COMPUTER SCIENCE PUBLICATION

PARALLELIZED SIMULATION OF GRIDS BY HYPERCUBES

M.Y. Chan and F.Y.L. Chin

Technical Report TR-90-11

October 1990

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF ENGINEERING
UNIVERSITY OF HONG KONG
POKFULAM ROAD
HONG KONG

PARALLELIZED SIMULATION OF GRIDS BY HYPERCUBES

M Y Chan[1]
James Capel (Far East) Limited
Hong Kong

Francis Chin[2]
Department of Computer Science
University of Hong Kong
Hong Kong

ABSTRACT

This paper parallelizes the embedding strategy for mapping any two dimensional grid into its optimal hypercube with minimal dilation The parallelization allows each hypercube node to independently determine, in constant time which grid node it will simulate and the communication paths it will take to reach the hypercube nodes which simulate its grid neighbors The paths between grid neighbors are chosen in such a way as to curb the congestion at each hypercube node and across each hypercube edge

KEYWORDS embedding, hypercube, simulation, grid, dilation, parallel algorithm, congestion

---

[1] This research was done while M Y Chan was an Assistant Professor in the Computer Science Program at The University of Texas at Dallas Richardson, Texas 75083 and visiting the University of Hong Kong She is currently associated with James Capel (Far East) Limited a subsidiary of the Hong Kong and Shanghai Banking Corporation

[2] Research supported in part by the ONR grant N00014 87 K-0833 This research was done while Francis Chin was visiting the Computer Science Program The University of Texas at Dallas, Richardson Texas 75083

1. INTRODUCTION

Hypercube multiprocessor machines are commercially available today [S]. One of the key features of the hypercube is a rich interconnection structure which permits many important network topologies, such as grids, to be efficiently simulated. A *binary hypercube of dimension n* or *binary n-cube* can be thought of as an undirected graph of 2^n nodes labeled 0 to 2^n-1 in binary; two nodes are connected by an edge if and only if their labelings differ in exactly one bit position. A number of important algorithms run well on two-dimensional grids of processors [D, ESS]. To simulate a grid on the hypercube, nodes of the grid must be mapped to hypercube nodes [BMS, C1, C2, C3, CC, G, LS]. In this paper, we are interested in the efficient simulation of any two-dimensional grid by *its optimal hypercube*, the smallest hypercube with at least as many nodes as the grid.

Chan [C3] has given a method of mapping the nodes of any two-dimensional grid to the nodes of its optimal hypercube, so that the mapping is one-to-one and the *dilation* (the worst case distance between grid-neighbors in the hypercube) is minimal (at most 2). Unfortunately, Chan's method is deficient in two aspects: (1) the algorithm is sequential in nature, thus not all processors in the hypercube participate in the computation of the embedding, and (2) the algorithm fails to consider how grid-neighbors can communicate in the hypercube and *ad hoc* communication may lead to heavy congestion in the hypercube.

In this paper, we parallelize the embedding algorithm in [C3] for mapping any two-dimensional grid to its optimal hypercube. Initially, the size of the grid to be simulated is broadcasted to each node in the hypercube. With our parallel algorithm, each hypercube node, when given the size of the grid, can then determine in constant time which node of the grid it will simulate. Moreover, the hypercube node can also determine the communication paths it will take to reach the hypercube nodes which simulate its grid-neighbors. When a pair of grid-neighbors are distanced by two edges in the hypercube, communication must go through an intermediate node in the hypercube. Our parallel algorithm will ensure that intermediate nodes are chosen in such a way that each hypercube node will act as an intermediate node for at most two pairs of grid-neighbors, thus curbing the *congestion* at each hypercube node and across each hypercube edge.

The paper is organized in the following manner. Section 2 reviews Chan's embedding strategy and outlines useful properties. Section 3 describes how each hypercube node independently determines which grid node it will simulate in constant time. Section 4 addresses the determination of the communication path between hypercube nodes which simulate grid-neighbors. Section 5 concludes.

2. REVIEW OF THE EMBEDDING STRATEGY

Over 61% of all two-dimensional grids can be embedded into their optimal hypercubes with a dilation of 1 (i.e. all grid-neighbors are also neighbors in the hypercube) by using binary-reflected Gray codes [SS]. For the other over 38% of all two-dimensional grids, which require at least dilation 2 [BS], Chan's method [C3] can be applied to give the best possible dilation of 2. In this section, we briefly review Chan's method.

Suppose we are given an $\alpha \times \beta$ grid G. Let $\bar{\alpha} = 2^{\lfloor \log_2 \alpha \rfloor}$ and $\bar{\beta} = 2^{\lfloor \log_2 \beta \rfloor}$.

The binary-reflected Gray code strategy of [SS] already embeds G into its optimal hypercube with dilation 1 when $\alpha\beta > 2\hat{\alpha}\hat{\beta}$, or $\alpha = \hat{\alpha}$ or $\beta = \hat{\beta}$ (i.e. α or β is a power of two). For this reason, we are only interested in the case where $\alpha\beta \leq 2\hat{\alpha}\hat{\beta}$, $\alpha \neq \hat{\alpha}$ and $\beta \neq \hat{\beta}$. Thus, it can be assumed, without loss of generality, that $\alpha \leq 3\hat{\alpha}/2$. (Either $\alpha \leq 3\hat{\alpha}/2$ or $\beta \leq 3\hat{\beta}/2$; for otherwise, $\alpha\beta > 9\hat{\alpha}\hat{\beta}/4 > 2\hat{\alpha}\hat{\beta}$.)

With $\alpha\beta \leq 2\hat{\alpha}\hat{\beta}$, the objective is to label each node of the grid with a unique $(\lfloor\log_2\alpha\rfloor + \lfloor\log_2\beta\rfloor+1)$-bit binary number, which effectively names the node in the optimal $(\lfloor\log_2\alpha\rfloor + \lfloor\log_2\beta\rfloor+1)$-cube to which it is mapped. Since dilation 2 is needed, the labels for grid-neighbors are allowed to differ in at most 2 bit positions. We will use the 11×11 grid, whose optimal hypercube is the 7-cube, as a running example throughout the rest of this section.

We first show how a dilation-4 embedding can be achieved. With a shift of labels, we arrive at a dilation-3 embedding. Finally, the last bit of each label is reassigned in order to arrive at a dilation-2 embedding.

2.1. Dilation-4 Embedding

To achieve a dilation-4 embedding, the label for each node of the grid is determined in two stages: (1) the initial $\lfloor\log_2\alpha\rfloor$ bits of the label for each node, and then (2) the final $\lfloor\log_2\beta\rfloor+1$ bits of the label. To arrive at the first $\lfloor\log_2\alpha\rfloor$ bits, the nodes of the grid are systematically partitioned into $\hat{\alpha}$ groups, which are called "chains", of at most $2\hat{\beta}$ nodes each. The partitioning will be done so that grid-neighbors will be assigned to either the same or adjacent chains. Thus, in the case of a 11×11 grid (Figure 2.1), we will partition the grid into 8 groups, chains 1 through 8, of ≤ 16 nodes each, and if a node is assigned to, say, chain 3, then its grid-neighbors can only belong to chains 2, 3 or 4. Nodes belonging to the same chain will be given the same bits as the first $\lfloor\log_2\alpha\rfloor$ bits of its label. We shall show in the later part of this section that, by the use of binary-reflected Gray code, the first $\lfloor\log_2\alpha\rfloor$ bits given to nodes of chain i will differ from the first $\lfloor\log_2\alpha\rfloor$ bits given to nodes of chain

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 2 & 1 & 2 & 2 & 1 & 2 & 2 & 1 & 2 & 1 \\
2 & 2 & 2 & 2 & 3 & 2 & 2 & 3 & 2 & 2 & 2 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
4 & 4 & 3 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 3 \\
4 & 5 & 4 & 4 & 5 & 4 & 5 & 5 & 4 & 5 & 4 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 & 5 & 5 & 5 \\
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
7 & 7 & 6 & 7 & 7 & 6 & 7 & 7 & 7 & 7 & 6 \\
7 & 8 & 7 & 7 & 8 & 7 & 7 & 8 & 7 & 8 & 7 \\
8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8
\end{bmatrix}$$

Figure 2.1 An example of CHAIN assignment for an 11 x 11 grid

i+1 or i−1 by exactly one bit. Hence, the first $\lfloor \log_2 \alpha \rfloor$ bits of the labels assigned to grid-neighbors will differ in at most one bit position.

The partitioning can be described formally by a *partitioning matrix* $A(\alpha,\beta)$, or simply A, an integer matrix comprised of 1's and 2's having $\hat{\alpha}$ rows and β columns, i.e. an $\hat{\alpha} \times \beta$ matrix. The idea is that $a_{i,j}$ (i.e. the element in the i^{th} row, j^{th} column of matrix A) indicates how many nodes from column j of grid G will belong to chain i. So, if $a_{i,j} = 2$, the i^{th} chain "doubles up" at the j^{th} column. Because we wish to partition G into $\hat{\alpha}$ chains, matrix A has $\hat{\alpha}$ rows, and because there are β columns in grid G, matrix A has β columns. Let us now define matrix A. Matrix A has as its first column the vector

$$\begin{bmatrix} a_{1,1} \\ a_{2,1} \\ a_{3,1} \\ a_{4,1} \\ \vdots \\ a_{\hat{\alpha},1} \end{bmatrix} = \begin{bmatrix} \lceil \alpha/\hat{\alpha} \rceil \\ \lfloor \alpha/\hat{\alpha} \rfloor \\ \lfloor 2\alpha/\hat{\alpha} \rfloor - \lfloor \alpha/\hat{\alpha} \rfloor \\ \lfloor 3\alpha/\hat{\alpha} \rfloor - \lfloor 2\alpha/\hat{\alpha} \rfloor \\ \vdots \\ \lfloor (\hat{\alpha}-1)\alpha/\hat{\alpha} \rfloor - \lfloor (\hat{\alpha}-2)\alpha/\hat{\alpha} \rfloor \end{bmatrix}$$

Precisely, for all $1 \le i \le \hat{\alpha}$, $a_{i,1} = \lfloor (i-1)\alpha/\hat{\alpha} \rfloor - \lfloor (i-2)\alpha/\hat{\alpha} \rfloor$, which is either 1 or 2. Hence, for the 11 × 11 grid, the first column vector is

$$\begin{bmatrix} 2 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 2 \\ 1 \end{bmatrix}$$

