^{k} \{ (g_k(v), (g_k(v))^{2i-1}) : v = (i, x_1, x_2); x_1 \in \{2, 5\}_8; x_2 \in \{1, 6\}_8 \};$$  \hfill (13)

$$E_4 = \{ (g_k(v), (g_k(v))^{2k+1}) : v = (k, x_1, x_2); x_1, x_2 \in \{1, 2\}_4 \};$$  \hfill (14)

$$E_5 = \bigcup_{i=1}^{k-1} \{ (g_k(v), (g_k(v))^{2i+1}) : v = (i, x_1, x_2); x_1 \text{ or } x_2 \text{ is not in } \{1, 2\}_4 \};$$  \hfill (15)

$$E_6 = \bigcup_{i=0}^{k-1} \{ (g_k(v), (g_k(v))^{2i+2}) : v = (i, x_1, x_2) \}$$  \hfill (16)

$$E_7 = E(g_k(P_k)) - \bigcup_{j=1}^{6} E_j.$$  \hfill (17)

We will show that the sets $E_1$ through $E_7$ are pairwise disjoint and then analyze the congestion of edges in each of these sets. Specifically, we will show that edges in $E_1$ and $E_2$ have congestion three; edges in $E_3$ through $E_6$ have congestion two; and edges in $E_7$ have congestion one. The disjoint quality of these sets can then be used to obtain the desired result by summing the cardinalities of the appropriate $E_i$'s.

By definition, $E_1 \cap E_7 = \emptyset$. Since the bit differences in edges of $E_6$ are even, $E_1 \cap E_6 = \emptyset$. Similarly, the only bit difference in edges of $E_4$ is $2k+1$, so $E_1 \cap E_4 = \emptyset$. Letting $E' = \bigcup_{i=1}^{k} g_k(B_i)$, Lemma 3.2.9(iii) shows that $E_2 \subseteq E'$ and Lemma 3.2.9(i) shows that $E_5 \subseteq E'$. Lemma 3.2.9(v) shows that $E_1 \subseteq \bigcup_{i=1}^{k} (g_k(C_i) \cap g_k(D_i))$. 
Corollary 3.2.10(iii) then indicates that $E_1 \cap E' = \emptyset$, so $E_1 \cap E_2 = \emptyset$ and $E_1 \cap E_3 = \emptyset$.

Corollary 3.2.5 indicates that $E_1 \cap E_3 = \emptyset$. Therefore, $E_1 \cap E_j = \emptyset$ for $2 \leq j \leq 7$.

Similar arguments yield that the sets $E_1$ through $E_7$ are, in fact, pairwise disjoint.

Lemma 3.2.9(v) indicates that $E_1 \subseteq \bigcup_{i=1}^{k}(g_k(C_i) \cap g_k(D_i))$. If $e = (g_k(v), (g_k(v))^{2i-1}) \in E_1$, then $e$ has congestion one within $g_k(C_i)$ (by Lemma 3.2.11(ii)) and congestion two within $g_k(D_i)$ (by Lemma 3.2.11(iii)). Corollary 3.2.10 indicates that these are the only occurrences of $e$ in $g_k(P_k)$, so $e$ has congestion three. Thus, edges in $E_1$ have congestion three in $g_k(P_k)$. The edge $(10001, 10101)$ in Figure 11 illustrates the congestion of edges in $E_1$.

Lemma 3.2.9(i) and (iii) indicate that $E_2 \subseteq \bigcup_{i=1}^{k-1}(g_k(B_i) \cap g_k(C_i) \cap g_k(B_{i+1}))$.

Any edge $e = (g_k(v), (g_k(v))^{2i+1}) \in E_2$ has congestion one within $g_k(B_i)$, $g_k(C_i)$, and $g_k(B_{i+1})$ by Lemma 3.2.11(i) and (ii). As with $E_1$ above, Corollary 3.2.10 now indicates that edges in $E_2$ (e.g., $(00101, 00001)$ in Figure 11) have congestion three.

Similarly, since Lemma 3.2.9(v) indicates that $E_3 \subseteq \bigcup_{i=1}^{k-1}(g_k(C_i) \cap g_k(D_i))$, Lemma 3.2.11(ii) and (iii) and Corollary 3.2.10 show that any edge in $E_3$ (e.g., $(10111, 10011)$ in Figure 11) has congestion two.

$E_4 = g_k(B_k) \cap g_k(C_k)$ by Lemma 3.2.9(iii), so edges in $E_4$ have congestion two by Corollary 3.2.10 and Lemma 3.2.11(i) and (ii). In Figure 11, $(10111, 00111) \in E_4$.

Lemma 3.2.9(i) and Lemma 3.2.4(ii)-(iv) indicate that each element of $E_5$ is in either $\bigcup_{i=1}^{k-1}(g_k(B_{i+1}) \cap g_k(C_i))$ (e.g., $(00011, 00111)$ in Figure 11), or $\bigcup_{i=1}^{k-1}(g_k(B_i) \cap g_k(B_{i+1}))$ (e.g., $(00110, 00010)$ in Figure 11). In either case, Corollary 3.2.10 and
Lemma 3.2.11(i) and (ii) indicate that edges in $E_6$ have congestion two.

$E_6 = \bigcup_{i=1}^{k} \tau_2^k(B_i)$ by Lemma 3.2.11(i). Lemma 3.2.9(i), (iii), and (iv) and Corollary 3.2.10(i) indicate that additional occurrences of edges in $E_6$ are impossible, so edges in $E_6$ (e.g., (00000,00010) in Figure 11) have congestion two.

Lemma 3.2.4 and Lemma 3.2.6 indicate that $g_k(B_i) \cap g_k(C_j) = \emptyset$ and $g_k(B_i) \cap g_k(D_j) = \emptyset$ unless $j \in \{i-1,i\}$. Thus, Lemma 3.2.9 and Corollary 3.2.10 indicate that the only overlaps between edges in the various $g_k(B_i)$'s, $g_k(C_i)$'s, and $g_k(D_i)$'s occur in $\bigcup_{j=1}^{k} E_j$. Lemma 3.2.11 indicates that $\bigcup_{i=1}^{k} \tau_2^k(B_i) \subseteq E_6$, $\bigcup_{i=1}^{k} \tau_2^k(C_i) = \emptyset$, and $\bigcup_{i=1}^{k} \tau_2^k(D_i) \subseteq E_1$. We may conclude that all edges with congestion larger than one are in $E_1$ through $E_6$, so edges in $E_7$ have congestion one. Note that $|E_1| = |E_2| = |E_3| = |E_6| = \frac{1}{3}(4^{k-1} - 1)$, $|E_4| = 4^{k-1}$, and $|E_5| = 4^{k-1} - 1$. Since $E_1$ through $E_7$ are pairwise disjoint, Theorem 3.2.7 indicates that $|E_7| = 4^{k+1} - 3 \cdot 2^{k+1} + 5$ and our result follows. □

Theorem 3.2.12 yields the following corollary.

**Corollary 3.2.13** \( \overline{\text{Cong}}(g_k) = (19 \cdot 4^k - 9 \cdot 2^{k+1} - 1)/(61 \cdot 4^{k-1} - 9 \cdot 2^{k+1} + 8) \).

Note that $\overline{\text{Cong}}(g_k) \to \frac{76}{61} \approx 1.25$ as $k \to \infty$.

### 3.3 A Congestion-Two Embedding

In this section, we present a recursive algorithm that embeds the pyramid in a hypercube with optimal expansion. This algorithm has maximum congestion two
and maximum dilation three. It consequently represents a significant improvement over the previous pyramid embedding with respect to the direct-connect hypercube technology.

3.3.1 The Algorithm

Define embedding $h_1 : P_1 \rightarrow H_3$ as illustrated in Figure 12. Of the eight edges in $P_1$, one receives dilation three, two receive dilation two, and five receive dilation one. Only one edge of $h_1(P_1)$, $(000, 100)$, receives congestion two; the ten others receive unit congestion. Note that 000, the image of the apex, is adjacent to 010, an element of $V(h_1(P_1)) - h_1(V(P_1))$ and that the edge between these two vertices has congestion one under $h_1$. The existence of such an edge will be important in the recursive extension of this embedding to larger pyramids.
Assume now that $h_{k-1} : P_{k-1} \to H_{2k-1}$ is an embedding with $\alpha = h_{k-1}(0,0,0)$, and that $\beta = \alpha^{2k-2} \in V(h_{k-1}(P_{k-1})) - h_{k-1}(V(P_{k-1}))$ such that $\alpha$ and $\beta$ are adjacent in $h_{k-1}(P_{k-1})$ and the edge $(\alpha, \beta)$ has congestion one under $h_{k-1}$. We show that $h_{k-1}$ can be used to construct an embedding $h_k : P_k \to H_{2k+1}$ such that $\alpha' = h_k(0,0,0)$ has a neighbor $\beta' = (\alpha')^{2k} \in V(h_k(P_k)) - h_k(V(P_k))$ and such that edge $(\alpha', \beta')$ has congestion one.

Let $P_{k-1}(1, x_1, x_2)$ denote the subpyramid of $P_k$ with height $k - 1$ and apex $(1, x_1, x_2)$. Note that $P_k$ has four such subpyramids. On the other hand, $H_{2k+1}$ has four $(2k - 1)$-dimensional subcubes: namely, $x_2x_1H_{2k-1}$, where $x_1, x_2 \in \{0, 1\}$.

Intuitively, we will use $h_{k-1}$ to embed each $P_{k-1}(1, x_1, x_2)$ into $x_2x_1H_{2k-1}$. The images of these embeddings will then be symmetrically arranged in a “mirrored” fashion, so that corresponding vertices on different subpyramids face each other. The $(\alpha, \beta)$ edge incident to each subpyramid’s apex will then be used to construct the apex of the embedded height-$k$ pyramid.

In order to formally describe the embedding, we introduce some notation. Define $\phi_{k-1}^V : P_{k-1} \to P_{k-1}$ to be the “vertical-reflection” embedding of $P_{k-1}$ into itself, so that vertex $(i, x_1, x_2) \in V(P_{k-1})$ is mapped to $\phi_{k-1}^V(i, x_1, x_2) = (i, x_1, 2^i - x_2 - 1)$ and edge $(u, v) \in E(P_{k-1})$ is mapped to $\phi_{k-1}^V(u, v) = (\phi_{k-1}^V(u), \phi_{k-1}^V(v))$. Similarly, $\phi_{k-1}^H : P_{k-1} \to P_{k-1}$ is defined as the “horizontal-reflection” embedding of $P_{k-1}$ into itself, so that $\phi_{k-1}^H(i, x_1, x_2) = (i, 2^i - x_1 - 1, x_2)$ and $\phi_{k-1}^H(u, v) = (\phi_{k-1}^H(u), \phi_{k-1}^H(v))$. 


