Quadtrees and Hypercubes: Grid Embedding Strategies Based on Spatial Data Structure Addressing

L. A. BREENE

Department of Computer Science, Texas A&M University, TX 77843, USA

A uniform toroidal addressing scheme for k-trees, or spacial data structures, is given. These include bintrees, for decomposing the line, or partitioning linear arrays; quadtrees, for two-dimensional structures; octrees, for three dimensions; etc. Reinterpretation of these addresses as hypercube node identifiers affords simple conceptualization of processor grids of arbitrary Euclidean dimension. Use of gray code produces a hierarchy of topological neighborhoods reflected in the addresses themselves and, with this, fast multicast algorithms for various multiple-processor subgroups.

Received November 2, 1992, revised February 12, 1993

1. INTRODUCTION

Quadtrees began as hierarchical spacial data structures which store two-dimensional array, or lattice data [10], although recently the term has become generic applying now to any number of dimensions. These structures are used primarily in image processing and sparse matrix algorithms [11]. They were first introduced by Hunter and Steiglitz [6], with Gargantini [3] subsequently giving a locational code, or tree address, assignment strategy which associates each node with its position in the space/array.

In this paper we present a new addressing strategy and interpret the addresses so defined so as to embed grids of various dimension in hypercubes [1, 2, 4, 9, 12]. Our addressing strategy requires a minimal number of bits to store [5] and makes use of gray code not only to characterize the geometry of the plane by preserving topological neighborhood relationships at each level of decomposition, but also to directly relate this geometry to the hypercube neighborhoods.

Reinterpretation of the addresses as hypercube node id's gives embeddings of arbitrary dimension and facilitates the development of log p (p processors) multicasts, i.e. single source broadcasts to 'subspaces' of processors. While all topologically nearest neighbors can be reached in at most log p steps, half can be reached in one. The distance and path is determined from the node address and characterizes the relative positions of the neighbors in the tree. Our approach generalizes naturally from one (the bintree) to k dimensions (the k-tree).

In the next sections, methods for assigning tree addresses and for establishing a correspondence between these and hypercube node addresses are described, example multicasts for the one and two dimensional embeddings are presented, and application sketches are given.

2. ADDRESSING

The k-tree is a hierarchical spacial data structure used to represent the discrete k-dimensional grid, or lattice, with all dimensions of size n, where $n=2^m$ and m is the height of the tree. The root node represents the grid taken as a whole. It has 2^k children corresponding to a decomposition of the space into 2^k subspaces of equal extent with all sides of length $n/2$. When $k=1$, the grid covers a line segment and the tree is a bintree. The left and right halves of the line segment correspond to the left and right children of the root and are labeled *0* and *1*. When $k=2$, the tree is a quadtree, and the nodes (from left to right) representing the subsquares are labeled *00*, *01*, *11*, *10* (0 through 3 in gray code, defined in more detail below), starting at the lower left, and proceeding in a counterclockwise direction (see Figure 1), as Karnaugh [7] labeled the circuit simplifying charts described by Veitch [13]. Gray code labeling here ensures that neighboring subsquares (and the subspaces in higher dimensions) differ in at most a single bit at the current level. When $k=3$, a octree, subcubes at front and back are labeled as in the quadtree, with the front cube addresses prefixed by *0* and back addresses by *1*. When $k>3$, successive dimensions contribute additional digits to the address on the left in the same way.

This procedure gives a k digit label to each of the subspaces resulting from a single decomposition. The decompositions continue until each node (now leaf) represents a subspace at the resolution limit which is determined by the number of processors (for compression, decomposition ends when the subspace is of one colour or value). These subsequent decompositions concatenate substrings of length k on the right to the current label, giving a label, or address, of length mk.

The labels map to lattice points as follows. We fix the origin, $(x_1, \ldots, x_i) = (0, \ldots, 0)$, at one of the lattice points whose neighborhood is of minimum cardinality, i.e. an