The entire matrix A is based on a cyclic shift of the first column, i.e., for all $1 \le i < \hat{\alpha}$ and $1 \le j < \beta$, $a_{i,j} = a_{i+1,j+1}$ and $a_{\hat{\alpha},j} = a_{1,j+1}$. In general, we have for all $1 \le i \le \hat{\alpha}$ and $1 \le j \le \beta$, $a_{i,j} = a_{(i-j) \bmod \hat{\alpha}+1,1}$:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \cdots & a_{1,\beta} \\ a_{2,1} & a_{2,2} \cdots & a_{2,\beta} \\ a_{3,1} & a_{3,2} \cdots & a_{3,\beta} \\ . & . & . \\ . & . & . \\ . & . & . \\ a_{\hat{\alpha},1} & a_{\hat{\alpha},2} \cdots & a_{\hat{\alpha},\beta} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{\hat{\alpha},1} & a_{\hat{\alpha}-1,1} \cdots \\ a_{2,1} & a_{1,1} & a_{\hat{\alpha},1} \cdots \\ a_{3,1} & a_{2,1} & a_{1,1} \cdots \\ . & . & . \\ . & . & . \\ . & . & . \\ a_{\hat{\alpha},1} & a_{\hat{\alpha}-1,1} & a_{\hat{\alpha}-2,1} \cdots \end{bmatrix}$$

Thus, for the 11 × 11 grid, matrix A is the following 8 × 11 matrix:

$$\begin{bmatrix} 2 & 1 & 2 & 1 & 1 & 2 & 1 & 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 2 & 1 & 1 & 2 & 1 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 & 1 & 2 & 1 & 1 & 2 \\ 2 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 2 \\ 2 & 1 & 1 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 1 & 2 & 1 & 1 & 2 & 1 & 2 & 1 \end{bmatrix}$$

The reader can easily verify that the row sum of each row in matrix A for the 11×11 grid is no more than 16, i.e., there are ≤ 16 nodes belonging to each of the 8 chains. Moreover, the column sum of each column in matrix A for the 11×11 grid is 11, meaning each node within each column of the grid is assigned to some chain. In general, there will be no more than 2β nodes in each chain and each column sums to α. The formal proofs of these facts are given in [C3].

Let $[x,y]$ denote the node in row x, column y of the grid G where $1 \leq x \leq \alpha$ and $1 \leq y \leq \beta$. In order to determine the first $\lfloor \log_2 \alpha \rfloor$ bits of each grid node $[x,y]$'s label, one has to find out to which chain $[x,y]$ belongs. Let $CHAIN[x,y]$ denote the chain to which node $[x,y]$ belongs. Figure 2.1 gives the *CHAIN* values given to each node of the 11×11 grid. In general, *CHAIN* values can be derived from the partition matrix A in the following manner:

$$CHAIN[x,y] = k \quad \text{iff} \quad COLSUM(y;k-1) < x \leq COLSUM(y;k) \tag{1}$$

where $COLSUM(j;k) = \sum_{i=1}^{k} a_{i,j}$, the partial column sum from row 1 to row k of the j^{th} column elements of A[3]. It can be proved [C3] that grid-neighbors are given *CHAIN* values that differ by at most 1. This fact is obvious for vertical grid-neighbors. Based on properties of partition matrix A, the partial column sums from row 1 to row k of any two columns of elements in A should be almost equal, i.e., $| COLSUM(j;k) - COLSUM(j';k) | \leq 1$ for any $1 \leq j,j' \leq \beta$, $1 \leq k \leq \alpha$. Thus, the *CHAIN* values of horizontal grid-neighbors also differ by at most 1.

Let $GRAY(t,p)$ denote the $((p-1) \bmod 2^t + 1)^{th}$ element of the t-bit binary-reflected Gray code sequence (see [RND] for a definition of binary reflected Gray code). For example, $GRAY(3,4) \equiv 010$ since 010 is the 4^{th} element of (000,001,011,010,110,111,101,100). Let the first $\lfloor \log_2 \alpha \rfloor$ bits of the label given to node $[x,y]$ of G be $GRAY(\lfloor \log_2 \alpha \rfloor, CHAIN[x,y])$. Thus, we have the following property.

Property 1. The first $\lfloor \log_2 \alpha \rfloor$ bits of the labels given to grid-neighbors differ in at most 1 bit position.

To determine the last $\lfloor \log_2 \beta \rfloor + 1$ bits, the following convention is adopted to join together nodes belonging to the same chain: two nodes in the same column belonging to the same chain are joined

[3] *COLSUM* can be readily computed using the following formula:

$$COLSUM(j;k) = \begin{cases} \lfloor (\alpha-j+k)\alpha/\alpha \rfloor - \lfloor (\alpha-j)\alpha/\alpha \rfloor & \text{if } 1 \leq k < j \\ \lfloor (\alpha-1)\alpha/\alpha \rfloor - \lfloor (\alpha-j)\alpha/\alpha \rfloor + \lceil \alpha/\alpha \rceil + \lfloor (k-j)\alpha/\alpha \rfloor & \text{if } 1 \leq j \leq k \end{cases}$$

by a line segment, and the topmost node in column j belonging to chain i is joined to the bottommost node in column $j+1$ belonging to the same chain i. The line segments of any chain can only be between horizontal grid-neighbors (i.e. $[x,y]$ and $[x,y+1]$), be between vertical grid-neighbors (i.e. $[x,y]$ and $[x+1,y]$), or slope down at most one row from $[x,y]$ to $[x+1,y+1]$. The nodes of each chain are then numbered sequentially starting at 1 proceeding along the line segments. $NUMBER[x,y]$ denotes the number given to node $[x,y]$. Figure 2.2(a) shows the line segments of each chain and the $NUMBER$ values given to each node of the 11×11 grid. In general, for each node $[x,y]$ of G,

$$NUMBER[x,y] = ROWSUM(CHAIN[x,y];y-1) + \delta[x,y] \tag{2}$$

where $ROWSUM(i;k) = \sum_{j=1}^{k} a_{i,j}$, the partial row sum from column 1 to column k of the i^{th} row elements of A[4],

and $\delta[x,y] = \begin{cases} 2 & \text{if } CHAIN[x+1,y] = CHAIN[x,y] \\ 1 & \text{otherwise.} \end{cases}$

(Note that $1 \le NUMBER[x,y] \le 2\beta$.)

Since the partition matrix has the property that the partial row sum from column 1 to column k of any two rows in A are almost equal, i.e., $|\, ROWSUM(i;k) - ROWSUM(i';k) \,| \le 1$ for $1 \le i,i' \le \alpha$, $1 \le k \le \beta$, the $NUMBER$ values of grid-neighbors differ by at most 3. Replacing $NUMBER$ values by corresponding binary-reflected Gray code values, i.e. $GRAY(\lfloor \log_2 \beta \rfloor + 1, NUMBER[x,y])$, the last

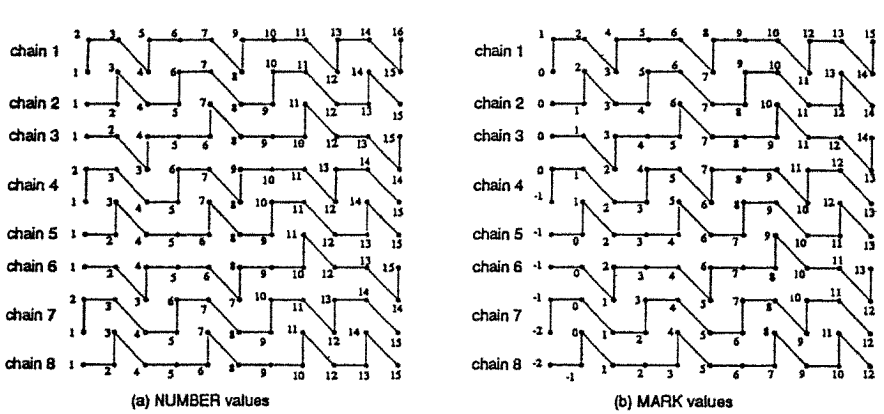


FIGURE 2.2 Example of NUMBER and MARK values for an 11 x 11 grid

[4] Let $k = p\alpha + q$ where p,q are integers and $0 \le q < \alpha$. $ROWSUM$ can be readily computed using the following formula:

$$ROWSUM(i;k) = \begin{cases} p\alpha + \lfloor (i-1)\alpha/\alpha \rfloor - \lfloor (i-1-q)\alpha/\alpha \rfloor & \text{if } 0 \le q < i \le \alpha \\ p\alpha + \lfloor (i-1)\alpha/\alpha \rfloor + \lceil \alpha/\alpha \rceil + \lfloor (\alpha-1)\alpha/\alpha \rfloor - \lfloor (\alpha-q+i-1)\alpha/\alpha \rfloor & \text{if } 1 \le i \le q < \alpha \end{cases}$$

$\lfloor \log_2 \beta \rfloor + 1$ bits of grid-neighbors will differ in at most 3 bit positions. Together with Property 1, we have a dilation-4 embedding.

2.2. Dilation-3 Embedding

Close study of Figure 2.2(a) reveals that the difference of 3 in *NUMBER* values occurs when grid-neighbors are on different chains. For example, nodes [5,3] and [5,4] are grid-neighbors which belong to adjacent chains whose *NUMBER* values differ by at most 3. In Figure 2.2(a), notice that *by subtracting one* from each of the *NUMBER* values associated with nodes of chain 4, chain 4 can be "synchronized" with chain 3 in the sense that if node $[x+1,y]$ belonged to chain 4 and node $[x,y]$ (the node above $[x+1,y]$) belonged to chain 3, they would then have the same *NUMBER* value. This synchronization allows the *NUMBER* values of grid-neighbors to differ by at most 2. With synchronization, nodes [5,3] and [5,4] will have *NUMBER* values that no longer differ by 3 but rather by 2. Essentially, synchronization can be achieved by adding an appropriate offset to each node's *NUMBER* value to result in a new MARK value, i.e.

$$MARK[x,y] = CHAIN[x,y] - COLSUM(1;CHAIN[x,y]) + NUMBER[x,y]. \tag{3}$$

Figure 2.2(b) gives the *MARK* values for the 11×11 grid. The reader can easily verify that, for the 11×11 grid, horizontal grid-neighbors have *MARK* values which differ by at most 2, while vertical grid-neighbors have *MARK* values which differ by at most 1. In general, we can show that this is true for any grid. So, by replacing *CHAIN* values and *MARK* values by corresponding binary-reflected Gray code, we can ensure that the first $\lfloor \log_2 \alpha \rfloor$ bits of the labels assigned to grid-neighbors will differ in at most one bit position, and the last $\lfloor \log_2 \beta \rfloor + 1$ bits of the labels assigned to horizontal grid-neighbors will differ in at most two bit positions and to vertical grid-neighbors will differ in at most one bit position. Because binary-reflected Gray code is used, if the last $\lfloor \log_2 \beta \rfloor + 1$ bits of the labels for two (horizontal) grid-neighbors differ by two bit positions, they must differ in the last bit position. Thus, omitting the last bit, we have the following.