Define three additional embeddings of $P_{k-1}$ into $H_{2k-1}$ as follows:

1. $h^V_{k-1} = h_{k-1} \circ \varphi^V_{k-1}$, the vertical reflection of $h_{k-1}$.

2. $h^H_{k-1} = h_{k-1} \circ \varphi^H_{k-1}$, the horizontal reflection of $h_{k-1}$.

3. $h^{VH}_{k-1} = h_{k-1} \circ \varphi^V_{k-1} \circ \varphi^H_{k-1}$, the double reflection of $h_{k-1}$.

Figure 13 illustrates $h^V_1$, $h^H_1$, and $h^{VH}_1$.

Define $t_{x_1,x_2} : P_{k-1}(1,x_1,x_2) \rightarrow P_{k-1}$ so that $t_{x_1,x_2}(i,x'_1,x'_2) = (i - 1, x'_1 - x_1(2^{i-1}), x'_2 - x_2(2^{i-1}))$ for any vertex $(i,x'_1,x'_2)$ in $P_{k-1}(1,x_1,x_2)$ and $t_{x_1,x_2}(u,v) = (t_{x_1,x_2}(u), t_{x_1,x_2}(v))$ for any edge $(u,v)$ in $P_{k-1}(1,x_1,x_2)$. Thus, $t_{x_1,x_2}$ merely transforms the coordinates of vertices in $P_{k-1}(1,x_1,x_2)$ into the corresponding coordinates in $P_{k-1}$.

Finally, for any two binary values $b_1, b_2$, define the prefix function $p_{b_2b_1} : H_{2k-1} \rightarrow H_{2k+1}$ so that $p_{b_2b_1}(x) = b_2b_1x$ and $p_{b_2b_1}(x,x') = (b_2b_1x, b_2b_1x')$ for any vertex $x$ and edge $(x,x')$ in $H_{2k-1}$.

Using this notation, we may present our recursive algorithm for the embedding $h_k : P_k \rightarrow H_{2k+1}$:

**Algorithm 2**

1. If $k = 1$, embed $P_k$ in $H_{2k+1}$ using $h_1$.

2. If $k > 1$, use $h_{k-1}$ to define $h_k$ as follows:
Figure 13: The image of $P_1$ under $h_1^V$, $h_1^H$, and $h_1^{VH}$
a) Embed the subpyramid $P_i(0,0,0)$ in the 3-dimensional subcube of $H_{2k+1}$ generated by $\{00\alpha, 00\beta, 01\alpha, 01\beta, 10\alpha, 10\beta, 11\alpha, 11\beta\}$ such that:

$$
\begin{align*}
  h_k(0,0,0) &= 00\beta, \\
  h_k(1,0,0) &= 00\alpha, \\
  h_k(1,1,0) &= 01\alpha, \\
  h_k((0,0,0),(1,0,0)) &= (00\beta,00\alpha), \\
  h_k((0,0,0),(1,0,1)) &= (00\beta,10\beta,10\alpha), \\
  h_k((0,0,0),(1,1,0)) &= (00\beta,01\beta,01\alpha), \\
  h_k((0,0,0),(1,1,1)) &= (00\beta,10\beta,11\beta,11\alpha).
\end{align*}
$$

b) For each vertex $(1, x_1, x_2)$, embed $P_{k-1}(1, x_1, x_2)$ in $x_2 x_1 H_{2k-1}$ using the mapping:

$$
\begin{align*}
  p_{00} \circ h_{k-1} \circ t_{x_1,x_2} & \quad \text{if } x_1 = x_2 = 0 \\
  p_{10} \circ h_{k-1}^H \circ t_{x_1,x_2} & \quad \text{if } x_1 = 1, x_2 = 0 \\
  p_{01} \circ h_{k-1}^Y \circ t_{x_1,x_2} & \quad \text{if } x_1 = 0, x_2 = 1 \\
  p_{11} \circ h_{k-1}^{YH} \circ t_{x_1,x_2} & \quad \text{if } x_1 = x_2 = 1
\end{align*}
$$

c) For each edge $(u, v) \in E(P_k)$ with $u$ and $v$ in different subpyramids of the type $P_{k-1}(1, x_1, x_2)$, let $h_k(u, v) = (h_k(u), h_k(v))$.

The algorithm is illustrated in Figure 14.
Figure 14: The recursive definition of $h_k$
3.3.2 Correctness

To validate the correctness of our algorithm, we must verify that the vertices are mapped in a one-to-one fashion, that the congestion-one edge \((\alpha, \beta)\) assumed in \(h_{k-1}(P_{k-1})\) has a corresponding congestion-one edge \((\alpha', \beta')\) in \(h_k(P_k)\), and that the edge mappings described in step 2(c) of the algorithm are well-defined. We should also note that several of the mappings described in step 2(a) are redefined in steps 2(b) and 2(c), but that these definitions are identical.

**Lemma 3.3.1** Using the mapping \(h_k\), \(V(P_k)\) is mapped as follows:

\[
h_k(i, x_1, x_2) =
\begin{cases}
00\beta & \text{if } (i, x_1, x_2) = (0, 0, 0) \\
00h_{k-1}(i - 1, x_1, x_2) & \text{if } (i, x_1, x_2) \in P_{k-1}(1, 0, 0) \\
01h_{k-1}(i - 1, 2^i - x_1 - 1, x_2) & \text{if } (i, x_1, x_2) \in P_{k-1}(1, 1, 0) \\
10h_{k-1}(i - 1, x_1, 2^i - x_2 - 1) & \text{if } (i, x_1, x_2) \in P_{k-1}(1, 0, 1) \\
11h_{k-1}(i - 1, 2^i - x_1 - 1, 2^i - x_2 - 1) & \text{if } (i, x_1, x_2) \in P_{k-1}(1, 1, 1)
\end{cases}
\]

**Proof.** This follows directly from steps 2(a) and 2(b) of the algorithm and the definitions of \(p_{b2b1}, h_{k-1}, h_{k-1}^V, h_{k-1}^H, h_{k-1}^{HV}\), and \(t_{x_1, x_2}\). □

The following corollary illustrates the validity of step 2(c) of the algorithm.

**Corollary 3.3.2** If \((u, v)\) is an edge in \(P_k\) and \(u\) and \(v\) are in different subpyramids of the type \(P_{k-1}(1, x_1, x_2)\), then \(h_k(u)\) is adjacent to \(h_k(v)\) in \(H_{2k+1}\).

**Proof.** The lemma shows that, for \(1 \leq i \leq k\) and \(0 \leq x_1, x_2 \leq 2^i - 1\), \(h_k(i, 2^{i-1} - 1, x_2)\) is adjacent to \(h_k(i, 2^{i-1}, x_2)\) in bit \(2k\), and \(h_k(i, x_1, 2^{i-1} - 1)\) is adjacent to \(h_k(i, x_1, 2^{i-1})\) in bit \(2k+1\). □
Corollary 3.3.3 For $u, v \in V(P_k)$, if $u \neq v$, then $h_k(u) \neq h_k(v)$.

Proof. Simple induction on $k$, utilizing the previous lemma. □

Corollary 3.3.3 demonstrates that $h_k$ is an embedding of $P_k$ into $H_{2k+1}$. Note that $h_k(0,0,0) = (00\beta)$ and that $01\beta = (00\beta)^2 \in V(h_k(P_k) - h_k(V(P_k))$ such that $00\beta$ and $01\beta$ are adjacent in $h_k(P_k)$ and the edge $(00\beta, 01\beta)$ has congestion one under $h_k$. Our recursive algorithm is well-defined.

Some notation is needed before we can examine the dilation and congestion of $h_k$. The edges of $P_k$ are composed of six disjoint sets:

- $E(P_{k-1}(1,0,0))$, $E(P_{k-1}(1,0,1))$, $E(P_{k-1}(1,1,0))$, and $E(P_{k-1}(1,1,1))$, as defined above;
- $X_k = \{(u,v) \in E(P_k) : u = (0,0,0)\}$, the edges joining the apex of $P_k$ to the apex of each $P_{k-1}(1,x_1,x_2)$;
- $Y_k = \{((i,2^{i-1} - 1,x_2),(i,2^{i-1},x_2)) : 1 \leq i \leq k, 0 \leq x_2 \leq 2^i - 1\} \cup \{((i, x_1,2^{i-1} - 1),(i, x_1,2^{i-1})) : 1 \leq i \leq k, 0 \leq x_1 \leq 2^i - 1\}$, the edges joining corresponding vertices in neighboring subpyramids of the form $P_{k-1}(1,x_1,x_2)$.

3.3.3 Dilation

The results of the previous section enable us to determine the dilation of $h_k$ with a minimum of effort.
Theorem 3.3.4 Using embedding $h_k$, $\frac{1}{3}(4^k - 1)$ edges of $P_k$ receive dilation three, $\frac{2}{3}(4^k - 1)$ edges receive dilation two, and the remaining $3 \cdot 4^k - 2^{k+2} + 1$ edges receive dilation one.

Proof. A simple induction shows that the maximum dilation under $h_k$ is three. The theorem has already been shown for $h_1$. Assuming, then, that it holds for $h_{k-1}$, we will prove it for $h_k$.

For $X \subseteq E(P_k)$, define $\delta_i^k(X) = |\{e \in X : e \text{ receives dilation } i \text{ under embedding } h_k\}|$. Hence:

$$\delta_3^k(E(P_k)) = \delta_3^k(X_k) + \delta_3^k(Y_k) + \sum_{0 \leq x_1, x_2 \leq 1} \delta_3^k(E(P_{k-1}(1, x_1, x_2)))$$

$$= 1 + 0 + 4\delta_3^{k-1}(E(P_{k-1}))$$

$$= 1 + 4\left(\frac{1}{3}(4^{k-1} - 1)\right)$$

$$= \frac{1}{3}(4^k - 1). \quad (18)$$

$$\delta_2^k(E(P_k)) = \delta_2^k(X_k) + \delta_2^k(Y_k) + \sum_{0 \leq x_1, x_2 \leq 1} \delta_2^k(E(P_{k-1}(1, x_1, x_2)))$$

$$= 2 + 0 + 4\delta_2^{k-1}(E(P_{k-1}))$$

$$= 2 + 4\left(\frac{2}{3}(4^{k-1} - 1)\right)$$

$$= \frac{2}{3}(4^k - 1). \quad (19)$$

$$\delta_1^k(E(P_k)) = |E(P_k)| - \delta_3^k(E(P_k)) - \delta_2^k(E(P_k))$$
The following corollary is an immediate result of Theorem 3.3.4.

Corollary 3.3.5  \( \overline{Dil}(h_k) = \frac{2^{k+2} + 1}{3 \cdot 2^k} \).