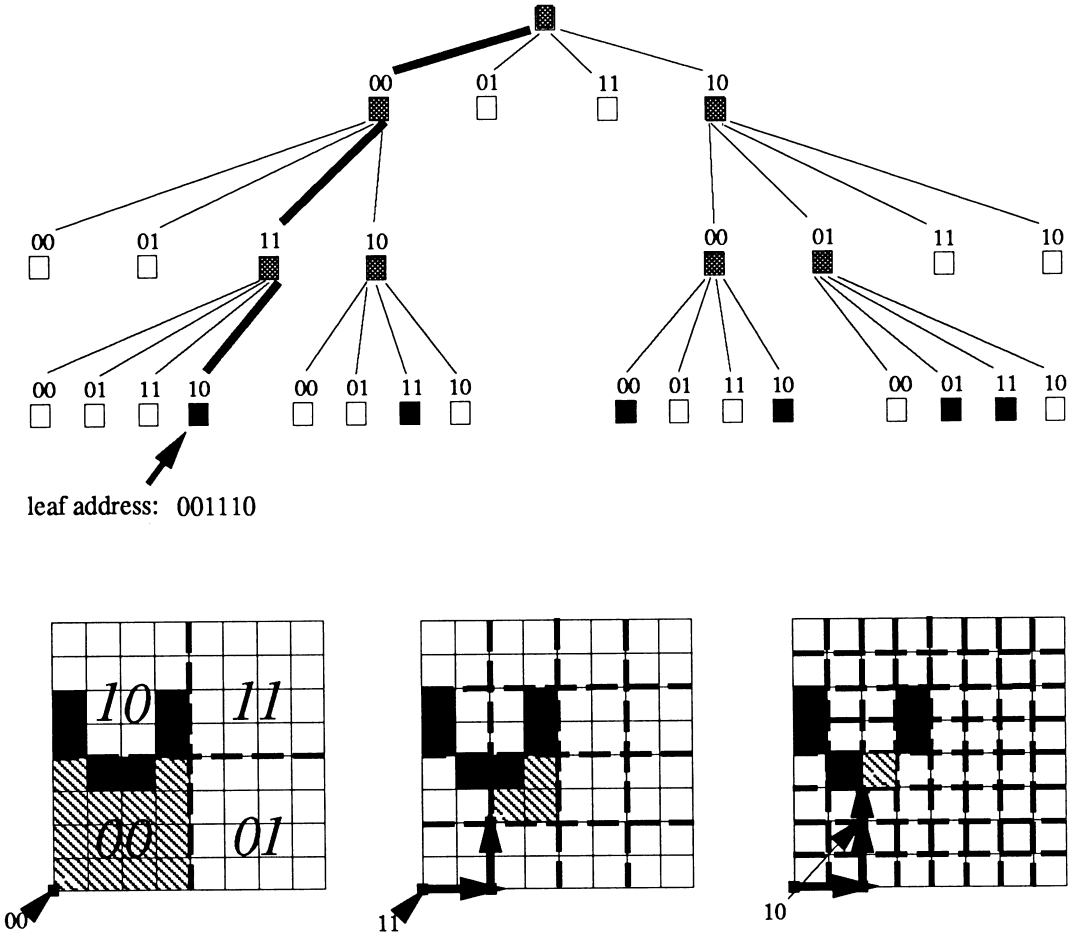

FIGURE 1. Hierarchical decomposition and addressing.

endpoint for the line segment, corner for a square or cube, etc. As the origin is to the whole space and corresponds to the root node, so the lattice point is to that cell corresponding to a particular node of the k-tree. Some lattice points are associated with more than one node. In fact, for any interior node of the tree, the set of leftmost children to the leaf all map to the same lattice point. For purposes of illustration, we take the origin to be the leftmost point of the line segment, the lower left corner of the square, the front lower left corner of the cube, etc.

The Cartesian coordinates of a lattice point defined by a k-tree address are now calculated as follows. The address may be interpreted as the definition of a path, with step of decreasing length equal to the increasing resolution of the grid defined at each decomposition. The path starts at the origin (root), and proceeds to the node representing the lattice point in $l \leqslant m/2$ steps. Summing these steps in each of the k directions gives the desired coordinates.

In two dimensions (again, refer to Figure 1) an address defines a path from the origin to the lower left hand corner of the cell represented by the node. The path step starts at $n/2$, and decreases at most to 1 (if the cell is a unit square of the grid). The coordinates of the point represented by the cell labeled $y_1x_1y_2x_2, \ldots, y_lx_l$ (which we take to be the lower left cell corner) are then

$$(x, y) = \frac{n}{2}(x_1, y_1) + \frac{n}{4}(x_2, y_2) + \cdots + \frac{n}{2^l}(x_l, y_l),$$

$$= \sum_{i=1}^{l} \frac{n}{2^i}(x_i, y_i), \tag{1}$$

where $l \leqslant m/2$. Calculation is fast because it can be performed using only bitwise **shift** and **or** operations in the integers. Formulae for k other than 2 are analogous.

This labeling scheme encodes hierarchical neighborhoods (i.e. neighbors differ in a single bit) among the cells at various levels of the decomposition. In one dimension, lattice points whose labels differ in the last bit only are nearest neighbors at the resolution limit. Two lattice points differing in a single bit i, for $i=1, \ldots, l-1$, occupy the same position in the two neighboring segments created at the ith decomposition. Likewise, in two dimensions, each of four squares resulting from a single decomposition differ from two of their nearest 'edge' neighbors (of the four: see Figure 1) in a single bit. This holds for dimensions greater than two with appropriate neighborhood expansion. By concatenating the contributions from subsequent decompositions on the right, these properties are retained at each resolution level and establish a series of hierarchical neighborhoods