Property 2. The first $\lfloor \log_2 \alpha \rfloor + \lfloor \log_2 \beta \rfloor$ bits of the labels assigned to horizontal grid-neighbors will differ in at most two bit positions (one from the first $\lfloor \log_2 \alpha \rfloor$ bits of the label and the other from the next $\lfloor \log_2 \beta \rfloor$ bits of the label), whereas vertical grid-neighbors will differ in at most one bit position.

In view of this observation, to achieve a dilation-2 embedding, the strategy is to modify the last bit.

2.3. Dilation-2 Embedding

The last bit of each node is modified so that

(a) two nodes with the same first $\lfloor \log_2 \alpha \rfloor + \lfloor \log_2 \beta \rfloor$ bits will differ in their last bit, and

(b) two (horizontal) grid-neighbors with first $\lfloor \log_2 \alpha \rfloor + \lfloor \log_2 \beta \rfloor$ bits which differ in exactly two bit positions will have the same last bit.

By design, this strategy forces a dilation of at most two. As it turns out, this kind of assignment of the last bit can always be accomplished. To this end, a dependency graph $G' = (V,E)$ is constructed which has as its nodes the nodes of G, i.e. $V(G') = \{[x,y] \mid 1 \le x \le \alpha, 1 \le y \le \beta\}$. There is an edge between two nodes in G' iff either they have the same first $\lfloor \log_2 \alpha \rfloor + \lfloor \log_2 \beta \rfloor$ bits, or they are grid-

neighbors whose first $\lfloor \log_2 \alpha \rfloor + \lfloor \log_2 \beta \rfloor$ bits differ in exactly two bit positions. In the former case, the edge is said to be a *twin* edge, while in the latter case, the edge is said to be a *critical* edge. An edge between nodes $[x,y]$ and $[u,v]$ indicates that the last bit assigned to $[x,y]$ will affect the last bit assigned to $[u,v]$, and vice versa, based on the idea that distinct labels are assigned to different nodes and grid-neighbors are within dilation 2 of each other. Certainly, if G' is acyclic, then the appropriate assignment of the last bit can be made. Traverse G' by using either a breadth-first or depth-first tree traversal scheme. When each node T is visited for the first time, its last bit is determined. Suppose T's parent in the breadth-first or depth-first tree is S. If there is a twin edge between S and T, then make T's last bit different from S's. If, on the other hand, there is a critical edge between S and T, then make T's last bit the same as S's. It has been indeed shown that G' will always be acyclic [C3]. Figure 2.3 shows G' for the 11 × 11 grid.

Definitions: An edge between nodes $[x,y]$ and $[x,y+1]$ is said to be *horizontal*. An edge between nodes $[x,y]$ and $[x+1,y]$ is said to be *vertical*. An edge between nodes $[x,y]$ and $[x+1,y+1]$ is said to *slope down*. An edge between nodes $[x,y]$ and $[x',y']$ where $y = 1$ and $y' = \beta$ is said to *wrap around*. □

Property 3: Critical edges in G' are horizontal.

Property 4: Twin edges in G' either slope down, wrap around, are vertical or are horizontal.

Let us now look at the overall *sequential* time complexity for the dilation-2 embedding strategy. Computing the $\alpha \times \beta$ matrix $A(\alpha,\beta)$ takes $O(\alpha\beta)$ time since each element of A can be computed in constant time. Determining the first $\lfloor \log_2 \alpha \rfloor$ bits of the labels for all nodes $[x,y]$, $1 \le x \le \alpha$ and $1 \le y \le \beta$, takes $O(\alpha\beta)$ time since $CHAIN[x,y]$ for all nodes $[x,y]$ can be computed in $O(\alpha\beta)$ time.

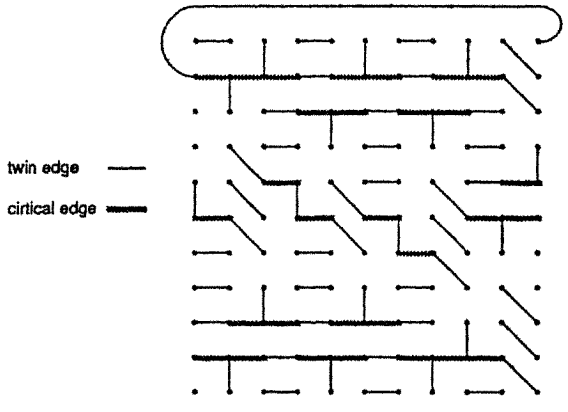


FIGURE 2.3 Dependency graph G' for an 11 x 11 grid

Determining the last $\lfloor \log_2 \beta \rfloor + 1$ bits of the labels for all nodes $[x,y]$ also takes $O(\alpha\beta)$ time since $MARK[x,y]$ for all nodes $[x,y]$ can be computed in $O(\alpha\beta)$ time, and modifying the last bit for all nodes $[x,y]$ takes $O(\alpha\beta)$ time since the construction and traversal of G', which has $\alpha\beta$ nodes and $O(\alpha\beta)$ edges, take $O(\alpha\beta)$ time. Thus, we have the following.

Fact: Any two-dimensional $\alpha \times \beta$ grid can be embedded into its optimal hypercube with a dilation of at most 2 in $O(\alpha\beta)$ time.

2.4. Column Splicing

Throughout the rest of this paper, we assume that β is odd. If β were even, we can easily construct a dilation-2 embedding for an $\alpha \times \beta$ grid by splicing 2^k copies of a dilation-2 embedding an $\alpha \times \beta'$ grid, where β' is odd and $2^k\beta' = \beta$ in the following manner.

The β columns of the $\alpha \times \beta$ grid are partitioned into β' blocks with 2^k columns each. The i^{th} block of the 2^k columns will have the same last $\lceil \log_2 \alpha\beta' \rceil$ bits as the i^{th} column of nodes in the $\alpha \times \beta'$ grid. The j^{th} column of nodes in each block, say the i^{th} block, will have either $GRAY(k,j)$ or $GRAY(k,2^k+1-j)$ as its first k bits depending on whether i is odd or even.

Example: $\beta = 12 = 2^2 3$, i.e. $k = 2$ and $\beta' = 3$. Assume that the labels of the $\alpha \times \beta'$ grid for a particular row are (a,b,c). Thus the node labels for the same row of the $\alpha \times \beta$ grid are $(00a,01a,11a,10a,10b,11b,01b,00b,00c,01c,11c,10c)$ □

Since the first k bits of the labels are the same for any two vertical grid neighbors in the $\alpha \times \beta$ grid, any two vertical grid neighbors in the $\alpha \times \beta$ grid are distanced from each other by the same amount (i.e. dilation ≤ 2) as in the $\alpha \times \beta'$ grid. As far as horizontal grid neighbors in the $\alpha \times \beta$ grid are concerned, if the two horizontal grid neighbors are from the same block, their labels only differ in the first k bits by one position (dilation = 1 from the binary-reflected Gray code property); if the two horizontal grid neighbors are from two different blocks, they should have the same first k bits in their labels and are distanced from each other by the same amount as the $\alpha \times \beta'$ grid (i.e. dilation ≤ 2). Thus, the dilation for the $\alpha \times \beta$ grid remains ≤ 2 after column splicing of the labels.

3. NODE-TO-NODE MAPPING

In this section, we shall give an algorithm by which each hypercube node can determine the grid node, if any, it will simulate according to the dilation-2 embedding strategy described in Section 2. Parallelizing the dilation-3 strategy is relatively straightforward. The difference between the dilation-3 and dilation-2 strategy lies in the determination of the last bit of each node's label, and herein lies the difficulty in parallelizing the dilation-2 strategy. Recall that the last bits can be determined via the traversal of the grid's dependency graph and the traversal is sequential in nature. So, we shall first parallelize the dilation-3 strategy, and then introduce some simple rules for determining the last bit for the dilation-2 strategy which avoids traversing the dependency graph in a sequential manner and makes possible the parallelization of the dilation-2 strategy.

We first introduce two useful functions g and h related to the dilation-3 embedding strategy. Function g is such that $g<C,M> = [x,y]$ if the node with mark value M in chain C is node $[x,y]$ of the grid, and function h is such that $h[x,y] = <C,M>$ if node $[x,y]$ of the grid is the node with mark

value M in chain C. Both functions can be computed using constant time; see Figures 3.1 and 3.2. The derivations of functions g and h come directly from Equations (1), (2) and (3) of Section 2.

Example 3.1: Suppose we wanted to know which node of the 11 × 11 grid has mark value 8 and is in chain 5. Applying the g function, we could conclude that the number value N for this node is $M - C + COLSUM(1;5) = 8\text{-}5\text{+}7 = 10$. Since $ROWSUM(5;7) = 10$, y is 7. With $a_{5,7} = 2$ and $N = 10 \geq ROWSUM(5;7)$, $\Delta = 1$ and $x = COLSUM(7;4) + \Delta = 5\text{+}1 = 6$. Thus, the node with mark value 8 in chain 5 is $[x,y] = [6,7]$ (as can be verified in Figure 2.2(b)). □

Example 3.2: Suppose we wanted to know what chain and mark value is assigned to node [8,4] of the 11 × 11 grid. Using function h as defined in Figure 3.2, the chain C is 6, $\Delta = 1$, $N = ROWSUM(6;3)+\Delta = 4\text{+}1 = 5$ and $M = C-COLSUM(1;6)+N = 6\text{-}8\text{+}5 = 3$. Thus, node [8,4] is given a mark value of 3 and is in chain 6 (as can be verified in Figure 2.2(b)). □

function $g\langle C,M\rangle$:

(1) compute $N = (M - C + COLSUM(1;C) - 1) \bmod 2\beta + 1$

(2) if not $(1 \leq C \leq \alpha)$ or not $(1 \leq N \leq ROWSUM(C;\beta))$ then return *undefined*

(3) let y be such that $ROWSUM(C;y-1) < N \leq ROWSUM(C;y)$
note: y is either $\lfloor N\,\alpha/\alpha \rfloor$ or $\lceil N\,\alpha/\alpha \rceil$