Note that \( \overline{Dil}(h_k) \rightarrow \frac{4}{3} \) as \( k \rightarrow \infty \).

3.3.4 Congestion

Examining the amount of overlap between the images of the disjoint subsets of \( E(P_k) \) and the amount of edge congestion that occurs within these sets, will allow us to specify the congestion of the embedding \( h_k \).

Lemma 3.3.6  
(i) If \( (x_1, x_2) \neq (x'_1, x'_2) \), then \( h_k(P_{k-1}(1, x_1, x_2)) \) and \( h_k(P_{k-1}(1, x'_1, x'_2)) \) have no edges in common.

(ii) Each edge in \( h_k(Y_k) \) has congestion one.

(iii) \( (x_1x_2\alpha, x_1x_2\beta) \) is the only edge that \( h_k(P_{k-1}(1, x_1, x_2)) \) and \( E(h_k(X_k)) \) have in common.

Proof.

(i) Step 2(b) of the algorithm shows that the two-bit prefixes of vertices in \( h_k(P_{k-1}(1, x_1, x_2)) \) and \( h_k(P_{k-1}(1, x'_1, x'_2)) \) must differ.
(ii) The proof of (i) above guarantees that no edge in $h_k(Y_k)$ is in any $h_k(P_{k-1}(1, x_1, x_2))$. Step 2(a) of the algorithm and the fact that $\beta$ was assumed not to be in $h_{k-1}(V(P_{k-1}))$ show that no edge in $h_k(Y_k)$ is in $h_k(X_k)$. The result then follows from Corollary 3.3.3.

(iii) This follows directly from steps 2(a) and 2(b) of the algorithm. □

**Corollary 3.3.7** $\text{Cong}(h_k) = 2$.

**Proof.** Simple induction on $k$, using Lemma 3.3.6. □

The previous results indicate that the congestion of $h_k$ can be determined by separating $E(P_k)$ into the six disjoint subsets mentioned in Section 3.3.2, observing the congestion that occurs within each subset, and taking note of the overlap that occurs between the images of the subsets.

**Theorem 3.3.8** Under embedding $h_k$, $\frac{1}{3}(2^{k+1}+5)$ edges in $h_k(P_k)$ have congestion two, while the remaining $4^{k+1} - 2^{k+2} + 2$ edges have congestion one.

**Proof.** (By induction on $k$) The result has been shown for $h_1$ and, after making the inductive assumption for $h_{k-1}$, we shall prove it for $h_k$.

For $Y \subseteq E(h_k(P_k))$, define $\gamma_k^i(Y) = |\{e \in Y : e \text{ has congestion } i \text{ under embedding } h_k\}|$. Therefore,

$$\gamma_2^k(E(h_k(P_k))) = \gamma_2^k(h_k(X_k)) + \gamma_2^k(h_k(Y_k)) + \sum_{0 \leq x_1, x_2 \leq 1} \gamma_2^k(h_k(E(P_{k-1}(1, x_1, x_2))))$$
\[
\sum_{0 \leq x_1, x_2 \leq 1} \gamma^k_2(h_k(X_k) \cap h_k(E(P_{k-1}(1, x_1, x_2))))
= 5 + 0 + 4\gamma^{k-1}_2(E(h_{k-1}(P_{k-1})))
= 5 + \frac{4}{3}(2^{2k-1} - 5)
= \frac{1}{3}(2^{2k+1} - 5).
\]

\[
\gamma^k_1(E(h_k(P_k))) = \gamma^k_1(h_k(X_k)) + \gamma^k_1(h_k(Y_k)) + \sum_{0 \leq x_1, x_2 \leq 1} \gamma^k_1(h_k(E(P_{k-1}(1, x_1, x_2))))
= 2 + |Y_k| + 4(\gamma^{k-1}_1(E(h_{k-1}(P_{k-1}))) - 1)
= 2 + 4(2^k - 1) + 4(4^k - 2^{k+1} + 1)
= 4^{k+1} - 2^{k+2} + 2.
\]

Theorem 3.3.8 enables us to compute the average congestion for \( h_k \).

**Corollary 3.3.9** \( \overline{\text{Cong}}(h_k) = (4^{k+2} - 3 \cdot 2^{k+2} - 4)/(14 \cdot 4^k - 3 \cdot 2^{k+2} + 1) \).

Note that \( \overline{\text{Cong}}(h_k) \to \frac{8}{7} \) as \( k \to \infty \).

### 3.4 Simulation Results

To demonstrate the significance of lower congestion when implementing parallel algorithms on the hypercube, the pyramid embeddings from the previous two sections were coded in Pascal and the results were used in the simulation of random computations and communications of pyramid nodes via direct-connect hypercube processors.
Table 1: Embedding $g_k$ Simulation Results - Static Routing

<table>
<thead>
<tr>
<th>$k$</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>pyr. ht.</th>
<th>ave. comm. time</th>
<th>ave. comm. gap</th>
<th># of msgs</th>
<th>max. comm. time</th>
<th># of delays</th>
<th>ave. delay per msg.</th>
<th>max. delay time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>64</td>
<td>733</td>
<td>23.77</td>
<td>9</td>
<td>0.03001</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>368</td>
<td>27.54</td>
<td>3</td>
<td>0.02446</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>204</td>
<td>20.28</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>91</td>
<td>15.42</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>64</td>
<td>2992</td>
<td>35.15</td>
<td>36</td>
<td>0.03910</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>1665</td>
<td>27.37</td>
<td>12</td>
<td>0.01742</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>857</td>
<td>22.97</td>
<td>3</td>
<td>0.00933</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>384</td>
<td>22.95</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>64</td>
<td>12276</td>
<td>37.92</td>
<td>190</td>
<td>0.05034</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>6541</td>
<td>44.36</td>
<td>37</td>
<td>0.01880</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>3332</td>
<td>36.58</td>
<td>13</td>
<td>0.01170</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>1629</td>
<td>33.53</td>
<td>3</td>
<td>0.00307</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>64</td>
<td>49157</td>
<td>44.36</td>
<td>677</td>
<td>0.05108</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>25330</td>
<td>44.36</td>
<td>190</td>
<td>0.02369</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>13164</td>
<td>38.82</td>
<td>46</td>
<td>0.01367</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>6279</td>
<td>34.77</td>
<td>7</td>
<td>0.00432</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using exponential distributions for both the length of a communication and the time gap between communications generated by a given processor, the results in Tables 1 and 2 were obtained. Message routing was static in nature (using the paths specified by the respective embedding algorithms), with delays resulting only when at least one edge in a specified communication path was already being used to transmit a message between another pair of processors. The duration of the simulation was ten thousand units of time.

Close analysis of these tables verifies that the $h_k$ embedding has superior performance over the $g_k$ embedding. Although both embeddings encounter relatively few
Table 2: Embedding $h_k$ Simulation Results - Static Routing

<table>
<thead>
<tr>
<th>pyr. ht. $k$</th>
<th>ave. comm. time</th>
<th>ave. comm. gap</th>
<th># of msgs</th>
<th>max. comm. time</th>
<th># of delays</th>
<th>ave. delay per msg.</th>
<th>max. delay time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>64</td>
<td>732</td>
<td>27.79</td>
<td>9</td>
<td>0.03142</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>368</td>
<td>27.54</td>
<td>0</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>195</td>
<td>25.15</td>
<td>0</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>90</td>
<td>22.46</td>
<td>0</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>64</td>
<td>3078</td>
<td>34.42</td>
<td>21</td>
<td>0.02242</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>1587</td>
<td>28.48</td>
<td>6</td>
<td>0.00567</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>849</td>
<td>44.36</td>
<td>1</td>
<td>0.01531</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>426</td>
<td>24.35</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>64</td>
<td>12313</td>
<td>35.57</td>
<td>51</td>
<td>0.01218</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>6451</td>
<td>44.36</td>
<td>21</td>
<td>0.01178</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>3173</td>
<td>31.18</td>
<td>6</td>
<td>0.00347</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>1624</td>
<td>29.22</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>64</td>
<td>49465</td>
<td>44.36</td>
<td>229</td>
<td>0.01579</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>25364</td>
<td>37.19</td>
<td>63</td>
<td>0.00840</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>13095</td>
<td>38.82</td>
<td>16</td>
<td>0.00458</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>6549</td>
<td>36.58</td>
<td>3</td>
<td>0.00183</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

delays in our simulation, we observe that, for larger pyramids, less than one-half of one percent of the messages encounter delays with $h_k$, while nearly 1.5 percent of the messages are delayed with $g_k$. The higher maximum congestion of $g_k$ results in significantly longer delays and an average delay per message that is as much as four times the size of the average delay per message under $h_k$.

It should be noted that the maximum congestion gauge has served as the better predictor of communication performance here; the average congestion of $g_k$ is actually lower than that of $h_k$. Obviously, the existence of congestion-three edges under $g_k$ results in dramatically more serious delays than the existence of a greater
The effect that dynamic routing has upon these embeddings is noteworthy. We ran these same simulations again, but with a dynamic message routing scheme which transmitted in a bitwise left-to-right fashion, keeping track of blocked bit positions, and returning to them later. Delays under this scheme would occur only when all of the bit positions still requiring traversal were blocked. The results of these simulations are recorded in Tables 3 and 4.