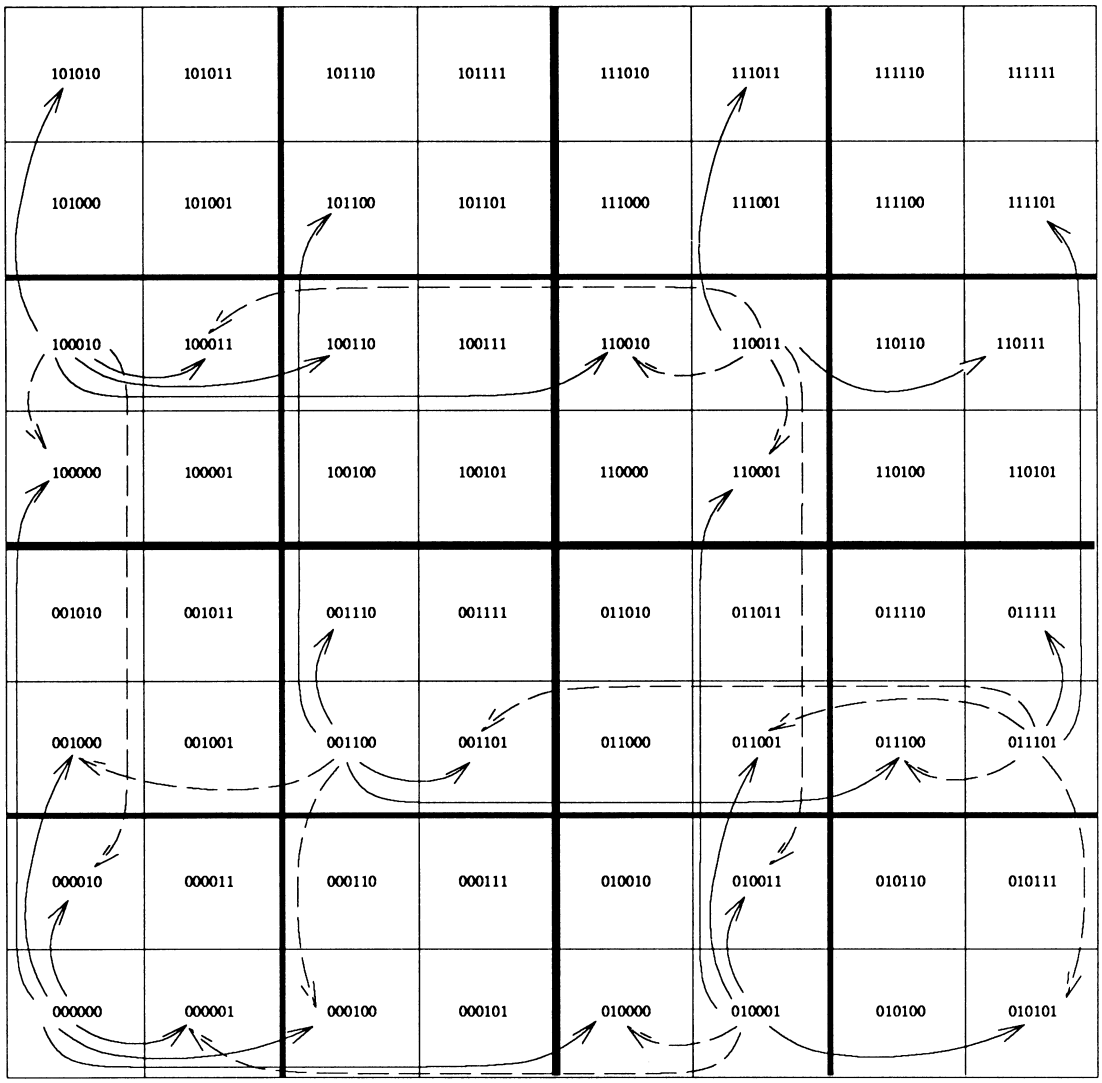

FIGURE 2. Gray code tree neighborhoods.

among the grid elements. Same size neighboring spaces remain neighbors, with finer subdivisions producing 'hierarchical nearest neighbors' each at the same position in the divided spaces. This is illustrated in Figure 2 for $n=64$. All levels of nearest neighbors are represented in the six bit tree address. For example, those at level k (taking the root as level 0) are addressed by $y_1x_1, \ldots, \bar{y}_kx_k$ and $y_1x_1, \ldots, y_k\bar{x}_k$ (with the most local represented by differences in the last two digits).

The hypercube grid embeddings are induced by simply interpreting the binary node identifiers as tree addresses of the desired dimension. Half of the Euclidean neighbors are of dilation 1. The dilation of the embedding is l, since the number of links between Euclidean nearest neighbors varies from 1 to l according to their place in the hierarchical structure, and records the distance to their nearest common ancestor in the tree structure. Reinterpretation of the binary identifiers (for dimensions $k>1$) as gray code numerals gives interesting additional properties.

Gray code is a non-positional number system made up of strings over the alphabet $\{0, 1\}$ in which any number differs from its successor/predecessor in only a single bit. Figure 3 gives the first few gray code values and their decimal equivalents.

0	000000	4	000110
1	000001	5	000111
2	000011	6	000101
3	000010	7	000100

FIGURE 3. Gray code.