(4) let $x = COLSUM(y;C-1) + \Delta$
where $\Delta = \begin{cases} 2 & \text{if } a_{C,y} = 2 \text{ and } N < ROWSUM(C;y) \\ 1 & \text{otherwise} \end{cases}$

(5) return $[x,y]$

Figure 3.1 Function to find grid position from chain number and mark value

function $h[x,y]$:

(1) if not $(1 \leq x \leq \alpha)$ or not $(1 \leq y \leq \beta)$ then return *undefined*

(2) let C be such that $COLSUM(y;C-1) < x \leq COLSUM(y;C)$
note: C is either $\lfloor x\,\alpha/\alpha \rfloor$ or $\lceil x\,\alpha/\alpha \rceil$

(3) compute $N = ROWSUM(C;y-1) + \Delta$
where $\Delta = \begin{cases} 2 & \text{if } a_{C,y} = 2 \text{ and } x < COLSUM(y;C) \\ 1 & \text{otherwise} \end{cases}$

(4) let $M = C - COLSUM(1;C) + N$

(5) return $\langle C,M\rangle$

Figure 3.2 Function to find chain number and mark value from grid position

With the help of function g, we actually have a scheme for parallelizing the dilation-3 embedding strategy. For example, if we wish to know which node in the 11 × 11 grid the hypercube node 0010011 simulates based on the dilation-3 embedding strategy, we need only evaluate $g<2,3>$, since the first three bits of the label, 001, encode the chain value of 2 and the last four bits, 0011, encodes the mark value of 3. (Recall $GRAY(3,2) = 001$ and $GRAY(4,3) = 0011$.) Since $g<2,3> = [3,3]$, node 0010011 simulates the node in the 3^{rd} row, 3^{rd} column in the grid.

Notice there are cases where function g returns *undefined*. For example, if we consider hypercube node 0011001, $g<2,15>$ would need to be computed (since $GRAY(3,2) = 001$ and $GRAY(4,15) = 1001$); *undefined* would be returned, and thus, hypercube node 0011001 would not simulate any grid node.

Function h is used to identify critical edges and twin edges. Figure 3.3 gives the functions *Critical* and *Twin* which, using h, determines whether or not an edge is, respectively, critical and twin.

The difference between the dilation-3 and dilation-2 strategy is the step which modifies the last bit. This step can be carried out by a traversal of the acyclic dependency graph G'. To efficiently parallelize this step, we avoid such a traversal by introducing a set of simple rules for determining the last bit of some nodes in accordance with G'. Rule E1 defines the last bit for nodes that are endpoints of critical edges. For twin edges with the last bit of one endpoint defined, Rule E2 will define the last bit of the other endpoint. A node whose last bit is not redefined by Rules E1 and E2 will retain the last bit given to it by the dilation-3 strategy.

Rule E1: If $\{[x,y],[x,y+1]\}$ is a critical edge (Property 3 in Section 2) and either

(a) $\{[x,y],[x-1,y]\}$ is a twin edge, or

function *Twin*($[x,y],[u,v]$):

(1) if either $h[x,y]$ or $h[u,v]$ is undefined then return *false*

(2) let $<C_{xy},M_{xy}> = h[x,y]$ and $<C_{uv},M_{uv}> = h[u,v]$

(3) if $C_{xy} = C_{uv}$ and $\lceil M_{xy}/2 \rceil = \lceil M_{uv}/2 \rceil$, then return *true* else return *false*

function *Critical*($[x,y],[u,v]$):

(1) if either $h[x,y]$ or $h[u,v]$ is undefined then return *false*

(2) let $<C_{xy},M_{xy}> = h[x,y]$ and $<C_{uv},M_{uv}> = h[u,v]$

(3) if $C_{xy} \neq C_{uv}$ and $\lceil M_{xy}/2 \rceil \neq \lceil M_{uv}/2 \rceil$, then return *true* else return *false*

Figure 3.3 Functions to determine whether an edge is critical or twin

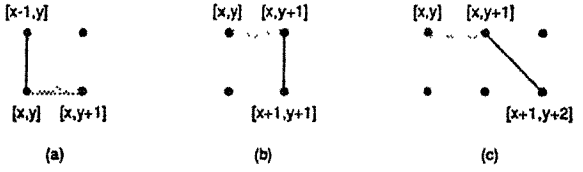
(b) $\{[x,y+1],[x+1,y+1]\}$ is a twin edge, or

(c) $\{[x,y+1],[x+1,y+2]\}$ is a twin edge,

then the last bit for both $[x,y]$ and $[x,y+1]$ is y *mod* 2, otherwise the last bit for both $[x,y]$ and $[x,y+1]$ is $(y+1)$ *mod* 2.

Rule E2: If $\{[x,y],[x',y']\}$ is a twin edge and $[x,y]$ is assigned a last bit by Rule E1, assign to $[x',y']$ a last bit different from $[x,y]$.

Figure 3.4 summarizes Rule E1 pictorially. Figure 3.5 shows the last bit values for nodes adjacent to critical edges in the 11×11 grid as determined by Rule E1, while Figure 3.6 shows the last bit values for all of the nodes affected by Rule E1 and E2.



The last bit for both [x,y] and [x,y+1] of the above cases are y mod 2.

Figure 3.4 Summary of Rule E1

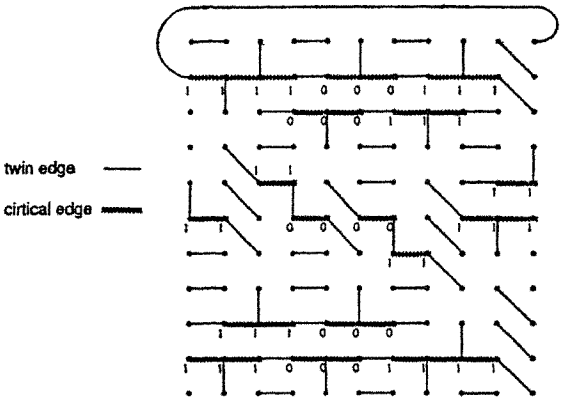


FIGURE 3.5 Last bit assignment according to Rule E1

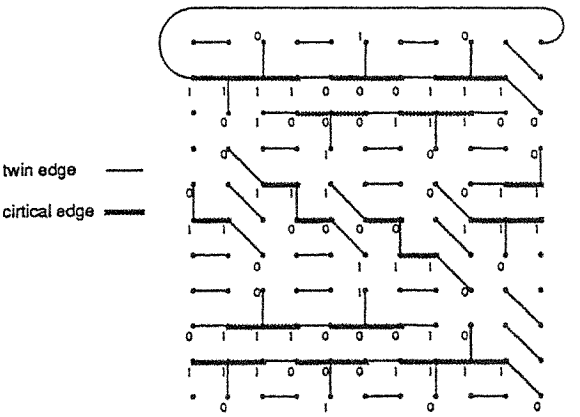


FIGURE 3.6 Last bit assignment according to Rules E1 and E2

Notice that no conflicts arise in applying Rule E1. For example, {[2,1],[2,2]} is a critical edge and {[2,2],[3,2]} is a twin edge. Applying Rule E1 with respect to this critical edge (condition (b) is satisfied) will yield 1 *mod* 2 = 1 as the last bit for both [2,1] and [2,2]. At the same time, notice that {[2,2],[2,3]} is a critical edge. Applying Rule E1 with respect to this edge (none of the conditions are satisfied) will yield (2+1) *mod* 2 = 1 as the last bit for both [2,2] and [2,3]. In both cases, 1 is assigned as the last bit of [2,2]. In general, there will be no conflicts in applying Rule E1.

Lemma 3.1: Suppose {[x,y−1],[x,y]} and {[x,y],[x,y+1]} are two critical edges. Applying Rule E1 with respect to each of these critical edges will cause the same last bit to be assigned to node [x,y].

Proof of Lemma 3.1: See Appendix. □

Note also that {[2,3],[2,4]} is a critical edge and {[2,3],[1,3]} is a twin edge. Thus, applying Rule E1 with respect to {[2,3],[2,4]} (condition (a) is satisfied) yields 3 *mod* 2 = 1 assigned to both [2,3] and [2,4]. At the same time, {[2,5],[2,6]} is a critical edge and applying Rule E1 with respect to this edge (none of the conditions are satisfied) yields (5+1) *mod* 2 = 0 assigned to both [2,5] and [2,6]. Thus, 1 is assigned as the last bit of [2,4] while 0 is assigned to [2,5], and {[2,4],[2,5]} is a twin edge. So, Rule E1 assignments do not conflict with Rule E2 which requires that the endpoints of each twin edge be assigned different last bits. In general, we have the following.

Lemma 3.2: Suppose {[x,y−1],[x,y]} and {[x',y'],[x',y'+1]} are critical edges and {[x,y],[x',y']} is a twin edge. Then, Rule E1 will assign different last bits to [x,y] and [x',y'].

Proof of Lemma 3.2: See Appendix. □

We are now ready to describe the algorithm by which each hypercube node can determine the grid node, if any, it will simulate according to the dilation-2 strategy. Suppose we are given a

hypercube node S whose label is uvw where u, v and w are bitstrings of length $\lfloor \log_2 \alpha \rfloor$, $\lfloor \log_2 \beta \rfloor$ and 1, respectively. Let S_0 denote the hypercube node whose label is $uv0$ and S_1 denote the hypercube node whose label is $uv1$. First, we compute the grid nodes $[x_0,y_0]$ and $[x_1,y_1]$ simulated by S_0 and S_1, respectively, in the dilation-3 strategy using function g. S will simulate either grid node $[x_0,y_0]$ or $[x_1,y_1]$ according to the dilation-2 strategy, and $\{[x_0,y_0],[x_1,y_1]\}$ is a twin edge. Next, we check if one of these two grid nodes is adjacent to a critical edge; if so, we can use Rule E1 and Rule E2 to determine the last bits of their labels, and thus which grid node of the two S will simulate. If neither of the two grid nodes is adjacent to a critical edge, then we assume S_0 will simulate grid node $[x_0,y_0]$ and S_1 will simulate $[x_1,y_1]$. Figure 3.7 gives the details of the algorithm.