The results of these simulations are even more profound. Changing from static to dynamic routing resulted in little improvement for the high-congestion $g_k$ em-

<table>
<thead>
<tr>
<th>pyr. ht. $k$</th>
<th>ave. comm. time</th>
<th>ave. comm. gap</th>
<th># of msgs</th>
<th>max. comm. time</th>
<th># of delays</th>
<th>ave. delay per msg.</th>
<th>max. delay time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>64</td>
<td>732</td>
<td>26.83</td>
<td>18</td>
<td>0.09016</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>419</td>
<td>22.03</td>
<td>3</td>
<td>0.02864</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>181</td>
<td>26.59</td>
<td>2</td>
<td>0.02210</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>108</td>
<td>20.98</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>64</td>
<td>3044</td>
<td>36.58</td>
<td>39</td>
<td>0.03581</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>1524</td>
<td>36.04</td>
<td>9</td>
<td>0.02034</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>857</td>
<td>24.14</td>
<td>4</td>
<td>0.00248</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>415</td>
<td>21.45</td>
<td>1</td>
<td>0.00482</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>64</td>
<td>12340</td>
<td>39.97</td>
<td>170</td>
<td>0.05429</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>6627</td>
<td>44.36</td>
<td>47</td>
<td>0.03456</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>3160</td>
<td>36.04</td>
<td>16</td>
<td>0.01772</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>1632</td>
<td>33.03</td>
<td>1</td>
<td>0.00490</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>64</td>
<td>49413</td>
<td>41.59</td>
<td>660</td>
<td>0.04819</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>25560</td>
<td>41.59</td>
<td>157</td>
<td>0.02246</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>12876</td>
<td>36.04</td>
<td>35</td>
<td>0.01002</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>6566</td>
<td>37.19</td>
<td>11</td>
<td>0.00381</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
Table 4: Embedding $h_k$ Simulation Results - Dynamic Routing

<table>
<thead>
<tr>
<th>pyr. ht. $k$</th>
<th>ave. comm. time</th>
<th>ave. comm. gap</th>
<th># of msgs</th>
<th>max. comm. time</th>
<th># of delays</th>
<th>ave. delay per msg.</th>
<th>max. delay time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>64</td>
<td>736</td>
<td>28.48</td>
<td>0</td>
<td>0.00000</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>390</td>
<td>22.54</td>
<td>0</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>181</td>
<td>20.82</td>
<td>0</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>98</td>
<td>24.68</td>
<td>0</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>64</td>
<td>3011</td>
<td>28.12</td>
<td>3</td>
<td>0.00133</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>1545</td>
<td>36.04</td>
<td>2</td>
<td>0.00841</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>825</td>
<td>24.74</td>
<td>0</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>426</td>
<td>25.49</td>
<td>1</td>
<td>0.03064</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>64</td>
<td>12280</td>
<td>37.19</td>
<td>16</td>
<td>0.00318</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>6609</td>
<td>44.36</td>
<td>6</td>
<td>0.00454</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>3228</td>
<td>36.04</td>
<td>3</td>
<td>0.00465</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>1682</td>
<td>34.10</td>
<td>0</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>64</td>
<td>49322</td>
<td>44.36</td>
<td>72</td>
<td>0.00489</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>25595</td>
<td>36.58</td>
<td>13</td>
<td>0.00145</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>13012</td>
<td>41.59</td>
<td>6</td>
<td>0.00115</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>6709</td>
<td>37.19</td>
<td>1</td>
<td>0.00134</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

bedding; the average delay per message and percentage of messages delayed were only slightly decreased. However, the improvement of performance for the low-congestion $h_k$ embedding is marked: for the larger pyramids, $h_k$ encountered nearly one-third the delays with dynamic routing as with static routing.

The lack of improvement in $g_k$'s performance may be attributed to the fact that the dynamic routing scheme ignores the well-planned paths of the original embedding in favor of a straightforward left-to-right trail, thus opening the door to increasing its already high congestion. Since $h_k$ has lower congestion, the negative effects of such increases are more than counteracted by the routing improvement.
The excellent performance of \( h_k \) in these simulations indicates that maximum congestion two is quite acceptable when unit congestion cannot be attained. However, the results of the simulations using \( g_k \) illustrate how quickly the damaging effects of higher congestion are felt.

### 3.5 A Multiple-Pyramid Embedding with Unit Expansion

The embedding algorithm \( h_k \) presented earlier has optimal expansion and near-optimal congestion and dilation. However, approximately one-third of the nodes of the hypercube are used solely for data communication. In this section, we present an extension of this algorithm that makes use of these nodes to embed two additional, smaller pyramids into the same hypercube, increasing the total expansion to one. Once again, the traditional approach of allocating tasks to subcubes is avoided in order to maximize processor utilization.

#### 3.5.1 The Algorithm

Define embeddings \( a_i : P_i \rightarrow H_5 \) and \( b_i : P_i \rightarrow H_5 \) as illustrated in Figure 15.

Note that \( a_i(V(P_i)) \), \( b_i(V(P_i)) \), and \( h_2(V(P_2)) \) are disjoint subsets of \( V(H_5) \).

Also, using the reflection notation introduced in Section 3.1, note that \( b_i = \mu_4 \circ a_i^H \), where \( \mu_i : H_{i+1} \rightarrow H_{i+1} \) such that, for \( v \in V(H_{i+1}) \) and \( (v_1, v_2) \in E(H_{i+1}) \), \( \mu_i(v) = v^t \) and \( \mu_i(v_1, v_2) = (v_1^t, v_2^t) \).
Figure 15: $a_1(P_1)$ and $b_1(P_1)$

Assume that $a_{k-1} : P_{k-1} \rightarrow H_{2k+1}$ and $b_{k-1} : P_{k-1} \rightarrow H_{2k+1}$ are embeddings with $b_{k-1} = \mu_{2k} \circ a^H_{k-1}$. Letting $\rho = a_{k-1}(0,0,0)$, we have $b_{k-1}(0,0,0) = \rho^{2k}$.

Algorithm 3

1. If $k = 1$, embed $P_k$ in $H_{2k+3}$ using $a_1$.

2. If $k > 1$, use $a_{k-1}$ and $b_{k-1}$ to define $a_k$ as follows:

   a) Embed the subpyramid $P_k(0,0,0)$ in $H_{2k+3}$ so that:

   \[
   a_k(0,0,0) = 10\rho^{2k,2k+1},
   \]
   \[
   a_k(1,0,0) = 00\rho, \quad a_k(1,0,1) = 10\rho,
   \]
   \[
   a_k(1,1,0) = 01\rho^{2k}, \quad a_k(1,1,1) = 11\rho^{2k},
   \]
\begin{align*}
\alpha_k((0,0,0),(1,0,0)) &= (10\rho^{2k,2k+1},00\rho^{2k,2k+1},00\rho^{2k},00\rho), \\
\alpha_k((0,0,0),(1,0,1)) &= (10\rho^{2k,2k+1},10\rho^{2k},10\rho), \\
\alpha_k((0,0,0),(1,1,0)) &= (10\rho^{2k,2k+1},00\rho^{2k,2k+1},00\rho^{2k},01\rho^{2k}), \\
\alpha_k((0,0,0),(1,1,1)) &= (10\rho^{2k,2k+1},10\rho^{2k},11\rho^{2k}).
\end{align*}

b) For each vertex \((1,x_1,x_2)\), embed \(P_{k-1}(1,x_1,x_2)\) in \(x_2x_1H_{2k+1}\) using the mapping:

\begin{align*}
p_{00} \circ \alpha_{k-1} \circ t_{x_1,x_2} & \quad \text{if } x_1 = x_2 = 0 \\
p_{10} \circ b^H_{k-1} \circ t_{x_1,x_2} & \quad \text{if } x_1 = 1, x_2 = 0 \\
p_{01} \circ a^V_{k-1} \circ t_{x_1,x_2} & \quad \text{if } x_1 = 0, x_2 = 1 \\
p_{11} \circ b^{VH}_{k-1} \circ t_{x_1,x_2} & \quad \text{if } x_1 = x_2 = 1
\end{align*}

c) For each edge \((u,v) \in E(P_k)\) with \(u\) and \(v\) in different subpyramids of the type \(P_{k-1}(1,x_1,x_2)\), let

\begin{align*}
\alpha_k(u,v) &= \begin{cases} 
(a_k(u),a_k(u)^{2k+3}) & \text{if } u = (i,t_1,t_2) \text{ and } v = (i,t_1,t_2 + 1) \\
(a_k(u),a_k(u)^{2k+2},a_k(u)^{2k+2k+3}) & \text{if } u = (i,t_1,t_2) \text{ and } v = (i,t_1 + 1,t_2)
\end{cases}
\end{align*}

3. Define \(b_k = \mu_{2k+2} \circ a^H_k\).

The algorithm is demonstrated in Figure 16 and Figure 17.

### 3.5.2 Correctness

As with our previous algorithms, we must validate the correctness of Algorithm 3 by verifying that the vertex mapping is one-to-one and that the edge mappings in step 2(c) of the algorithm are well-defined.
Figure 16: $a_2(P_2)$
Figure 17: $b_2(P_2)$
For $a, b \in \{0, 1, \ldots, m - 1\}$, define $\{a\}_m = \{x : x \mod m = a\}$ and $\{a, b\}_m = \{a\}_m \cup \{b\}_m$.

Lemma 3.5.1 For $v = (i, x_1, x_2) \in V(P_{k-1})$,

\[
\begin{align*}
a_{k-1}(i, x_1, x_2) &= \begin{cases} h_k(v)^{2k-2i+1} & \text{if } x_1 \in \{0\}_2 \\ h_k(v)^{2k-2i+1,2k-2i} & \text{if } x_1 \in \{1\}_2 \end{cases} \\
b_{k-1}(i, x_1, x_2) &= \begin{cases} h_k(v)^{2k-2i+1,2k-2i} & \text{if } x_1 \in \{0\}_2 \\ h_k(v)^{2k-2i+1} & \text{if } x_1 \in \{1\}_2 \end{cases}
\end{align*}
\]

Proof. Since $b_{k-1} = \mu_{2k} \circ a_{k-1}^H$, the result for $b_{k-1}$ will follow from the result for $a_{k-1}$. To prove the latter, we will induct on $k$.

For $k = 2$, the result follows from Figure 14 and Figure 15. Assuming the result for $k$, we will prove it for $k + 1$.

Let $v = (i, x_1, x_2) \in V(P_k)$. If $i > 0$, then four possibilities exist: $0 \leq x_1, x_2 < 2^{i-1}$; $0 \leq x_1 < 2^{i-1}$, $2^{i-1} \leq x_2 < 2^i$; $2^{i-1} \leq x_1 < 2^i$, $0 \leq x_2 < 2^{i-1}$; or $2^{i-1} \leq x_1, x_2 < 2^i$.

Assume that $0 \leq x_1, x_2 < 2^{i-1}$. In this case,

\[
a_k(v) = h_{00} \circ a_{k-1} \circ t_{0,0}(v)
\]

\[
= 00a_{k-1}(i - 1, x_1, x_2)
\]

\[
= \begin{cases} 00h_k(i - 1, x_1, x_2)^{2k-2i+3} & \text{if } x_1 \in \{0\}_2 \\ 00h_k(i - 1, x_1, x_2)^{2k-2i+3,2k-2i+2} & \text{if } x_1 \in \{1\}_2 \end{cases}
\]

\[
= \begin{cases} h_{k+1}(v)^{2k-2i+3} & \text{if } x_1 \in \{0\}_2 \\ h_{k+1}(v)^{2k-2i+3,2k-2i+2} & \text{if } x_1 \in \{1\}_2 \end{cases}
\]

(23)
as desired. The other three cases are similar.

Finally, if \( v = (0,0,0) \), then \( a_k(v) = a_k(1,0,0)^{2k,2k+1,2k+3} \) (by step 2(a) of Algorithm 3) = \( h_{k+1}(1,0,0)^{2k,2k+3} \) (by the above argument) = \( h_{k+1}(0,0,0)^{2k+3} \) (by step 2(a) of Algorithm 2), as desired. \( \square \)

Lemma 3.5.2 Let \( v = (i,x_1,x_2) \in V(P_k) \) and \( T(v) = \{h_k(v)^{2k-2i+1}, h_k(v)^{2k-2i}, h_k(v)^{2k-2i+1,2k-2i}\} \). Then \( T(v) \cap h_k(V(P_k)) = \emptyset \).