Using Karnaugh maps and an inductive argument on the length of the code, one can show that conversion between binary and gray code representations is characterized by

$$g_0 = b_0,$$
$$g_i = b_{i-1} \oplus b_i, \quad (2)$$
$$b_0 = g_0,$$
$$b_i = g_0 \oplus g_1 \oplus \ldots \oplus g_i, \quad (3)$$

for $i=1, \ldots, r$, where, $g_0, \ldots, g_r$ and, $b_0, \ldots, b_r$ are gray code and binary strings, respectively, and $\oplus$ is **exclusive or**. It is well known that the gray code numbers from 2^n through $2^{n+1}-1$ may be determined by **inclusive or** 'ing 2^n *in binary* with the gray code for 0 through 2^n-1 and reversing their order [8]. From this it is clear that the code is a context sensitive language over the alphabet {0, 1}. In the sense that the succeeding 'octave' of gray codes may be found en-masse from current one, the code is a parallel number system.

Returning to the tree addresses for $k>1$, interpreted as gray code numbers, hierarchical neighborhoods are intact. However, further, a processor ordering from 0 to $2^m-1=p$ has been established and each node's successor/predecessor in the ordering is a hypercube nearest (gray code) neighbor. In hypercube connected computers this of course means that the processors are directly connected by a physical link. These neighborhood relations can be exploited in a variety of ways. The next section gives a few examples.

3. HYPERCUBE MULTICAST EXAMPLES

The embedded grid for the examples of this section is two dimensional; the tree a quadtree. We use a Boolean six-cube to complement the discussion (refer back to Figure 2). The hypercube has 64 processors with six bit node addresses. Aside from the root, the quadtree has three levels giving three hierarchical neighborhoods. Each processor has six links to neighbors, with two neighbors at each level of the hierarchy. A processor's most local neighbors may or may not be its gray code successor and predecessor. In the two dimensional grid each of a cell's most local neighbors share with it a common edge. Node ids, taken as gray code bit strings, order the processors from 0 to 63.

By an i **to** $p-1$ **multicast** we mean that node i is to pass a message to all nodes of the hypercube with id/address greater (in gray code sequence) than i. In describing the procedures that accomplish the multicast the following terms will be useful. *Forward link* designates a directed link betwen two nodes such that the gray code of the sender is less than that of the receiver. A *quadrant* is one of the $p/4$ node subsquares created by the first quadtree decomposition. With $p=2^{2l}$, the node addresses (processor ids) are of the form $y_1x_1y_2x_2, \ldots, y_lx_l$. We call y_ix_i the ith *hierarchical group*.

Pseudo-code for the first multicast follows.

```
i to p−1 multicast
  case: Node is source
    send message on all forward links.
  case: A non-duplicate message is received
    relay to neighbor which is successor
      to the current node at the same level
      on which the sender is predecessor;
    relay on all links at levels below
      that on which the signal is received;
    if node successor is not among the receivers
      and is not the sender, relay to successor.
    endif
  endcases
end i to p − 1 multicast
```

Broadcast of a message from node 0 to all other nodes is shown in Figure 4. Nodes are labeled with their gray code and decimal equivalents. Links carry the step number in which they are active. Source node 0 uses all six links, sending the message to all of its neighbors, since all are forward. Other nodes use fewer links, depending on the link through which they receive the message. The multicast is completed in six steps (assuming 1 step per local message), or in log p time.

Figure 5 shows the multicast from an 'interior' node. A broadcast from source node 9 (gray code $y_1x_1y_2x_2y_3x_3=001101$) uses the four forward links (of the possible six). Node 22(011101) receives the message from 9, so the gray codes differ in the first hierarchical group (y_1x_1). The node relays the message to each of its gray code neighbors in the second and third hierarchical groups, i.e. (23(011100), 17(011001), 21(011111), 25(010101)). It also relays the message to its successor in the first group, which is 41(111101).

Node 15 is representative of an important class of relay nodes. It receives a message from its immediate predecessor and relays it to its successor (16(011000)) in a non-local neighborhood. These operations fill in the portion of the grid that would have received messages originating from nodes below the source, were they active in the multicast. Other nodes with a similar task include 11, 35, and 59.

We now show this multicast to be $O(\log p)$. First consider broadcast from 0. Let the number of processors, $p=2^{2l}$. Node addresses are of the form $y_1x_1y_2x_2, \ldots, y_lx_l$. The proof is by induction on l. For $l=1$, the hypercube has four processors. Node zero sends to nodes 1 and 3. Node 1 relays the message to node 2. The broadcast takes two steps, as required. Assume the hypothesis true for $l=n$ or $p=2^{2n}$ processors. Node 0, in addition to its local (in quadrant) messaging, sends its message to the lower left nodes of quadrants *10* and *01*. In the second step, the lower left node of quadrant *01* relays its message to its counterpart in quadrant *11*. Once the broadcast has reached all quadrants, the inductive hypothesis applies, and the message is relayed within those quadrants in $\log p=2n$ steps. Hence the total number of steps is $2n+2$ and the theorem holds for $p=2^{2(n+1)}$ processors.