Example 3.4: Suppose we wish to know which node in the 11 × 11 grid the hypercube node 0010011 simulates based on the dilation-2 strategy. We would first compute the grid nodes simulated by 0010010 and 0010011 in the dilation-3 strategy: 0010011 would simulate $g\langle 2,4\rangle = [3,4]$ and 0010010 would simulate $g\langle 2,3\rangle = [3,3]$. Twin edge $\{[3,3],[3,4]\}$ is adjacent to critical edge $\{[3,4],[3,5]\}$ (see Figure 3.6). Since $\{[3,5],[4,5]\}$ is a twin edge, by Rule E1, nodes [3,4] and [3,5] will be assigned 4 *mod* 2 = 0 as their last bits. Hence, 0010010 will simulate [3,4] and 0010011 will simulate [3,3] in the dilation-2 strategy. □

Theorem 3.1: Using Rules E1 and E2 to re-define the labels given by the dilation-3 strategy, we can embed any any grid into its optimal hypercube with a dilation of at most 2.

Proof of Theorem 3.1: What we need to prove is that applying Rules E1 and E2 will ensure that
(a) different last bits are assigned to the endpoints of twin edges, and
(b) the same last bit is assigned endpoints of critical edges.

Lemmas 3.1 and 3.2 guarantee that by using Rules E1 and E2 to redefine (the last bit of) labels given by the dilation-3 strategy will result in each node being assigned a unique last bit, i.e. either 0 or 1 but not both. Rule E1, by definition, ensures that the endpoints of critical edges are assigned the same last bit. Rule E2, by definition, ensures that the endpoints of twin edges, having one endpoint assigned a last bit by Rule E1, are assigned different last bits. The endpoints of all other twin edges, i.e. those whose endpoints are not redefined by Rule E1, were given different last bits by the dilation-3 strategy. Thus, both (a) and (b) are satisfied. □

Theorem 3.2: The dilation-2 embedding of any grid into its optimal hypercube can be accomplished in constant parallel time.

Proof on Theorem 3.2: Consider function *grid* in Figure 3.7. Recall that functions g and h (Figures 3.1 and 3.2) only take constant time, making functions *Twin* and *Critical* also computable in constant time. Thus, function *Assign* (Figure 3.7), which incorporates Rule E1, is computable in constant time. Step (5) of function *grid* incorporates Rule E2. So, function *grid*, the algorithm for embedding a grid into its optimal hypercube, can be carried out in constant parallel steps by each hypercube node. □

function *Assign*([*u*,*v*],[*u*,*v*+1]):

if *Twin*([*u*,*v*],[*u*−1,*v*]) or *Twin*([*u*,*v*+1],[*u*+1,*v*+1]) or *Twin*([*u*,*v*+1],[*u*+1,*v*+2])
then return *v* *mod* 2
else return (*v*+1) *mod* 2

function *grid*(*S*):

/* Algorithm for hypercube node S to find [*x*,*y*], the node in the $\alpha \times \beta$ grid mapped to S */

(1) let the label for S be *uvw* where *u*,*v*,*w* are bitstrings of length $\lfloor \log_2 \alpha \rfloor$, $\lfloor \log_2 \beta \rfloor$, 1, respectively.

(2) let C, M_0 and M_1 be such that $GRAY(\lfloor \log_2 \alpha \rfloor, C) = u$, $GRAY(\lfloor \log_2 \beta \rfloor + 1, M_0) = v0$ and $GRAY(\lfloor \log_2 \beta \rfloor + 1, M_1) = v1$

(3) if either $g\langle C, M_0 \rangle$ or $g\langle C, M_1 \rangle$ is undefined then

if $w = 0$ then return $g\langle C, M_0 \rangle$
else return $g\langle C, M_1 \rangle$

(4) compute $[x_0,y_0] = g\langle C, M_0 \rangle$ and $[x_1,y_1] = g\langle C, M_1 \rangle$
note: $\{[x_0,y_0],[x_1,y_1]\}$ is a twin edge and either $[x_0,y_0]$ or $[x_1,y_1]$ will map to S

(5) if *Critical*($[x_0,y_0-1],[x_0,y_0]$) then

if *Assign*($[x_0,y_0-1],[x_0,y_0]$) = w then return $[x_0,y_0]$
else return $[x_1,y_1]$

else if *Critical*($[x_0,y_0],[x_0,y_0+1]$) then

if *Assign*($[x_0,y_0],[x_0,y_0+1]$) = w then return $[x_0,y_0]$
else return $[x_1,y_1]$

else if *Critical*($[x_1,y_1-1],[x_1,y_1]$) then

if *Assign*($[x_1,y_1-1],[x_1,y_1]$) = w then return $[x_1,y_1]$
else return $[x_0,y_0]$

else if *Critical*($[x_1,y_1],[x_1,y_1+1]$) then

if *Assign*($[x_1,y_1],[x_1,y_1+1]$) = w then return $[x_1,y_1]$
else return $[x_0,y_0]$

else if $w = 0$ then return $[x_0,y_0]$ else return $[x_1,y_1]$

Figure 3.7 Function to determine the grid node to be simulated by an hypercube node *S*

4. PATH BETWEEN GRID-NEIGHBORS

After each hypercube node identifies the grid node it will simulate, the next step is to set up communicate paths to the hypercube nodes which simulate its grid neighbors. Function *hypercube*[x,y] will return the label of the hypercube node which simulates grid node [x,y]. Function *dilation*([x,y],[x',y']) will return the dilation (which is either 1 or 2) of the grid edge {[x,y],[x',y']}. Details of the algorithms *hypercube* and *dilation* are given in Figure 4.4.

Assume [x,y] and [x',y'] are two grid neighbors and *dilation*([x,y],[x',y']) = 1, then there is a hypercube edge connecting *hypercube*[x,y] and *hypercube*[x',y'], and the two grid neighbors [x,y] and [x',y'] can communicate directly over the hypercube edge. However, if *dilation*([x,y],[x',y']) = 2, the communication between these two grid neighbors [x,y] and [x',y'] has to go through an intermediate node. In this section, we shall give simple rules by which each hypercube node can determine the communication paths it will take to reach the hypercube nodes which simulate its grid neighbors.

First, some notation needs to be introduced. Let the label given to a grid node [x,y], using the dilation-2 strategy, be divided into three parts <u,v,w> where the lengths of bitstrings u, v and w are, respectively, $\lfloor \log_2 \alpha \rfloor$, $\lfloor \log_2 \beta \rfloor$ and 1. Note that $u = GRAY(\lfloor \log_2 \alpha \rfloor, CHAIN[x,y])$ and $v = GRAY(\lfloor \log_2 \beta \rfloor \lceil MARK[x,y]/2 \rceil)$. Let u_- and u_+ be respectively the bitstring before and the bitstring after u in the binary-reflected Gray code sequence. For example, if $u = 001$, then $u_- = 000$ and $u_+ = 011$.

Lemma 4.1: If <u,v,w> and <u',v',w'> are the labels for two grid-neighbors [x,y] and [x',y'], then

(a) u' may only be u_-, u or u_+,

(b) v' may only be v_-, v or v_+, and

(c) w' may only be w or $\overline{w}$.

Proof of Lemma 4.1: Since $u = GRAY(\lfloor \log_2 \alpha \rfloor, CHAIN[x,y])$, $u' = GRAY(\lfloor \log_2 \alpha \rfloor, CHAIN[x',y'])$ and the *CHAIN* values of any pair of grid-neighbors differ by at most one, u' may only be u_-, u or u_+. Since $v = GRAY(\lfloor \log_2 \beta \rfloor \lceil MARK[x,y]/2 \rceil)$, $v' = GRAY(\lfloor \log_2 \beta \rfloor \lceil MARK[x',y']/2 \rceil)$ and the *MARK* values of any pair of grid-neighbors differ by at most 2, v' may only be v_-, v or v_+. Since w and w' are two single bits, obviously w' may only be w or $\overline{w}$. □

Definition: Let $H(x,y)$ be the Hamming distance between bitstrings x and y. <u',v',w'> is an *intermediate* of the edge between nodes $<u_1,v_1,w_1>$ and $<u_2,v_2,w_2>$ where $H(u_1v_1w_1,u_2v_2w_2) = 2$ iff $H(u_1v_1w_1,u'v'w') = 1$ and $H(u_2v_2w_2,u'v'w') = 1$.

We wish to consider all edges in the grid that suffer a dilation of 2. Let E denote the set of all such edges. For each edge in E, there are exactly two intermediates. For example, <001,001,0> and <000,011,0> are the two intermediates of {<000,001,0>,<001,011,0>}. For each edge in E, we are interested in choosing one of the two intermediates as the *chosen intermediate*. When grid-neighbors N_1 and N_2, distanced by 2 in the hypercube, communicate in the hypercube, they will go through the chosen intermediate for the edge $\{N_1,N_2\}$ in E.

Before we proceed any further about the determination of chosen intermediates for the dilation-2 edges, let us first consider how the assumption of odd β remains valid and how the communicaiion paths will be affected due to column splicing. As in Section 2.4, assume $2^k\beta'=\beta$. Let the labels for the two dilation-2 grid-neighbors in the α x β grid be μr and μt which have the same first k bits, μ, and different last $\lceil \log_2 \alpha\beta' \rceil$ bits, r and t. Then, r and t will be the labels of dilation-2 grid-neighbors in the α x β' grid. If the chosen intermediate for the path between r and t is s, then μs would be the chosen intermediate for μr and μt. Thus, we can still assume an odd β for the remainder of this paper.

Lemma 4.2: Node $<u,v,w>$ can only be the intermediate of eight possible edges in E. These eight edges are depicted in Figure 4.1:

$e_1 = \{<u_-,v,w>,<u,v_-,w>\}$ $\quad$ $e_5 = \{<u,v,\overline{w}>,<u,v_+,w>\}$

$e_2 = \{<u_-,v,w>,<u,v,\overline{w}>\}$ $\quad$ $e_6 = \{<u_+,v,w>,<u,v_-,w>\}$

$e_3 = \{<u_-,v,w>,<u,v_+,w>\}$ $\quad$ $e_7 = \{<u_+,v,w>,<u,v,\overline{w}>\}$

$e_4 = \{<u,v,\overline{w}>,<u,v_-,w>\}$ $\quad$ $e_8 = \{<u_+,v,w>,<u,v_+,w>\}$

Proof of Lemma 4.2: If $<u,v,w>$ is an intermediate for edge $\{N_1,N_2\}$ in E, then both N_1 and N_2 must be at Hamming distance 1 from $<u,v,w>$. There are five nodes at Hamming distance 1 from $<u,v,w>$: $<u_-,v,w>$, $<u_+,v,w>$, $<u,v_-,w>$, $<u,v_+,w>$ and $<u,v,\overline{w}>$. The only edges among these nodes respecting Lemma 4.1 are those depicted in Figure 4.1. □

Lemma 4.3:

(a) e_1, e_2 and e_4 cannot all co-exist in E;

(b) e_2, e_3 and e_5 cannot all co-exist in E;

(c) e_4, e_6 and e_7 cannot all co-exist in E; and

(d) e_5, e_7 and e_8 cannot all co-exist in E.