Proof. The result for \( k = 1 \) is illustrated in Figure 12. Assuming that it is true for \( k - 1 \), we will prove it for \( k \).

If \( v = (0,0,0) \), then, using the notation of Algorithm 2, we have \( h_k(v) = 00/\beta \) and \( T(v) = \{10/\beta, 01/\beta, 11/\beta\} \). Algorithm 2 specifically guarantees that \( T(v) \cap h_k(V(P_k)) = \emptyset \).

If \( i > 0 \), then there are four possible ranges for \( x_1, x_2 \) (as mentioned in the proof of the previous lemma). Assume that \( 0 \leq x_1, x_2 < 2^{i-1} \). Thus, \( h_k(v) = 00h_{k-1}(i-1, x_1, x_2) \). Since \( 2(k - 1) - 2(i - 1) = 2k - 2i \), our inductive hypothesis shows that \( T(v) \cap 00h_{k-1}(V(P_{k-1})) = \emptyset \). The two-bit prefix 00 of \( h_k(v) \) guarantees that \( T(v) \cap b_1b_2h_{k-1}(V(P_{k-1})) = \emptyset \) for \( b_1b_2 \in \{10, 01, 11\} \). The definition of \( \beta \) in Algorithm 2 shows that \( h_k(0,0,0) \) is not in \( T(v) \). Thus, \( T(v) \cap h_k(V(P_k)) = \emptyset \).

Similar arguments follow for the other three \( x_1, x_2 \) ranges. \( \square \)

Lemma 3.5.3 If \( v, w \in V(P_{k-1}) \) such that \( v \neq w \), then \( a_{k-1}(v), a_{k-1}(w), b_{k-1}(v), \) and \( b_{k-1}(w) \) are four distinct elements of \( V(H_{2k+1}) \).
Proof. By induction on $k$, similar to the proof of Lemma 3.5.2. □

**Corollary 3.5.4** The sets $a_{k-1}(V(P_{k-1}))$, $b_{k-1}(V(P_{k-1}))$, and $h_k(V(P_k))$ are mutually disjoint subsets of $V(H_{2k+1})$. There is only one element of $V(H_{2k+1})$ that is in none of these three sets.

Proof. This follows from the three previous lemmas and the fact that $|V(P_k)| = \frac{1}{3}(4^{k+1} - 1)$, $|V(P_{k-1})| = \frac{1}{3}(4^k - 1)$, and $|V(H_{2k+1})| = 2^{2k+1} = \frac{1}{3}(4^{k+1} - 1) + \frac{2}{3}(4^k - 1) + 1$. □

Finally, we may note that Lemma 3.5.1 and Corollary 3.3.2 illustrate that the edge mappings in step 2(c) of Algorithm 2 are well-defined.

### 3.5.3 Dilation

The dilation of $a_{k-1}$ and $b_{k-1}$ are easily determined.

**Theorem 3.5.5** Using embedding $a_{k-1}$ (or $b_{k-1}$), $\frac{2}{3}(4^{k-1} - 1)$ edges of $P_{k-1}$ receive dilation three, $2^{2k-1} - 2^k$ edges receive dilation two, and the remaining $\frac{1}{3}(4^k + 2) - 2^k$ edges receive dilation one.

Proof. A simple induction shows that the maximum dilation under $a_{k-1}$ (and $b_{k-1}$) is three. The theorem has already been shown for $a_1$ (and $b_1$). Assuming, then, that it holds for $a_{k-2}$ (and $b_{k-2}$), we will prove it for $a_{k-1}$.
For \( X \subseteq E(P_{k-1}) \), define \( \xi_i^{k-1}(X) = |\{e \in X : e \text{ receives dilation } i \text{ under embedding } a_{k-1}\}|. \) Hence:

\[
\xi_3^{k-1}(E(P_{k-1})) = \xi_3^{k-1}(X_{k-1}) + \xi_3^{k-1}(Y_{k-1}) + \sum_{0 \leq x_1, x_2 \leq 1} \xi_3^{k-1}(E(P_{k-2}(1, x_1, x_2)))
\]

\[
= 2 + 4(2^{3 - 2} - 1)
\]

\[
= \frac{2}{3}(4^{k-1} - 1).
\]

(24)

\[
\xi_2^{k-1}(E(P_{k-1})) = \xi_2^{k-1}(X_{k-1}) + \xi_2^{k-1}(Y_{k-1}) + \sum_{0 \leq x_1, x_2 \leq 1} \xi_2^{k-1}(E(P_{k-2}(1, x_1, x_2)))
\]

\[
= 2 + \frac{1}{2}|Y_{k-1}| + 4\xi_2^{k-2}(E(P_{k-2}))
\]

\[
= 2 + (2^k - 2) + 4(2^{2k-3} - 2^{k-1})
\]

\[
= 2^{2k-1} - 2^k.
\]

(25)

\[
\xi_1^{k-1}(E(P_{k-1})) = |E(P_{k-1})| - \xi_3^{k-1}(E(P_{k-1})) - \xi_2^{k-1}(E(P_{k-1}))
\]

\[
= (4^k - 2^{k+1}) - \frac{2}{3}(4^{k-1} - 1) - (2^{2k-1} - 2^k)
\]

\[
= \frac{1}{3}(4^k + 2) - 2^k.
\]

(26)

A similar argument holds for \( b_{k-1} \).  

The following corollary follows directly from Theorem 3.5.5.

**Corollary 3.5.6** \( \overline{Dil}(a_{k-1}) = \overline{Dil}(b_{k-1}) = (11 \cdot 2^{2k-1} - 9 \cdot 2^k - 4)/(3 \cdot 4^k - 3 \cdot 2^{k+1}). \)
Note that $\overline{\text{Dil}}(a_{k-1}) \to \frac{11}{6}$ as $k \to \infty$.

Let $R_k$ be a graph consisting of two height $k - 1$ pyramids, $P_{k-1}$ and $P_{k-1}'$, and one height $k$ pyramid $P_k$. Define $\omega_k : R_k \to H_{2k+1}$ such that $\omega_k|_{P_{k-1}} = a_{k-1}$, $\omega_k|_{P_{k-1}'} = b_{k-1}$, and $\omega_k|_{P_k} = h_k$. Our previous results indicate that $\omega_k$ is an embedding.

**Theorem 3.5.7**

(i) $\overline{\text{Dil}}(\omega_k) = 3$.

(ii) $\overline{\text{Dil}}(\omega_k) = \frac{(9 \cdot 4^k - 5 \cdot 2^{k+1} - 4)}{(6 \cdot 4^k - 2^{k+3})}$.

**Proof.** Part (i) follows from Theorem 3.3.4 and Theorem 3.5.5. Part (ii) follows from Corollary 3.3.5 and Corollary 3.5.6. \[ \square \]

Note that $\overline{\text{Dil}}(\omega_k) \to \frac{3}{2}$ as $k \to \infty$.

### 3.5.4 Congestion

The congestion of $a_{k-1}$ and $b_{k-1}$ can be specified in a manner similar to that used for $h_k$.

**Lemma 3.5.8**

(i) If $(x_1, x_2) \neq (x_1', x_2')$, then $a_{k-1}(P_{k-2}(1, x_1, x_2))$ and $a_{k-1}(P_{k-2}(1, x_1', x_2'))$ have no edges in common.

(ii) Define $E_1, E_2 \subseteq E(a_{k-1}(Y_{k-1}))$ so that $E_1 = \{(a_{k-1}(u), a_{k-1}(u)^{2k-2i}) : u = (i, 2^{i-1}, x_2), 2 \leq i \leq k-1\}$ and $E_2 = E(a_{k-1}(Y_{k-1})) - E_1$.

Then each element of $E_1$ receives congestion two under $a_{k-1}$ and each
element of $E_2$ receives congestion one under $a_{k-1}$. In fact, if $e = (a_{k-1}(u), a_{k-1}(u)^{2k-2i}) \in E_1$, then $e \in E(a_{k-1}(P_{k-2}(1, 1, \lfloor x_2/2^{i-1} \rfloor))$.

(iii) For any $x_1, x_2 \in \{0, 1\}, E(a_{k-1}(P_{k-2}(1, x_1, x_2))) \cap E(a_{k-1}(P_{k-1})) = \emptyset$.

Proof.

(i) Step 2(b) of Algorithm 2 shows that the two-bit prefixes of vertices in $a_{k-1}(P_{k-2}(1, x_1, x_2))$ and $a_{k-1}(P_{k-2}(1, x_1', x_2'))$ must differ.

(ii) Step 2(c) of Algorithm 2 indicates that the edges of $E_2$ have endpoints differing in bit $2k$ or $2k + 1$. Part (i) above illustrates that these edges have congestion one. Step 2(a) of Algorithm 2 indicates that, for $u = (i, 2^{i-1}, x_2)$ and $u' = (i - 1, 2^{i-2}, \lfloor x_2/2 \rfloor)$,

$$a_{k-1}(u', u) = \begin{cases} (a_{k-1}(u)^{2k-2i}, a_{k-1}(u)^{2k-2i}, a_{k-1}(u)) & \text{if } x_2 \in \{1, 2\}_4 \\ (a_{k-1}(u)^{2k-2i+1, 2k-2i+1}, a_{k-1}(u)^{2k-2i}, a_{k-1}(u)) & \text{if } x_2 \in \{0, 3\}_4 \end{cases}$$

The desired result then follows from Lemma 3.5.1 and Lemma 3.5.2.

(iii) This follows from Lemma 3.5.2 and steps 2(a) and 2(b) of Algorithm 2.

Similar results follow for $b_{k-1}$.

**Corollary 3.5.9** $\text{Cong}(a_{k-1}) = \text{Cong}(b_{k-1}) = 2$. 
Proof. Simple induction on \( k \). \( \square \)

**Theorem 3.5.10** Under embedding \( a_{k-1}, \frac{5}{3}4^{k-1} - 2^k + \frac{1}{3} \) edges in \( a_{k-1}(P_{k-1}) \) have congestion two, while the remaining \( 4^k - 2^k - 2 \) edges have congestion one. Similar results hold for \( b_{k-1} \).

Proof. The result has been illustrated for \( a_1 \) (and \( b_1 \)). Using the inductive assumption for \( a_{k-2} \) (and \( b_{k-2} \)), we shall prove the result for \( a_{k-1} \) (and \( b_{k-1} \)).