■

Broadcast from nodes other than zero remains $O(\log p)$, with additional steps required to reach short sequences of nodes at the beginning of quadrants, corresponding to the nodes in the source quadrant not involved in the broadcast.

While this multicast is fast, the number of duplicate messages is high due to the need to fill in the initial portions of the quadrants above that of the source. This

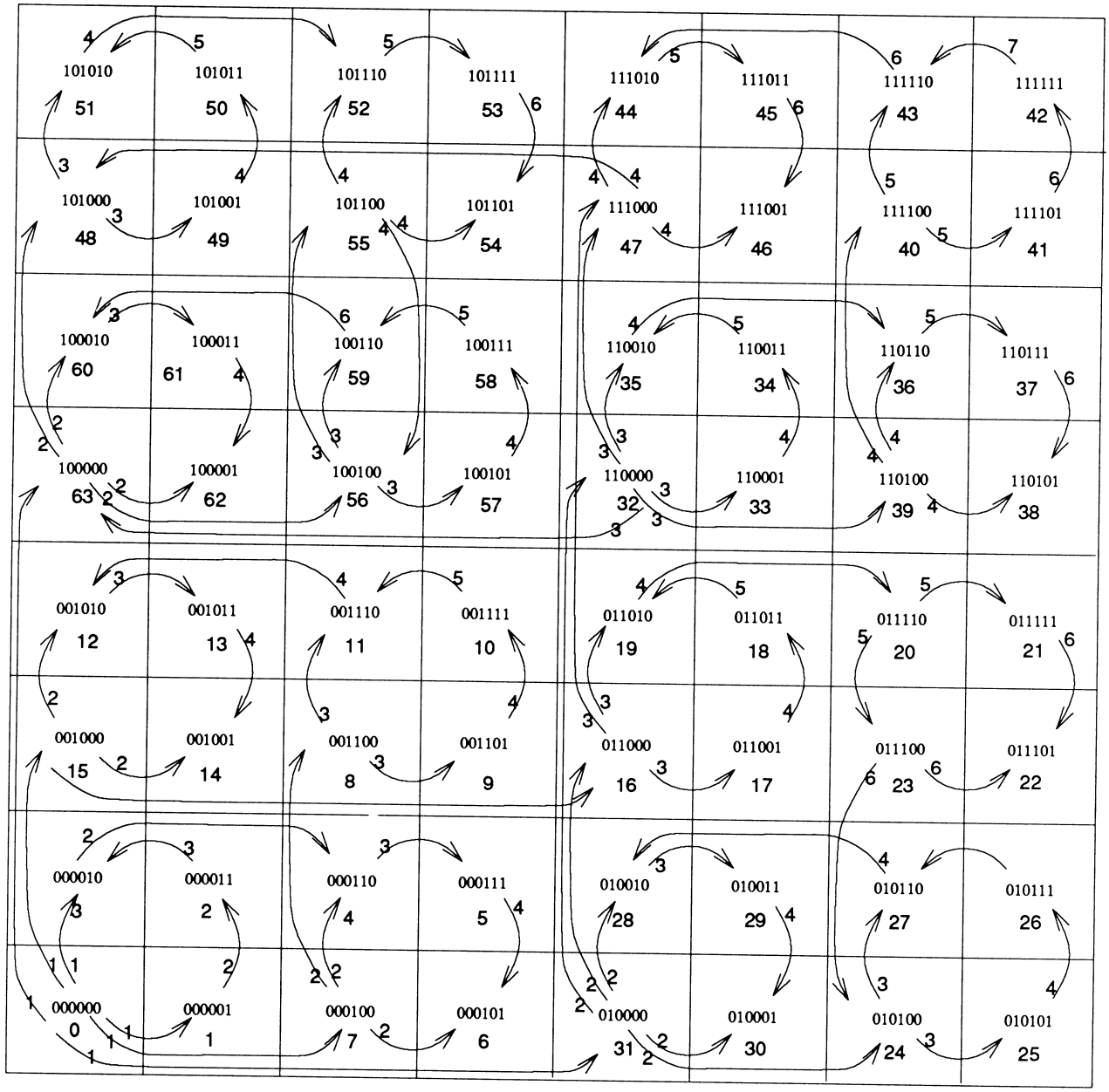

FIGURE 4. i to $p-1$ multicast with $i=0$ and $p=64$.

is accomplished by including the successor in the receiver set in each type of relay. The next algorithm remedies this situation, and introduces row/column broadcasts which are of general interest.

The basic idea of the second i to $p-1$ multicast is to distribute the message to a set of 'propagating columns' and to let the column nodes broadcast it to their rows. Special instructions operate in the quadrant in which the source resides. We describe general column and row multicasts first, and then the full i to $p-1$ multicast.