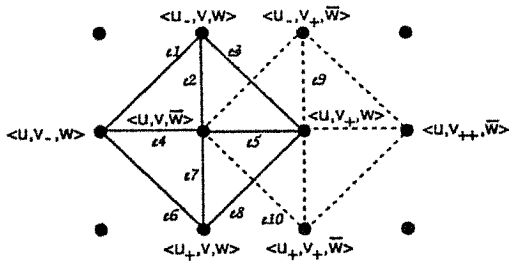


FIGURE 4.1 Edges (solid lines) which can use <u,v,w> as chosen intermediate

Proof of Lemma 4.3: Recall that E, the set of dilation-2 edges, is a subset of the edges in the grid, which does not contain a cycle of length 3. □

Lemma 4.4:

(a) e_1, e_3, e_4 and e_5 cannot all co-exist in E; and

(b) e_4, e_5, e_6 and e_8 cannot all co-exist in E.

Proof of Lemma 4.4: If a pair of nodes with labels $<x,y,z>$ and $<x_+,y',z'>$ are vertical neighbors (i.e., vertical neighbors of different chains), then $y = y'$ (Property 2 in Section 2), i.e. their second components will agree (remember that vertical neighbors belonging to different chains are synchronized). If e_1, e_3, e_4 and e_5 were all to exist in E, then $<u_-,v,w>$, $<u,v_+,w>$, $<u,v,\bar{w}>$ and $<u,v_-,w>$ would be labels of four nodes of some square cycle in the grid. This means that either $<u_-,v,w>$ and $<u,v_+,w>$, or $<u_-,v,w>$ and $<u,v_-,w>$ ought to be vertical neighbors, but they cannot be since their second components do not agree i.e. they do not have the same *MARK* value. The same argument applies for e_4, e_5, e_6 and e_8. □

Lemma 4.5:

(a) e_1 and e_6 cannot both co-exist in E; and

(b) e_3 and e_8 cannot both co-exist in E.

Proof of Lemma 4.5: To prove (a), suppose nodes R, S and T are labelled $<u_-,v,w>$, $<u,v_-,w>$ and $<u_+,v,w>$, respectively. Nodes R and S cannot be vertical grid-neighbors, since, by looking at their labels, we can see that they belong to different chains but they do not have the same *MARK* value. Likewise, S and T cannot be vertical grid-neighbors. So, if e_1 and e_6 were to exist in E, R and T should be horizontal grid-neighbors of S. Thus, the three nodes R, S and T, all on the same row of the grid, belong to three different chains (chain u_-, chain u and chain u_+). This is impossible (see I6 of Appendix). Thus, e_1 and e_6 cannot co-exist in E. The proof of (b) is similar. □

Corollary 4.1: At most four edges among $e_1, e_2, \cdots, e_8$ can co-exist in E.

Proof of Corollary 4.1: Follows from Lemmas 4.3, 4.4 and 4.5. □

Determining chosen intermediates can be formulated as a bipartite graph matching problem instance for which there exists a complete matching. The formulation is as follows. Let V_E and V_H be two disjoint subsets of vertices of a bipartite graph where V_E corresponds to the set of dilation-2 edges and V_H corresponds to the set of hypercube nodes. There is an edge between a vertex $S \in V_E$ and a vertex $T \in V_H$ iff T is an intermediate for the edge in E represented by S. Since each edge in E has exactly two intermediates, the degree for each vertex in V_E of the bipartite graph is two. Since each node of the hypercube can act as the intermediate of at most four edges in E (by Corollary 4.1), the degree of each vertex in V_H is at most 4. By splitting each vertex in V_H into two copies, we can guarantee that each copy of any vertex in V_H is adjacent to at most 2 edges in the bipartite graph, i.e., degree at most 2. The problem is to find a set of edges in the bipartite graph which covers every vertex in V_E (maximum matching problem in bipartite graph), i.e., to find a chosen intermediate for

each dilation-2 edge. In the following, we shall show that this can always be done.

Let V be any subset of V_E and $R(V)$ be the subset of V_H whose vertices are adjacent to the vertices in V. With degree 2 for each vertex in V_E and degree ≤ 2 for each copy of a vertex in V_H, we have $|V| \leq |R(V)|$ and thus a complete matching is ensured [L]. If edge $\{S,T\}$, $S \in V_E$ and $T \in V_H$, is in the matching of the bipartite graph, then T is selected as the chosen intermediate for the edge in E represented by S. In consequence, since there are two copies of each vertex in V_H, each hypercube node will act as the chosen intermediate of at most two dilation-2 edges. Thus, we have the following.

Theorem 4.1: There exists an assignment of chosen intermediates such that each hypercube node can be chosen intermediate of at most two dilation-2 edges. □

Parallelizing this step in determining the chosen intermediates reduces to the parallelization of the complete matching problem. It is still an open problem whether there exists an efficient parallel algorithm for the complete matching problem [P], [KUW]. As it turns out, we can avoid doing matching altogether and determine the chosen intermediates in constant steps by following the set of rules described below for selecting chosen intermediates:

P-Rules:

(P1) Edge $e_1 = \{\langle u_-,v,w\rangle,\langle u,v_-,w\rangle\}$ should use $\langle u,v,w\rangle$ as its chosen intermediate.

(P2) Edge $e_2 = \{\langle u_-,v,w\rangle,\langle u,v,\overline{w}\rangle\}$ should use $\langle u,v,w\rangle$ as its chosen intermediate.

(P3) Edge $e_6 = \{\langle u,v_-,w\rangle,\langle u_+,v,w\rangle\}$ should use $\langle u,v,w\rangle$ as its chosen intermediate.

(P4) Edge $e_4 = \{\langle u,v_-,w\rangle,\langle u,v,\overline{w}\rangle\}$ should use $\langle u,v,w\rangle$ as its chosen intermediate unless both $e_2 = \{\langle u_-,v,w\rangle,\langle u,v,\overline{w}\rangle\}$ and $e_6 = \{\langle u,v_-,w\rangle,\langle u_+,v,w\rangle\}$ are edges in E, whereupon $\langle u,v_-,\overline{w}\rangle$ is used as the chosen intermediate instead.
□

Example: Thus, <000,011,0> would be the chosen intermediate for {<000,001,0>,<001,011,0>} by Rule (P3) with $u = 000$, $v = 011$ and $w = 0$. □

Lemma 4.6: Using the P-Rules, each edge in E is assigned exactly one chosen intermediate.

Proof of Lemma 4.6: From Lemma 4.1, for each edge $\{\langle u,v,w\rangle,\langle u',v',w'\rangle\}$ in E, exactly two of the following pairs of bitstrings (u,u'), (v,v') and (w,w') will be different. If (u,u') and (v,v') are different, exactly one intermediate will be chosen by either Rule (P1) or (P3). If (u,u') and (w,w') are different, exactly one intermediate will be chosen by Rule (P2). If (v,v') and (w,w') are different, exactly one intermediate will be chosen by Rule (P4). It is easy to see that under all cases, exactly one intermediate will be chosen for each edge in E. □

Figure 4.2 shows the five edges, e_1, e_2, e_4, e_5 and e_6, which might potentially use $\langle u,v,w\rangle$ as the chosen intermediate in view of the P-Rules. e_5 uses $\langle u,v,w\rangle$ iff $e_9 = \{\langle u_-,v_+,\overline{w}\rangle,\langle u,v_+,w\rangle\}$ and $e_{10} = \{\langle u,v,\overline{w}\rangle,\langle u_+,v_+,\overline{w}\rangle\}$ are in E (Figure 4.1).

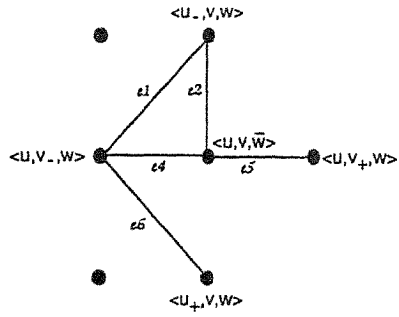


FIGURE 4.2 Edges which can use <u,v,w> as chosen intermediate with application of the P-rules

Lemma 4.7: At most two of the edges e_1, e_2, e_4 and e_6 uses $<u,v,w>$ as their chosen intermediate.

Proof of Lemma 4.7: Follows from the fact that

(a) e_1, e_2 and e_4 cannot co-exist in E (Lemma 4.3(a)),

(b) e_1 and e_6 cannot both co-exist in E (Lemma 4.5(a)), and

(c) e_2, e_4 and e_6 cannot all use $<u,v,w>$ as their chosen intermediates since Rule (P4) prevents this. □

Lemma 4.8: If e_5 uses $<u,v,w>$ as its chosen intermediate, then at most one of the edges e_1, e_2, e_4 and e_6 uses $<u,v,w>$ as their chosen intermediate.

Proof of Lemma 4.8: As observed before, e_5 will use $<u,v,w>$ iff $e_9 = \{<u_-,v_+,\bar{w}>,<u,v_+,w>\}$ and $e_{10} = \{<u,v,\bar{w}>,<u_+,v_+,\bar{w}>\}$ are in E. To induce such a scenario in E (the set of grid edges which suffer a dilation of 2), the situation in the grid must be like the one shown in Figure 4.3. If node $<u,v,\bar{w}>$ is not located in column 1 of the grid, there are two possibilities for the position of $<u,v,w>$ because of the fact that $\{<u,v,w>,<u,v,\bar{w}>\}$ is a twin edge and Property 4 in Section 2, i.e. $\{<u,v,w>,<u,v,\bar{w}>\}$ is either horizontal or sloping downward. These two possibilities give rise to the situations depicted in Figure 4.3. In both of these cases and the case where node $<u,v,\bar{w}>$ is located in column 1 of the grid, $<u,v,\bar{w}>$ cannot be adjacent to $<u_-,v,w>$ or $<u,v_-,w>$, i.e. e_2 and e_4 are not in E. Furthermore, since e_1 and e_6 cannot co-exist in E (Lemma 4.5(a)), at most one of the edges e_1, e_2, e_4 and e_6 will use $<u,v,w>$ as their chosen intermediate. □

Theorem 4.2: Using the P-Rules, each hypercube node will act as the chosen intermediate of at most two edges in E and each chosen intermediate can be determined in constant parallel time.