For \( Y \subseteq E(a_{k-1}(P_{k-1})) \), define \( \theta^{k-1}_i(Y) = |\{e \in Y : e \text{ has congestion } i \text{ under embedding } a_{k-1}\}|. \) Therefore,

\[
\theta^{k-1}_2(E(a_{k-1}(P_{k-1}))) = \theta^{k-1}_2(a_{k-1}(X_{k-1})) \\
+ \sum_{0 \leq x_1, x_2 \leq 1} \theta^{k-1}_2(a_{k-1}(E(P_{k-2}(1, x_1, x_2)))) \\
= 3 + 4\theta^{k-2}_2(E(a_{k-2}(P_{k-2}))) + \frac{1}{2}(|Y_{k-1}|-4) \\
= 3 + 4\left(\frac{5}{3}4^{k-2} - 2^{k-1} + \frac{1}{3}\right) + \frac{1}{2}(4(2^{k-1} - 1) - 4) \\
= \frac{5}{3}4^{k-1} - 2^k + \frac{1}{3}.
\] (27)

Lemma 3.5.8(ii) indicates that

\[
\theta^{k-1}_1(a_{k-1}(E(P_{k-2}(1, x_1, x_2)))) = \begin{cases} 
\theta^{k-2}_1(E(a_{k-2}(P_{k-2}))) & \text{if } x_1 = 0 \\
\theta^{k-2}_1(E(a_{k-2}(P_{k-2}))) - (2^{k-1} - 2) & \text{if } x_1 = 1 
\end{cases}
\] (28)

Therefore,

\[
\theta^{k-1}_1(E(a_{k-1}(P_{k-1}))) = \theta^{k-1}_1(a_{k-1}(X_{k-1})) + \theta^{k-1}_1(a_{k-1}(Y_{k-1}))
\]


\[ + \sum_{0 \leq x_1, x_2 \leq 1} \theta_{i-1}^k(a_{k-1}(E(P_{k-2}(1, x_1, x_2)))) \]

\[ = 4 + (|Y_{k-1}| + 2) + 2\theta_{i-2}^k(E(a_{k-2}(P_{k-2}))) \]

\[ + 2(\theta_{i-2}^k(E(a_{k-2}(P_{k-2}))) - 2^{k-1} + 2) \]

\[ = 4 + (4(2^{k-1} - 1) + 2) + 2(4^{k-1} - 2^{k-1} - 2) \]

\[ + 2(4^{k-1} - 2^{k-1} - 2 - 2^{k-1} + 2) \]

\[ = 4^k - 2^k - 2. \quad (29) \]

Similar arguments hold for \( b_{k-1} \). \( \square \)

**Corollary 3.5.11** \( \text{Cong}(a_{k-1}) = \text{Cong}(b_{k-1}) = \frac{(22 \cdot 4^k - 9 \cdot 2^{k+2} - 16)}{(17 \cdot 4^k - 3 \cdot 2^{k+3} - 20)} \).

**Proof.** This follows directly from the preceding theorem. \( \square \)

Note that \( \text{Cong}(a_{k-1}) \to \frac{22}{17} \) as \( k \to \infty \).

**Theorem 3.5.12** For \( k \geq 3 \), \( 4 \leq \text{Cong}(\omega_k) \leq 6 \).

**Proof.** Figure 16 and Figure 17 illustrate that \( \omega_k(P_{2k+1}) \) contains edges with congestion at least 4. Corollary 3.3.7 and Corollary 3.5.9 indicate that \( \text{Cong}(\omega_k) \) is at most 6.

Although the congestion here is hardly as impressive as that associated with the multiple quadtree embedding of the previous chapter, the fact that multiple pyramid embeddings can be achieved with dilation three, congestion at most six,
and, above all, unit expansion merits attention. When implementing pyramid algorithms which require relatively low amounts of communication between processes, such multiple embeddings are clearly advantageous.

3.6 Summary

In this chapter, we have presented two algorithms for efficiently embedding a pyramid algorithm into a hypercube. Each algorithm satisfies our requirement for optimal expansion. The first algorithm, with maximum congestion three and maximum dilation two, is recommended for use with the store-and-forward hypercube technology. Its optimal dilation minimizes the effect of that technology's inherent communication problems.

Our second algorithm, with maximum congestion two and maximum dilation three, is recommended for use with the direct-connect hypercube technology, in which higher dilation is of little concern. As illustrated by our simulations, this algorithm's low congestion results in very few communication delays. Neither algorithm performs poorly in our direct-connect simulations, but the pronounced superiority of the second algorithm indicates that while low congestion may be acceptable, pursuing the lowest possible congestion is often worth the effort.

We have also presented an embedding of three disjoint pyramids (one height-\(k\) and two height-\((k - 1)\)) into a hypercube with \(2k + 1\) dimensions. Once again, the complete utilization of hypercube processors that results from this nontraditional
approach to task allocation, with reasonably low values of dilation and congestion, represents a viable alternative to the assignment of subcubes to individual tasks.

We note that Ho and Johnsson [27] have extended our results to develop a minimal expansion pyramid embedding with maximum congestion two and maximum dilation two. They have also extended our multiple-pyramid embedding so that it has unit expansion, maximum congestion three, and maximum dilation three.

In this chapter and the preceding chapter, we have examined the embeddings of specific algorithmic structures into the hypercube. The NP-completeness of the problems of embedding general trees and general graphs into the hypercube makes the pursuit of embeddings for such specific classes of graphs worthwhile. However, in the next chapter we shall address the question of embedding into hypercubes from a different perspective when we explore the possibility of developing heuristics that yield good results for the intractable general problems.
CHAPTER IV

Embedding General Trees and Graphs into the Hypercube

Krumme, et al., [33] have shown that the problem of embedding general graphs into a hypercube with unit dilation is NP-complete. More recently, Wagner and Corneil [50] have proved that this problem is also NP-complete when one restricts oneself to embedding general trees into a hypercube. Of particular interest when considering the implementation of algorithms on the newer direct-connect hypercube technology are the results of Kim and Lai [32] which indicate that embedding general graphs into a hypercube with unit congestion is NP-complete.

In the two previous chapters, the intractability of these general embedding problems has led us to restrict our consideration to certain specific classes of graphs: namely, complete quadtrees and pyramids. The advent of the direct-connect hypercube technology has allowed us to disregard the previously serious problem of developing embeddings with minimal dilation, thus permitting us to consider our unit-congestion quadtree embedding optimal. Furthermore, the simu-
lations involving our pyramid embeddings indicate that, although unit congestion is undoubtedly desired, embeddings with very low maximum congestion may have an acceptably low level of communication delay.

In this chapter, we shall use these findings to give us perspective on the intractable general embedding problems mentioned above. We shall present two heuristic approaches to embedding general trees and graphs. Although such algorithms do not lend themselves to the rigorous mathematical scrutiny to which our quadtree and pyramid algorithms have been subjected, by examining the statistical results of several simulations using these general embeddings, we shall be able to estimate the relative merits of each heuristic approach.

4.1 General Tree Embeddings

In this section, we examine two types of heuristic approaches for the problem of embedding general trees into hypercubes. As indicated earlier, guaranteeing that such an embedding algorithm will always yield unit congestion would be a difficult task. In order to achieve low congestion, however, we shall develop heuristics which manipulate the structure of the tree so that it may be positioned in the hypercube without causing severe traffic problems.

4.1.1 Greedy Methods

An obvious place to start when attempting to minimize congestion in an embedding of a graph into a hypercube is to ensure that those nodes in the graph which have
Given graph $G$ to be embedded in hypercube $H$.

$x \leftarrow$ vertex in $G$ with maximum vertex degree;
label($x$) $\leftarrow$ 0;
repeat until \{labelled vertices\} = $V(G)$
\begin{align*}
x & \leftarrow$ labelled vertex in $G$ with maximum
number of unlabelled neighbors;
for each unlabelled neighbor $y$ of $x$ do
label($y$) $\leftarrow$ unused label closest to label($x$)
\end{align*}

Figure 18: Algorithm MAXDEG

high vertex degree are strategically positioned so that their incident edges begin
their embedded paths along different dimensions of the hypercube. When the graph
to be embedded is a tree, a greedy algorithm is a natural method for accomplishing
this.

Algorithm MAXDEG, presented in Figure 18, begins at the tree's vertex of
highest degree, and spans the entire tree by perpetually embedding the neighbors
of the embedded node with the most unembedded neighbors. When an embedded
node's neighbor is to be embedded, the hypercube address assigned to the neighbor
is the unassigned address which is closest to the embedded node's address (i.e., the
number of bit differences is minimized).

In order to gauge the success of this algorithm, we performed Pascal simulations
of random communications and computations performed by the nodes of a random
tree embedded into the hypercube by means of MAXDEG. One hundred different
random trees were analyzed; each tree had between 65 and 512 nodes and was em-
bedded into the appropriate hypercube to guarantee optimal expansion. In order to eliminate the skewing of our results, the vertex degree of the tree was bounded by the dimension of the hypercube into which it was embedded. In this way, high vertex degree was eliminated as a source of guaranteed congestion difficulties. As with our earlier pyramid embedding simulations, exponential distributions were employed for both the transmission time and the gap between individual processors' transmissions. Each simulation lasted ten thousand time units. Our results appear in Table 5 for both bitwise left-to-right static and dynamic routing.

Table 5: MAXDEG Simulation Results

<table>
<thead>
<tr>
<th>ave. comm. time</th>
<th>ave. comm. gap</th>
<th>STATIC ROUTING</th>
<th>DYNAMIC ROUTING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ratio of # delays to # messages</td>
<td>ave. delay per message</td>
<td>ratio of # delays to # messages</td>
</tr>
<tr>
<td>max.</td>
<td>ave.</td>
<td>max.</td>
<td>ave.</td>
</tr>
<tr>
<td>4 64</td>
<td>1.287%</td>
<td>0.291%</td>
<td>0.01233</td>
</tr>
<tr>
<td>4 128</td>
<td>0.792%</td>
<td>0.155%</td>
<td>0.00552</td>
</tr>
<tr>
<td>4 256</td>
<td>0.401%</td>
<td>0.081%</td>
<td>0.00383</td>
</tr>
<tr>
<td>4 512</td>
<td>0.169%</td>
<td>0.038%</td>
<td>0.00129</td>
</tr>
</tbody>
</table>

Analysis of this table indicates that MAXDEG yields excellent results for general trees. On average, only a fraction of one percent of the communications encounter delays under this embedding, with that fraction cut in half for the dynamic routing scheme. The low $O(nk)$ time complexity of this algorithm (where $n$ is the number of tree vertices and $k$ is the maximum vertex degree) also suggests that this heuristic approach is advantageous.
Given graph $G$ to be embedded in hypercube $H$.