Clearly, a cube column consists of all those nodes with fixed $x_1x_2 \ldots x_l$. Similarly, a cube row consists of all nodes with fixed $y_1y_2 \ldots y_l$. To broadcast to all nodes of a column, the source node toggles each of the y_j's of its address in turn giving the addresses of l relay/receiver nodes, sending the message to each. Nodes receiving the message from group o, say, toggle the bits $y_{o+1} \ldots y_l$ in turn, forwarding the message to $l-o$ column neighbors. All nodes of the column receive the message in less than $\log p = 2l$ steps, and there will be no nodes receiving more than one copy. The row broadcast is accomplished in the same way, interchanging x_j's and y_j's.

The i to $p-1$ multicast uses the row and column broadcasts and takes exactly $\log p$ steps to complete with at most $\log p$ processors receiving exactly one duplicate message. First, the multicast source, processor i, sends the message to each of its forward neighbors. Concatenated to the message is a single bit, call it '**cs**' for column source. This bit when on (=1) indicates that the receiver is to act as a column (sub)source. The **cs** bit is set off in all messages to receivers in the source

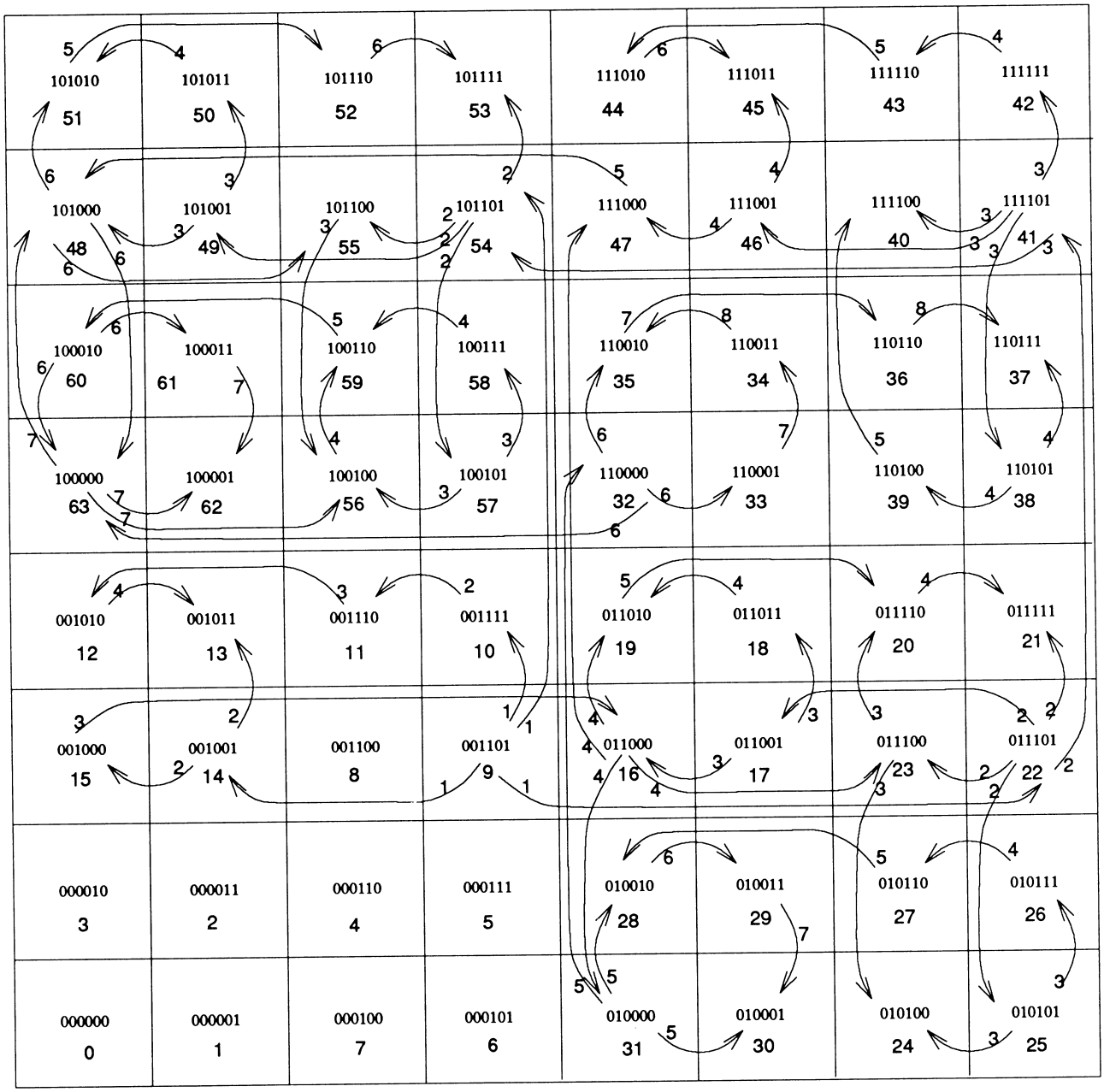

FIGURE 5. i to $p-1$ multicast with $i=9$ and $p=64$.

column. It is set on in the message to a forward row neighbor in two cases:

1. the receiver's id differs from that of the source in bit x_i, and the source has a column neighbor that is **not** forward with id differing in bit y_{i+1}.
2. the receiver's id differs from the source in bit x_0.