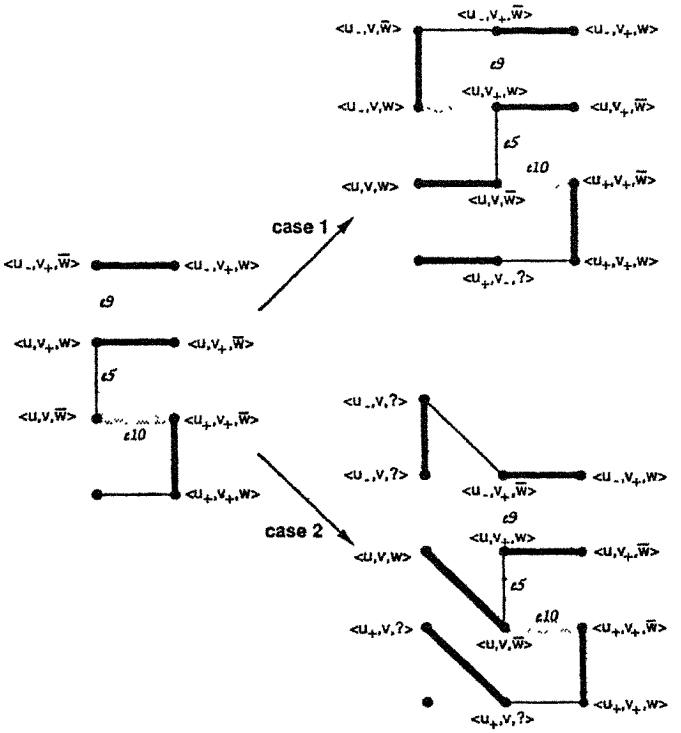


FIGURE 4.3 Scenario for <u,v,w> as the chosen intermediate for e

Proof: From Lemmas 4.7 and 4.8, each hypercube node can at most be chosen intermediates of at most two edges in E. According to Lemma 4.6, by the way the P-Rules are defined, there will be exactly one chosen intermediate assigned for each dilation-2 edge. Function *path* in Figure 4.4 gives the implementation of the P-Rules. Function *path* returns the hypercube node for forwarding a message from grid node $[x,y]$ to its grid neighbor, $[x',y']$. If the communication is over a dilation-2 edge, the chosen intermediate will be returned. Since functions *hypercube* and *grid* take constant time, the algorithm for determining the chosen intermediate (function *path*) will also take constant time for each hypercube node. □

5. CONCLUSION

Based on Chan's embedding strategy [C3], we have derived an efficient parallel embedding algorithm such that each hypercube node, when given the size of the grid, can determine in constant time which node of the grid it will simulate. This algorithm is particularly useful since all hypercube nodes participate in the computation of the embedding.

function *dilation*([x,y],[x',y'])

/* Algorithm to determine the dilation of two neighboring grid nodes, [x,y] and [x',y'], in the dilation-2 mapping of an $\alpha \times \beta$ grid into its optimal hypercube. */

If *hypercube*[x,y] and *hypercube*[x',y'] differs by two bits,
then return 2 else return 1.

function *hypercube*[x,y]

/* Algorithm to find the hypercube node to which the grid node [x,y] is mapped */

(1) let $<C_{xy}, M_{xy}> = h[x,y]$

(2) let $u = GRAY(\lfloor \log_2 \alpha \rfloor, C_{xy})$, $v = GRAY(\lfloor \log_2 \beta \rfloor, M_{xy})$

(3) if $grid(<u,v,0>) = [x,y]$ then return $uv0$
else return $uv1$

function *path*([x,y],[x',y'])

/* Algorithm to determine the chosen intermediate for forwarding a message from grid node [x,y] to grid node [x',y'] */

(1) If *dilation*([x,y],[x',y']) = 1 then return *hypercube*[x',y']

(2) Let uvw = *hypercube*[x,y] and $u'v'w'$ = *hypercube*[x',y']
where $u,u',v,v' \in \{0,1\}^*$ and $w,w' \in \{0,1\}$, $|u| = |u'| = \lfloor \log_2 \alpha \rfloor$ and $|v| = |v'| = \lfloor \log_2 \beta \rfloor$

(3) Note that $u' \in \{u_+, u_-\}, v \in \{v_+, v_-\}, w' \in \{w, \overline{w}\}$

Case: $u' \neq u$ and $v' = v_-$: return $u'vw$
$u' \neq u$ and $v' = v_+$: return $uv'w$
$u' = u_+$ and $w' = \overline{w}$: return $u'vw$
$u' = u_-$ and $w' = \overline{w}$: return uvw'
$v' = v_-$ and $w' = \overline{w}$:

if (($grid(<u_-,v,w'>)$ and $grid(<u,v,w>)$ are grid-neighbors)
and ($grid(<u,v',w'>)$ and $grid(<u_+,v,w'>)$ are grid-neighbors))
then return $uv'w$ else uvw'

$v' = v_+$ and $w' = \overline{w}$:

if (($grid(<u_-,v',w>)$ and $grid(<u,v',w'>)$ are grid-neighbors)
and ($grid(<u,v,w>)$ and $grid(<u_+,v',w>)$ are grid-neighbors))
then return uvw' else $uv'w$

Figure 4.4 Function path to determine the chosen intermediate

Besides node embedding, the second half of the paper deals with path embedding. Each edge in the grid is mapped into a path in the hypercube in parallel in constant time. In fact, each hypercube node can be the chosen intermediate of at most two dilation-2 edges, thus curbing congestion.

There are two types of *congestions* associated with an embedding: node congestion and edge congestion. According to [HJ], the *node congestion* $N_c(v)$ *of a particular hypercube node* v, is defined to be number of paths containing v. Similarly, the *edge congestion* $E_c(e)$ *of an edge* e, is the number of paths containing e. The *node congestion* $N_c(H)$ *of an embedding* H is defined to be the maximum $N_c(v)$ over all nodes v. Likewise, the *edge congestion* $E_c(H)$ *of an embedding* H is defined to be the maximum $E_c(e)$ over all edges e. Since each grid node has at most 4 neighbors and each hypercube node can be the chosen intermediate of at most two dilation-2 edges, $N_c(v) \leq 6$. On the other hand, when the size of the grid to be embedded is slightly less than the size of its optimal hypercube, the number of dilation-2 edges in the embedding may be strictly larger than the number of spare hypercube nodes (i.e. $2^n - \alpha\beta$). Thus some hypercube nodes, besides being involved in the paths to their 4 neighbors, have to be a chosen intermediate of at least one dilation-2 edge, i.e. $N_c(v) \geq 5$, making the node congestion of our embedding one above the optimal. As for edge congestion, since each endpoint of an edge in the hypercube can be the chosen intermediate of at most two dilation-2 edges, each edge in the hypercube can be in at most four paths for dilation-2 edges. This, together with the fact that each edge in the hypercube can act as an edge between two dilation-1 grid-neighbors, gives $E_c(e) \leq 5$, or $E_c(H) \leq 5$, which is four above the optimal.

REFERENCES

[BMS] S. Bettayeb, Z. Miller and I.H. Sudborough, Embedding Grids into Hypercubes, *Proceedings of the 3rd Aegean Workshop on Computing*, 1988.

[BS] J.E. Brandenburg and D.S. Scott, Embeddings of Communication Trees and Grids into Hypercubes, Intel Scientific Computers Report #280182-001, 1985.

[C1] M.Y. Chan, Dilation-2 Embeddings of Grids into Hypercubes, *International Conference on Parallel Processing*, August 1988.

[C2] M.Y. Chan, Embedding of 3-Dimensional Grids into Hypercubes, *International Conference on Hypercubes, Concurrent Computers, and Applications*, March 1989.

[C3] M.Y. Chan, Embedding of Grids into Optimal Hypercubes, *SIAM Journal on Computing*, under second revision.

[CC] M.Y. Chan and F.Y.L. Chin, On Embedding Rectangular Grids in Hypercubes, *IEEE Trans. on Computers C-37*, 1988, pp 1285-1288.

[D] I.S. Duff, Parallel Implementation of Multifrontal Techniques, Technical Memorandum 49, Argonne National Laboratory, 1985.

[ESS] S.C. Eisenstat, M.H. Schultz and A.H. Sherman, Applications of an Element Model for Gaussian Elimination, *Sparse Matrix Computations*, Academic Press, 1976, pp 85-96.

[G] D.S. Greenberg, Optimal Expansion Embeddings of Meshes in Hypercubes, Technical Report YALEU/CSD/RR-535, Dept of Computer Science, Yale University, August 1987.

[HJ] C.T. Ho and S.L. Johnsson, Embedding Meshes in Boolean Cubes by Graph Decomposition, YALEU/DCS/TR-689, Department of Computer Science, Yale University, March 1989.

[KUW] R.M. Karp, E. Upfal and A. Wigderson, Constructing a perfect matching is in random *NC*, *Combinatorica*, Vol 6, 1986, pp 35-58.

[L] C.L. Liu, *Introduction to Combinatorial Mathematics*, McGraw Hill, New York, 1968.

[LS] M. Livingston and Q.F. Stout, Embeddings in Hypercubes, *Proceedings Sixth International Conference on Mathematical Modeling*, August 1987.

[P] N. Pippenger, On Simultaneous Resource Bounds, *Proceeding of the 20th Annual IEEE Symposium on Foundation of Computer Science*, 1979, pp 307-311.

[RND] E.M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms*, Prentice Hall, New York, 1977.

[SS] Y. Saad and M.H. Schultz, Topological Properties of Hypercubes, Research Report 389, Dept. of Computer Science, Yale University, June 1985.

[S] C.L. Seitz, The Cosmic Cube, *CACM 28* , January 1985, pp. 22-33.

APPENDIX

Before proving Lemmas 3 1 and 3.2, we shall prove that the following situations in dependency graph G' are *not* possible.

(I1) If $\{[x,y-1],[x,y]\}$ is a critical edge, then neither $\{[x-1,y-1],[x,y]\}$ nor $\{[x,y-1],[x+1,y]\}$ can be a twin edge (Figure A.1(i)).

(I2) If $\{[x,y-1],[x,y]\}$ and $\{[x+1,y+1],[x+1,y+2]\}$ are critical edges, then $\{[x,y],[x+1,y+1]\}$ cannot be a twin edge (Figure A.1(ii)).

(I3) If $\{[x,y-1],[x,y]\}$ and $\{[x-1,y],[x-1,y+1]\}$ are critical edges, then $\{[x-1,y],[x,y]\}$ cannot be a twin edge (Figure A.1(iii)).