$x \leftarrow$ vertex in $G$ with maximum vertex degree  
(In case of ties, select the one  
with the largest subtree);  
$\text{label}(x) \leftarrow 0$;  
repeat until \{labelled vertices\} = $V(G)$  
  $x \leftarrow$ labelled vertex in $G$ with maximum  
number of unlabelled neighbors  
(In case of ties, select the one  
with the largest subtree);  
  for each unlabelled neighbor $y$ of $x$ do  
  $\text{label}(y) \leftarrow$ unused label closest to $\text{label}(x)$

Figure 19: Algorithm MAXDEG/MAXSUBTREE

Algorithm MAXDEG contains no specific mechanism for handling ties; if several embedded nodes have an equal number of unembedded neighbors, one is arbitrarily chosen to have its neighbors embedded. Examination of the specific trees for which MAXDEG performed the best or the worst revealed that when ties were decided in favor of the node with the largest subtree, then maximum congestion was significantly lower than when nodes with smaller subtrees were selected. Intuitively, such an approach improves the chances for key nodes to be strategically embedded. Algorithm MAXDEG/MAXSUBTREE, presented in Figure 19, represents this modification to MAXDEG. Its time complexity is also $O(nk)$.

Performing simulations of MAXDEG/MAXSUBTREE on the same group of trees for which MAXDEG was simulated yielded the results in Table 6. Comparison of the results for the two algorithms reveals no perceptible superiority of one
Table 6: MAXDEG/MAXSUBTREE Simulation Results

<table>
<thead>
<tr>
<th>ave. comm.</th>
<th>ave. comm. gap</th>
<th>STATIC ROUTING</th>
<th></th>
<th></th>
<th>DYNAMIC ROUTING</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ave. comm. time</td>
<td>ratio of # delays to # messages</td>
<td>ave. delay per message</td>
<td>max. ave.</td>
<td></td>
<td>ratio of # delays to # messages</td>
<td>ave. delay per message</td>
<td>max. ave.</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>1.326%</td>
<td>0.295%</td>
<td>0.01048</td>
<td></td>
<td>0.637%</td>
<td>0.156%</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>0.724%</td>
<td>0.158%</td>
<td>0.00543</td>
<td></td>
<td>0.332%</td>
<td>0.084%</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>0.330%</td>
<td>0.084%</td>
<td>0.00309</td>
<td></td>
<td>0.211%</td>
<td>0.040%</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>0.193%</td>
<td>0.039%</td>
<td>0.00138</td>
<td></td>
<td>0.139%</td>
<td>0.022%</td>
</tr>
</tbody>
</table>

Table 7: Maximum Congestion of Greedy Simulations

<table>
<thead>
<tr>
<th>Maximum Congestion</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXDEG</td>
<td>14%</td>
<td>62%</td>
<td>21%</td>
<td>1%</td>
<td>2%</td>
</tr>
<tr>
<td>MAXDEG/MAXSUBTREE</td>
<td>18%</td>
<td>65%</td>
<td>14%</td>
<td>2%</td>
<td>1%</td>
</tr>
</tbody>
</table>

algorithm over the other.

However, Table 7 illustrates that the maximum congestion resulting from our modified algorithm is noticeably lower than before. The fact that the two algorithms performed comparably well in our simulations serves to support our thesis that while unit congestion is certainly appealing theoretically, merely maintaining low congestion is frequently satisfactory in practice. Note that in the case of each algorithm, 97% of the trees were embedded with maximum congestion three or less.

Chen and Gehringer [16] have developed a different greedy algorithm for embedding graphs into hypercubes. This algorithm, presented in Figure 20 as MINDIL, stresses the minimization of dilation by iteratively selecting the next vertex to be
Given graph $G$ to be embedded in hypercube $H$.

repeat until $\{\text{labelled vertices}\} = V(G)$
  \text{hightotal} \leftarrow -1;
  \text{for } x \text{ in } V(G)-\{\text{labelled vertices}\} \text{ do}
    \text{for } l \text{ in } \{\text{unused hypercube labels}\} \text{ do}
      \text{newtotal} \leftarrow 0;
      \text{for } y \text{ in } \{\text{labelled vertices}\} \text{ do}
        \text{if } y \text{ in } \{\text{neighbors of } x\} \text{ then}
          \text{newtotal} \leftarrow \text{newtotal} + (\text{number of} \\
          \text{bits in which label}(y) \\
          \text{and } l \text{ are identical});
        \text{if } \text{newtotal} > \text{hightotal} \text{ then}
          \text{hightotal} \leftarrow \text{newtotal};
          \text{bestx} \leftarrow x;
          \text{bestl} \leftarrow l
      \text{endif;}
    \text{label}(\text{bestx}) \leftarrow \text{bestl}
  \text{endif;}
\text{labelled and the hypercube address with which it will be labelled. This selection}
\text{is based upon the resultant path lengths that such a labelling will cause.}

Simulating MINDIL with the same collection of trees previously used yielded
the results in Table 8 and Table 9. Note that while these results are quite satis-
factory, they do prove inferior to the more tree-oriented greedy solutions proposed
earlier. As expected, the emphasis on dilation with MINDIL is less effective when
dealing with the direct-connect hypercube technology than the emphasis on con-
gestion with MAXDEG and MAXDEG/MAXSUBTREE.
Table 8: MINDIL Simulation Results

<table>
<thead>
<tr>
<th>ave. comm. time</th>
<th>ave. comm. gap</th>
<th>STATIC ROUTING</th>
<th>DYNAMIC ROUTING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max.</td>
<td>ave.</td>
<td>max.</td>
</tr>
<tr>
<td></td>
<td># delays to</td>
<td>delay per</td>
<td># delays to</td>
</tr>
<tr>
<td></td>
<td># messages</td>
<td>message</td>
<td># messages</td>
</tr>
<tr>
<td>4 64</td>
<td>1.440%</td>
<td>0.423%</td>
<td>0.01562</td>
</tr>
<tr>
<td>4 128</td>
<td>0.894%</td>
<td>0.222%</td>
<td>0.00776</td>
</tr>
<tr>
<td>4 256</td>
<td>0.554%</td>
<td>0.117%</td>
<td>0.00390</td>
</tr>
<tr>
<td>4 512</td>
<td>0.289%</td>
<td>0.054%</td>
<td>0.00197</td>
</tr>
</tbody>
</table>

Table 9: Maximum Congestion of MINDIL Simulations

<table>
<thead>
<tr>
<th>Maximum Congestion</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINDIL</td>
<td>4%</td>
<td>46%</td>
<td>43%</td>
<td>7%</td>
</tr>
</tbody>
</table>

4.1.2 Partitioning

An alternative heuristic approach to embedding graphs in the hypercube has been
the recursive partitioning of the graph into subgraphs to be embedded in separate
subcubes. Kernighan and Lin [31] have developed a good heuristic for balancing a
partition by repeatedly trading vertices between the two partitioned halves of the
graph until such transactions no longer improve the partition’s balance. Ercal, et
al., [23] and, more recently, Nazief [39] have recommended using this method for
embedding graphs into hypercubes, using the number of edges connecting the two
halves of the partition (the ‘cut’) as the gauge by which satisfactory balancing is
determined. This heuristic involves the iterative bipartitioning of the graph into
halves with a minimum cut, assigning different values to the current bit position.
of nodes in the different halves, and then recursively repeating the process on each half. However, previous analyses of this technique have been restricted to mesh-like graphs for which such an approach is very likely to yield positive results.

In this section, we examine the merits of utilizing this heuristic when embedding general trees into the hypercube. Its time complexity, $O(nd)$, where $d$ is the hypercube dimension and $n$ is the number of tree vertices, is comparable to that of the greedy methods in the previous section.

The random trees used in our greedy simulations were embedded into the hypercube using this partitioning method. Our results appear in Tables 10 and 11.

With this algorithm, we have also managed to maintain relatively low values for maximum congestion, but the lack of any unit-congestion embeddings and the

---

**Table 10: Tree Partitioning Simulation Results**

<table>
<thead>
<tr>
<th></th>
<th>ave. comm.</th>
<th>ave. comm.</th>
<th>STATIC ROUTING</th>
<th>DYNAMIC ROUTING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>gap</td>
<td>ratio of # delays to # messages</td>
<td>ratio of # delays to # messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>max.</td>
<td>ave.</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>1.865%</td>
<td>1.125%</td>
<td>0.04093</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>1.111%</td>
<td>0.593%</td>
<td>0.02074</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>0.883%</td>
<td>0.292%</td>
<td>0.01054</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>0.489%</td>
<td>0.165%</td>
<td>0.00598</td>
</tr>
</tbody>
</table>

**Table 11: Maximum Congestion of Partitioning Simulations**

<table>
<thead>
<tr>
<th>Maximum Congestion</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioning</td>
<td>0%</td>
<td>49%</td>
<td>51%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>
Table 12: Arbitrary Tree Embedding Simulation Results

<table>
<thead>
<tr>
<th>ave. comm. time</th>
<th>ave. comm. gap</th>
<th>STATIC ROUTING</th>
<th>DYNAMIC ROUTING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ratio of # delays to # messages</td>
<td>ratio of # delays to # messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>max.</td>
<td>ave.</td>
</tr>
<tr>
<td>4 64</td>
<td>18.649%</td>
<td>14.674%</td>
<td>0.65157</td>
</tr>
<tr>
<td>4 128</td>
<td>10.580%</td>
<td>8.224%</td>
<td>0.32684</td>
</tr>
<tr>
<td>4 256</td>
<td>5.902%</td>
<td>4.328%</td>
<td>0.15961</td>
</tr>
<tr>
<td>4 512</td>
<td>3.036%</td>
<td>2.166%</td>
<td>0.07765</td>
</tr>
</tbody>
</table>

Table 13: Maximum Congestion of Arbitrary Embedding Simulations

<table>
<thead>
<tr>
<th>Maximum Congestion</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbit. Embedding</td>
<td>15%</td>
<td>46%</td>
<td>31%</td>
<td>7%</td>
<td>1%</td>
</tr>
</tbody>
</table>

A significant increase in congestion-three embeddings helps to explain the fact that this approach yields substantially less satisfactory results than our greedy methods.

Although trees lend themselves to bipartitioning based upon minimum cuts, the relative lack of edges in trees tends to counteract the primary advantage of the partitioning heuristic: edges joining vertices on opposite sides of the $i$th-dimensional partition have the same $(d - i - 1)$-bit prefix and, consequently, a greater chance of ultimately being embedded onto hypercube nodes with reasonable proximity.

4.1.3 Analysis

Both approaches used in this section have yielded good results for embedding general trees into hypercubes. Such results are hardly inevitable, however, as an analysis of the results of arbitrary embeddings for our set of random trees illustrates
in Table 12 and Table 13.