Nodes receiving the message from a node in the same column, or from a node in the same row with **cs** on, are column *and* row broadcasters. If the sender's address differs in hierarchical group i, the message is relayed (with **cs** off) to all column and row neighbors in hierarchical groups greater than i. Nodes receiving the message from a node in the same row with **cs** off relay the message (with **cs** off) to all row neighbors in hierarchical groups greater than i.

One final complication remains involving row broadcasters that receive messages with **cs** on, and column broadcasters. Let $s(y_ix_i)$ be the two-bit gray code successor of y_ix_i ($y_ix_i \neq 10$). If the broadcaster received the message from a node whose ith group is not $s(y_ix_i)$, and the node whose ith group is $s(y_ix_i)$ is the broadcaster's row neighbor, it relays the message to that node with **cs** on. This completes the multicast.

Notice that the **cs** bit is set on in the above algorithm in two situations. Both involve transmission to a processor in the same row as the source. The first arises from the presence of processors which would have

received their messages (in a full row/column multicast) from nodes with ids preceding that of the source. The second is needed, even when $i=0$, to get the message to the quadrant with id differing from that of the source in two bits (if there is one). The few duplicate messages originate here. Figure 6 gives an example of this multicast.

4. APPLICATIONS

The multicasts described in the last section are representative of the power of the embeddings defined by the spatial data structures. The obvious use, that of nearest neighbor message passing, its straightforward. Below we give several examples of applications requiring more complex messaging. Several have been implemented on a 64-processor nCube.

In a distributed implementation of LU-decomposition—the motivating force is the development of the embeddings—the matrix was allocated to processors in column blocks. The column allocation varied according to the work required; columns closer to the left side of the matrix called for fewer computations. Calculated results were then sent to the right to be used in calculating entries there. The two-dimensional embedding was used together with the gray code processor ordering and the second i to $p-1$ multicast was responsible for the message passing. Other methods were tried before the quadtree messaging was discovered and implemented. These methods suffered from extreme duplication of