(I4) If $\{[x-1,y-1],[x-1,y]\}$ and $\{[x,y-1],[x,y]\}$ are critical edges, then neither $\{[x-1,y],[x,y]\}$ nor $\{[x-1,y-1],[x,y-1]\}$ can be a twin edge (Figure A.1(iv)).

(I5) If $\{[x,y-1],[x,y]\}$ is a critical edge, then neither of the following cases (Figure A.1(v)) are possible:
- (a) both $\{[x,y-1],[x-1,y-1]\}$ and $\{[x,y],[x-1,y]\}$ are twin edges, nor
- (b) both $\{[x,y-1],[x+1,y-1]\}$ and $\{[x,y],[x+1,y]\}$ are twin edges, nor
- (c) $\{[x,y-1],[x+1,y-1]\}$ and $\{[x,y],[x+1,y+1]\}$ are twin edges.

(I6) Nodes in the same row of a grid cannot belong to more than two different chains.

Proof of (I1), (I3) and (I4) can be found in [C3]. □

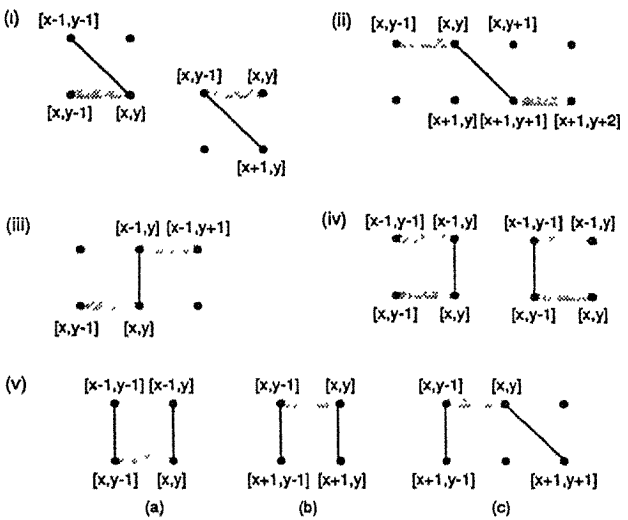


FIGURE A.1 Some impossible situations in dependency graph G'

Proof of (I2): Suppose to the contrary. Assume that $CHAIN[x,y-1] = C'$, $CHAIN[x+1,y+2] = C''$ and both $[x,y]$ and $[x+1,y+1]$ belong to chain C.

Case 1: $C'' = C - 1$: $COLSUM(y;C'') \leq x-1$ and $COLSUM(y+2;C'') \geq x+1$. This violates the fact that $| COLSUM(y;C') - COLSUM(y+2;C') | \leq 1$.

Case 2: $C' = C + 1$: $COLSUM(y-1;C) \leq x-1$ and $COLSUM(y+1;C) = x+1$. This violates the fact that $| COLSUM(y-1;C) - COLSUM(y+1;C) | \leq 1$.

Case 3: $C' = C - 1$ and $C'' = C + 1$: this implies that both $[x+1,y]$ and $[x,y+1]$ belong to chain C, i.e. there are two consecutive columns with two elements belonging to chain C. This contradicts the property of matrix A. □

Proof of (I5): Consider case (a) and suppose to the contrary. Since $\{[x,y-1],[x,y]\}$ is a critical edge, $CHAIN[x,y-1] \neq CHAIN[x,y]$. Furthermore, $COLSUM(y-1;CHAIN[x,y-1]) = COLSUM(y;CHAIN[x,y]) = x$ and $a_{CHAIN[x,y-1],y-1} = a_{CHAIN[x,y],y} = 2$. If $CHAIN[x,y-1] = CHAIN[x,y] + 1$, then $COLSUM(y-1;CHAIN[x,y]) = x-2$. Alternatively if $CHAIN[x,y] = CHAIN[x,y-1] + 1$, then $COLSUM(y;CHAIN[x,y-1]) = x-2$. In the former case, $| COLSUM(y-1;CHAIN[x,y]) - COLSUM(y;CHAIN[x,y]) | = 2$, while in the latter case, $| COLSUM(y-1;CHAIN[x,y-1]) - COLSUM(y;CHAIN[x,y-1]) | = 2$. Both violate the fact that $| COLSUM(j;k) - COLSUM(j';k) | \leq 1$. Proofs for cases (b) and (c) follow a similar argument. □

Proof of (I6): Suppose to the contrary. There must exist two nodes $[x,y]$ and $[x,y']$ on the same row belonging to chains C and C', respectively, where $C' = C - 2$ or $C' = C + 2$. Without loss of generality, let $C' = C - 2$; then, $COLSUM(y;C') \leq x-2$ and $COLSUM(y';C') \geq x$. This contradicts the fact that any two partial column sums of matrix A should be almost equal, i.e. $| COLSUM(y;C') - COLSUM(y';C') | \leq 1$. □

Lemma 3.1: Suppose $\{[x,y-1],[x,y]\}$ and $\{[x,y],[x,y+1]\}$ are two critical edges. Applying Rule 1 with respect to each of these critical edges will cause the same last bit to be assigned to node $[x,y]$.

Proof of Lemma 3.1: Consider the possible positions for $[u,v]$ where $\{[x,y],[u,v]\}$ is a twin edge. Since $\{[x,y-1],[x,y]\}$ and $\{[x,y],[x,y+1]\}$ are critical edges, the only possibilities for $[u,v]$ are $[x-1,y]$, $[x+1,y]$, $[x-1,y-1]$ and $[x+1,y+1]$ (Property 4 in Section 2).

Case i: $[u,v] = [x-1,y]$ (Figure A.2(i))
Rule 1 will assign y *mod* 2 as the last bit of $[x,y]$ and $[x,y+1]$. Based on the critical edge $\{[x,y-1],[x,y]\}$, the only way that $[x,y]$ can be conflictingly assigned $(y-1)$ *mod* 2 is for $\{[x-1,y-1],[x,y-1]\}$ to be a twin edge. But this is impossible in G' (by I5).

Case ii: $[u,v] = [x+1,y]$ (Figure A.2(ii))
Rule 1 will assign $(y-1)$ *mod* 2 as the last bit of $[x,y-1]$ and $[x,y]$. On basis of critical edge $\{[x,y],[x,y+1]\}$, the only way that $[x,y]$ can be conflictingly assigned y *mod* 2 is for either $\{[x,y+1],[x+1,y+1]\}$ or $\{[x,y+1],[x+1,y+2]\}$ to be a twin edge. But this is again impossible in G' (by I5).

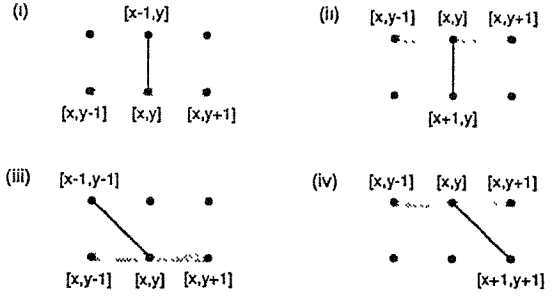


FIGURE A.2 Possible situations for a twin edge attached to the junction of two connecting critical edges

Case iii: $[u,v] = [x-1,y-1]$ (Figure A.2(iii))
This case is impossible in G' (by I1).

Case iv: $[u,v] = [x+1,y+1]$ (Figure A.2(iv))
This case is impossible in G' (by I1). □

Lemma 3.2: Suppose $\{[x,y-1],[x,y]\}$ and $\{[x',y'],[x',y'+1]\}$ are critical edges and $\{[x,y],[x',y']\}$ is a twin edge. Then, Rule 1 will assign different last bits to $[x,y]$ and $[x',y']$.

Proof of Lemma 3.2: The proof is by exhaustion. We shall consider all cases in which two critical edges are connected by a twin edge. There are four cases depending on the orientation of the twin edges.

Case i: The twin edge is a vertical edge.
Because of I3 and I4, the positions of the two critical edges relative to the twin edge must take the form as shown in Figure A.3(i). According to Rule 1, $(y-1)$ *mod* 2 and y *mod* 2 are assigned as the last bits of nodes $[x,y]$ and $[x+1,y]$, respectively. This assignment is consistent since different last bits are given to the endpoints of the twin edge.

Case ii: The twin edge is a horizontal edge.
Figure A.3(ii) gives the only two possible situations for this case. Without loss of generality, let us consider the first situation. According to Rule 1, $(y-1)$ *mod* 2 is assigned to $[x,y-1]$ and $[x,y]$. Since the twin edge connecting $[x,y+2]$ can neither be $\{[x,y+2],[x+1,y+2]\}$ nor $\{[x,y+2],[x+1,y+3]\}$ (otherwise this will violate the fact that the partial column sums of two columns differ by more than 1), according to Rule 1, the last bit assigned to $[x,y+1]$, $[x,y+2]$ should be y *mod* 2. This again leads to a consistent last bit assignment to the endpoints of a twin edge.

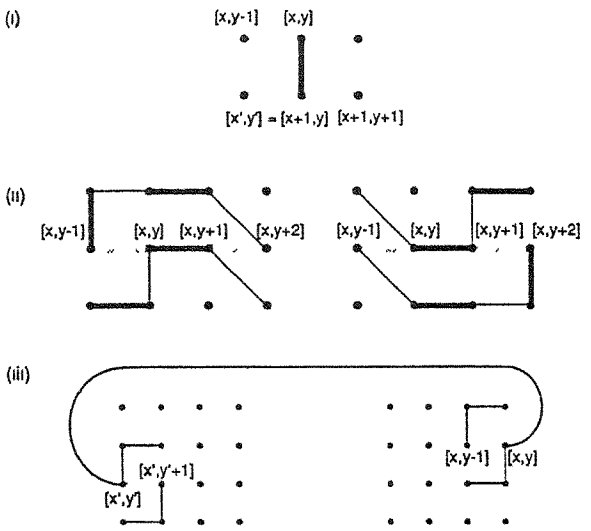


FIGURE A 3 Twin edge connecting two critical edges

Case iii: The twin edge wraps around.
Figure A.3(iii) gives the only possible situation for this case with $y' = 1$ and $y = \beta$. According to Rule 1, nodes $[x',y']$ and $[x,y]$ will be assigned 1 and $(\beta-1)$ *mod* 2 as their last bits. Since we only have to consider the case where β is odd with column splicing (Section 2.4), nodes $[x',y']$ and $[x,y]$, as the endpoints of a wraparound twin edge, have different last bits.

Case iv: The twin edge slopes down.
I1 and I2 make this case impossible. □