Although the partitioning heuristic approach was successful, our results indicate that the greedy methods, especially MAXDEG and MAXDEG/MAXSUBTREE, take greater advantage of the nature of the tree structure and supply us with superior results. In both cases, however, our simulations reveal that low congestion values are often quite sufficient to yield satisfactory levels of communication performance.

In the next section, we explore the effect that these heuristic approaches have on the problem of embedding general graphs into the hypercube.

4.2 General Graph Embeddings

In this section, the greedy and partitioning heuristics of the previous section are applied to the problem of embedding general graphs into hypercubes. We mentioned earlier that guaranteeing that such an embedding will have unit congestion is an intractable problem. Our previous results indicate, however, that maintaining low congestion may cause no significant communication delays. Should either of these heuristics accomplish that feat, we will have taken an important step towards the implementation of parallel algorithms on the hypercube.

4.2.1 Greedy Results

The MAXDEG algorithm in the previous section may be directly applied to a general graph. It will create a spanning subtree of the graph, with each step based
Table 14: MAXDEG Simulation Results on General Graphs

<table>
<thead>
<tr>
<th>ave. comm. time</th>
<th>ave. comm. gap</th>
<th>STATIC ROUTING</th>
<th>DYNAMIC ROUTING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ratio of # delays to # messages</td>
<td>ratio of # delays to # messages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max. ave. delay message</td>
<td>max. ave. delay per message</td>
<td>max. ave. delay per message</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>12.167% 8.186% 0.33705</td>
<td>5.756% 3.906% 0.13426</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>6.828% 4.400% 0.16765</td>
<td>3.018% 2.094% 0.07413</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>3.917% 2.348% 0.08591</td>
<td>1.645% 1.048% 0.03647</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>1.819% 1.090% 0.04957</td>
<td>0.859% 0.488% 0.01774</td>
</tr>
</tbody>
</table>

Table 15: Maximum Congestion of MAXDEG Simulations on General Graphs

<table>
<thead>
<tr>
<th>Maximum Congestion</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXDEG</td>
<td>10%</td>
<td>14%</td>
<td>34%</td>
<td>21%</td>
<td>14%</td>
<td>7%</td>
</tr>
</tbody>
</table>

on the embedded node with the most unembedded neighbors.

Simulations of the computations and communications executed by the nodes of a MAXDEG-embedded graph were performed as before. One hundred random graphs were used, with the same restrictions on vertex degree and graph size that we used in our study of trees. The results are recorded in Table 14 and Table 15.

The disappointing results of these simulations indicate that the greedy approach works far better on general trees than on general graphs. This is hardly surprising when one considers the nature of this heuristic; its creation of the spanning tree relies upon the assignment of hypercube addresses based upon the address of a single embedded neighbor. With trees, that approach was successful since only one neighbor of the newly embedded node could have already been assigned an
Table 16: MINDIL Simulation Results on General Graphs

<table>
<thead>
<tr>
<th>ave. comm. time</th>
<th>ave. comm. gap</th>
<th>STATIC ROUTING</th>
<th>DYNAMIC ROUTING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ratio of # delays to # messages</td>
<td>ratio of # delays to # messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>max. ave.</td>
<td>del per message</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>6.524%</td>
<td>4.704%</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>3.807%</td>
<td>2.536%</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>1.910%</td>
<td>1.315%</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>1.223%</td>
<td>0.679%</td>
</tr>
</tbody>
</table>

Table 17: Maximum Congestion of MINDIL Simulations on General Graphs

<table>
<thead>
<tr>
<th>Maximum Congestion</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINDIL</td>
<td>7%</td>
<td>48%</td>
<td>31%</td>
<td>10%</td>
<td>4%</td>
</tr>
</tbody>
</table>

address. With general graphs, a node may be assigned an address based upon the address of one neighbor, but resulting in significant communication problems with any other neighbors which have already been embedded.

Simulating the performance of the same collection of random graphs embedded using the MINDIL algorithm of the previous section yields the results in Table 16 and Table 17. This approach represents a significant improvement over the tree-based MAXDEG algorithm, but still yields an unsatisfactory proportion of delayed communications.

4.2.2 Partitioning Results

The partitioning heuristic presented earlier for trees may also be directly applied to general graphs. The fact that this approach does not rely on spanning trees is
Table 18: Partitioning Simulation Results on General Graphs

<table>
<thead>
<tr>
<th>ave. comm.</th>
<th>ave. comm. gap</th>
<th>STATIC ROUTING ratio of # delays to # messages per message</th>
<th>DYNAMIC ROUTING ratio of # delays to # messages per message</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>64</td>
<td>5.767%</td>
<td>3.888%</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>3.034%</td>
<td>2.076%</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>1.560%</td>
<td>1.045%</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>0.978%</td>
<td>0.483%</td>
</tr>
</tbody>
</table>

Table 19: Maximum Congestion of Partitioning Simulations on General Graphs

<table>
<thead>
<tr>
<th>Maximum Congestion</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioning</td>
<td>10%</td>
<td>38%</td>
<td>45%</td>
<td>7%</td>
</tr>
</tbody>
</table>

cause for optimism. The results of performing simulations using this algorithm on the same set of random graphs for which MAXDEG was simulated are reported in Table 18 and Table 19.

These results reveal a tremendous improvement over MAXDEG, with delay problems occurring with approximately half the frequency under the partitioning method. In addition, partitioning yields a significantly lower proportion of delayed communications than the dilation-based MINDIL greedy method. Admittedly, the delays experienced using this approach are hardly as small as when the partitioning heuristic is applied to trees or meshes, but the increased complexity of the graphs being embedded here permits us to consider these results to be signs of success.

The reversal of the relative merits of the two heuristic approaches when applied
to general graphs instead of general trees is noteworthy. The reason that partitioning is not as successful as our greedy approaches on general trees is precisely the reason that it has more success when applied to general graphs. Specifically, the size of minimum cuts in graph partitionings will be larger than the minimum cut size in tree partitionings, thus taking advantage of the partitioning method’s mechanism for embedding vertices on opposite sides of the partition onto hypercube nodes with close proximity.

4.2.3 Analysis

The partitioning heuristic is superior to both greedy heuristics when applied to general graphs. Its lack of dependence on spanning trees and the tendency of general graphs to make better use of its proximity mechanism help to explain this reversal of the relative preference of algorithms from that encountered with general trees.

As expected with general graphs, neither algorithm performs in a superlative manner. The wide variety of incidence relations that can be encountered when dealing with general graphs certainly inhibits the development of an excellent heuristic. In view of this, our results indicate that the partitioning heuristic is recommended for the embedding of general algorithmic structures into the hypercube.
4.3 Summary

In this chapter, we have departed from our previous efforts to embed specific classes of graphs into the hypercube and have turned to the problem of embedding general trees and graphs. The intractability of such a general problem has dictated the use of heuristics and a reliance on statistical results to evaluate our success.

Greedy methods have shown themselves to be superior for the general tree embedding problem. A greedy algorithm's basis in the tree structure has the effect of yielding very low, often unit, congestion. Two tree-based greedy methods were analyzed for general trees and one was found to produce significantly lower maximum congestion values than the other. However, since each algorithm consistently yielded extremely low congestion values, they performed equally well in our simulations. This fact helps to verify our thesis that unit congestion is not essential to good communication performance as long as low congestion can be supplied.

We also examined a partitioning heuristic for embedding graphs into the hypercube. Its performance on general trees was acceptable, although not as impressive as the performance of the tree-based greedy methods. Of much greater significance was the fact that this partitioning approach was quite successful when employed on general graphs, especially when one considers the complexity of such a task. Until the unlikely event of the development of an extremely low congestion general embedding technique, the partitioning approach will serve admirably.
CHAPTER V

Conclusions

In this thesis, we have examined the problems associated with implementing a parallel algorithm on the hypercube by means of graphically embedding the underlying topological structure of the algorithm into the hypercube structure. In view of the recent changes in hypercube technology and the intractability of many of the general embedding problems, we have reached several conclusions and can make a few recommendations for further research.

The development of the direct-connect hypercube technology, in which path length has negligible effect on communication efficiency, has enabled us to de-emphasize dilation as a gauge of the efficiency of an embedding algorithm. Instead, we stress the substantially more complicated congestion gauge. Contention for communication paths can potentially cause serious delays in the execution of parallel algorithms.

The ideal solution to this traffic problem is to develop embedding algorithms which have unit congestion. Unfortunately, this solution is hardly practical; even
relatively simple graphs like pyramids and, in fact, any graphs with high vertex degree, are difficult to embed into optimal-expansion hypercubes with unit congestion. One of our most important results is the revelation that unit congestion is not essential to efficient communication performance; our pyramid algorithms and the application of our greedy approaches to trees have enabled us to conclude that small maximum congestion values are often quite sufficient to yield acceptably low levels of communication delay.

The rigorous mathematical analysis required to prove the effectiveness of our quadtree and pyramid algorithms attests to the difficulty of embedding algorithms into the hypercube when their underlying structure does not lend itself easily to the hypercube technology. Without the symmetry of these classes of graphs and their bounded vertex degree, developing a verifiably efficient embedding appears to be an insurmountable problem. We conclude, then, that attempts to develop direct embedding approaches such as those we developed for quadtrees and pyramids, and those previously developed for the simpler binary tree and rectangular mesh structures, should be confined to similarly regular classes of graphs.

Embedding graphs of a less symmetrical, more skewed topology appears to be best approached from the heuristic angle. Fortunately, our studies have indicated that the two types of heuristics presented in this thesis yield quite satisfactory results. The greedy heuristics which we examined yielded extremely efficient embeddings for general trees, while the partitioning heuristic, which was dependent
on no particular structural configuration, performed admirably on general graphs. The development of additional heuristics to improve the general graph embedding results is certainly an area in which we heartily recommend further research.

Our alternative approach to task allocation on the hypercube, in which we avoided merely assigning separate tasks to distinct subcubes and, instead, embedded multiple graphs into a single hypercube, proved very successful for quadtrees and pyramids. The increased utilization of processors and the low values of the various embedding cost gauges serve as strong incentives for us to advocate additional investigation into this approach.

We conclude now with a few brief laudatory remarks about the hypercube structure and its performance as a multiprocessor configuration. The symmetric properties of the hypercube permit easy embedding of many traditional classes of algorithmic topologies, but, as our studies have shown, the relative ease in which more general graphs are embedded attests to the versatility of the hypercube. The dramatic improvement in communication performance that occurred when we considered dynamic routing techniques also indicates a potential capacity for adaptation to heavy message traffic. All in all, our research serves to enhance the reputation of the hypercube multiprocessor as an extremely effective means of implementing parallel algorithms.
Bibliography