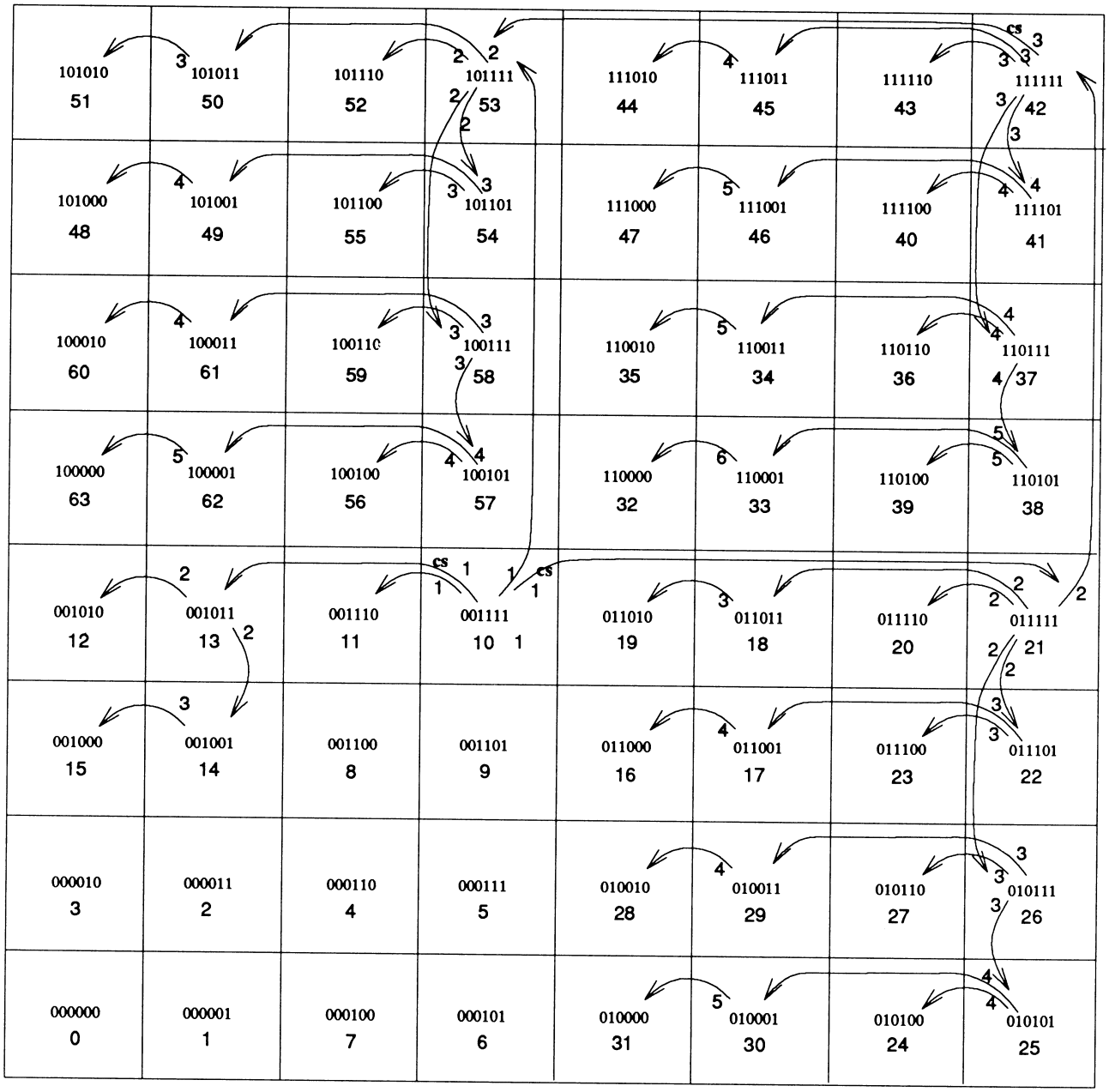

FIGURE 6. i to $p-1$ multicast with $i=10$ and $p=64$.

messages in a forward link strategy, to a crippling single neighbour 'bucket brigade' passing up the line of processors (ordered in binary). The problem required a *multicast* at *each* column block to deliver the current columns' results to *all* columns to their right. No other embedding that we are aware of gives this multicasting capability.

The tree structure of the embedding in two-dimensions was exploited in an nCube implementation based on Wise's [14] use of quadtrees for matrix inversion by Gaussian elimination with full pivoting. The matrix was decomposed in two dimensions into square submatrices and allocated one to a processor. The tree structure in two dimensions was used to determine the pivot elements at each step. Row and column broadcasts brought the appropriate elements to the processors needing them. The multicasting advantages discussed above were exploited here; however, additionally, the presence of the decomposition tree was used to quickly determine maximum elements and thus the pivots.

Image processing and graphics techniques and applications such as the Hough transform a ray tracing in two and three dimensions, respectively, are good candidates for future applications. By covering the space in straight lines formed from link and processor pairs, messages may be made to play the role of projection transformation and light.

Also of interest is the application of the addressing scheme to hardware memories. Memories in linear arrays might be decoded via the tree address to produce a space of any required dimension, with hierarchical neighborhoods.

5. CONCLUSIONS

We have given a new quadtree addressing scheme, have demonstrated some of its properties and have indicated how it can be used to achieve fast broadcast in hypercube architectures.

ACKNOWLEDGEMENTS

R. Hudson, C. Kelley, R. Schelton, B. C. Smith and J. Walker were all involved in the implementation of various multicasts and of the LU-Decomposition algorithm on the nCube. S. Shankar implemented the quadtree inversion based on Wise's paper on the same machine. This work was supported in part by the National Science Foundation via the Cornell National Supercomputer Facility (CNSF) Supercomputer Project for Undergraduate Research (SPUR) 1989–1990 and NSF grant no. CCR-9113715.

REFERENCES

[1] S. B. M. Bell and D. C. Mason, Tesseral quaternions for the octree. *The Computer Journal*, **33**(5), pp. 386–397 (1990).

[2] M. Y. Chan, Embedding of grids into optimal hypercubes. *SIAM J. Comput.*, **20**(5), pp. 834–964 (1991).

[3] I. Gargantini, An effective way to represent quadtrees. *Commun. ACM*, **27**(3), 248–249 (1984).

[4] Y. Hung and A. Rosenfeld, Parallel processing of linear quadtrees on a mesh-connected computer. *J. Parallel and Distributed Comput.*, **7**(1), pp. 1–27 (1989).

[5] A. Hunter and P. J. Willis, A note on the optimal labelling of quadtree nodes. *The Computer Journal*, **33**(5), 398–401 (1990).

[6] G. H. Hunter and K. Steiglitz, Operations on images using quad trees. *IEEE Trans. Pattern Analysis and Machine Intelligence*, **1**(2), pp. 145–153 (1979).

[7] M. Karnaugh, The map method for synthesis of combinational logic circuits. *AIEE Trans. Commun. Electron.*, **72(I)**, pp. 593–599 (1953).

[8] A. Nijenhuis and H. S. Wilf, *Combinational Algorithms for Computers and Calculators*. Academic Press, New York (1978).

[9] Y. Saad and M. H. Schultz, Topological properties of hypercubes. *IEEE Trans. Computers*, **37**(7), pp. 867–872, (1988).

[10] H. Samet, *The Design and Analysis of Spatial Data Structures*. Addison Wesley, Reading, MA (1990).

[11] H. Samet, *Applications of Spatial Data Structures*. Addison Wesley, Reading, MA (1990).

[12] R. Shankar and S. Ranka, Hypercube algorithms for operations on quadtrees. *Pattern Recognition*, **25**(7), pp. 741–747 (1992).

[13] E. W. Veitch, A chart method for simplifying truth and functions. *Proc. ACM*, pp. 127–133, Pittsburgh, PA May (1952).

[14] D. S. Wise, Representing matices as quadtrees for parallel processors. *Information Process. Lett.*, **20**, pp. 195–199 (1985).