Dense Edge-Disjoint Embedding of Binary Trees in the Mesh*

Alan Gibbons and Michael Paterson

RR208

We present an embedding of the complete binary tree with n leaves in the $\sqrt{n} \times \sqrt{n}$ mesh, for any $n = 2^{2m}$ where m is a positive integer. The embedding has the following properties: at most two tree nodes (one of which is a leaf) are mapped onto each mesh node, paths of the tree are mapped onto edge-disjoint paths in the mesh (each mesh edge considered as two anti-parallel directed edges) and the maximum distance from a leaf to the root of the tree is $\sqrt{n} + O(\log n)$ mesh steps. This embedding facilitates efficient implementation of many P-RAM algorithms on the mesh, particularly those using the balanced binary tree technique. Sush an embedding offers greater flexibility of use and improves the time complexity of these implementations by a constant factor compared with previously described embeddings.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

February 1992

Dense Edge-Disjoint Embedding of Binary Trees in the Mesh*

Alan Gibbons and *Michael Paterson*
Department of Computer Science
University of Warwick
Coventry, CV4 7AL
England

Abstract

We present an embedding of the complete binary tree with n leaves in the $\sqrt{n} \times \sqrt{n}$ mesh, for any $n = 2^{2m}$ where m is a positive integer. The embedding has the following properties: at most two tree nodes (one of which is a leaf) are mapped onto each mesh node, paths of the tree are mapped onto edge-disjoint paths in the mesh (each mesh edge considered as two anti-parallel directed edges) and the maximum distance from a leaf to the root of the tree is $\sqrt{n} + O(\log n)$ mesh steps. This embedding facilitates efficient implementation of many P-RAM algorithms on the mesh, particularly those using the balanced binary tree technique. Such an embedding offers greater flexibility of use and improves the time complexity of these implementations by a constant factor compared with previously described embeddings.

1. Introduction

Within the well-known P-RAM model of parallel computation (a shared memory model with constant time access from any processor to any memory location), the class NC defines the class of efficiently solvable problems. Such problems can be solved in polylogarithmic time using a polynomial number of processors (see [2] for example). Within current technology, for which packing constraints force the use of communication networks, it is not always possible to achieve such time complexities because a natural lower bound is $\Omega(d)$, where d is the diameter of the network over which messages have to be passed between co-operating processors.

When implementing parallel algorithms on distributed memory machines it is natural to draw upon the rich literature provided by P-RAM algorithmic research over the last decade or so. Our model of a distributed memory machine consists of a number of processors with local storage, each placed at a node of a communication network. If we associate (in one-to-one correspondence) a P-RAM processor with each network processor, then a P-RAM

*This work was partially supported by the ESPRIT II BRA Programme of the EC under contract # 3075 (ALCOM).

algorithm can be implemented on the distributed memory machine in a time equivalent to the P-RAM complexity except that each constant time required for simultaneous (SIMD) memory accesses of the P-RAM processors is replaced by the time to solve an equivalent routing problem on the network. The most useful paradigm of such routing problems is the *permutation routing problem* for which each network processor sends and receives precisely one constant-length message. There are well-known algorithms, for example [10, 5], to solve the permutation routing problem in optimal $O(d)$ time for the most commonly employed networks: hypercubes and meshes.

The time complexities for implementations of the type just described can often be improved upon. For problems in NC, the running times are usually within logarithmic factors of the lower bound $\Omega(d)$ and by the employment of certain strategies this lower bound can often be attained, see for example [4, 1, 3]. Particularly effective strategies in this regard include the technique of *compress and iterate* and the employment of *graph embedding.* A widely applicable technique advocated by Valiant [9], amongst others, to obtain optimality in a different sense is to use *parallel slackness* to hide the network *latency.* The optimality sought here is that of not exceeding the time-processor product (the *work* measure) of the P-RAM computation in the distributed memory machine implementation. If, for a particular architecture, the P-RAM can be emulated in this way, then the P-RAM is said to be *universal* for that architecture. Valiant [9] has shown that the P-RAM is universal for the hypercube, for example, but it is not universal for the mesh [7] (essentially because of its restricted bandwidth) which is the architecture of central interest in this paper.

Our concern in this paper is the improvement of running times for P-RAM algorithms implemented on the mesh-connected computer. The mesh-connected computer as a model for parallel computation is well-known [3, 5, 6, 8]. The notion of a balanced (usually binary) tree as an algorithmic structure, either implicitly or explicitly as in the *balanced binary tree technique* [2], is very important within P-RAM algorithm design. It is often precisely because such logarithmic-depth structures are used that a polylogarithmic time-complexity is attained. Data for a problem (or sub-problem) are placed at the leaves and the required result is obtained by performing computations at the internal nodes in one or more sweeps up or down the tree so that computations at the same depth are done in parallel. The universality of this technique would make an *efficient* embedding of the balanced binary tree in the mesh very widely useful in mesh implementations of P-RAM algorithms.

Such an embedding would need to satisfy the following properties if it could be described as efficient. The maximum path length from a leaf to the root needs to be $O(\sqrt{n})$ mesh steps to match the lower time bound of $\Omega(\sqrt{n})$ for genuinely distributed computation on the mesh. Clearly, the constant hidden by the notation needs to be minimised for time complexity reasons. All tree nodes at the same depth should be mapped into disjoint mesh nodes if (as in the P-RAM computation) computations are to be performed in parallel at these nodes. A minimum requirement of the embedding is that tree edges at the same depth should correspond to edge disjoint paths in the mesh if the commonest types of P-RAM algorithm employing this technique are to be simulated. However, if all tree edges are mapped into disjoint paths in the mesh then, as we indicate in Section 3, greater flexibility and some further complexity gains are possible.

In [4] an embedding was implicitly described in which edges connecting two levels within the tree are mapped into disjoint regions of the mesh. For that embedding, the maximum path length from a leaf to the root is $(5\sqrt{n/2} - 4)$ mesh steps for trees of odd height and $(7\sqrt{n/4} - 4)$ for trees of even height, and so at most $3.54\sqrt{n}$ mesh steps for all trees. In all respects the embedding also fulfils the requirements of being efficient as defined in the previous paragraph. Several examples of applying this embedding to obtain efficient P-RAM algorithmic implementations on the mesh were described in the same paper. In the following section we describe embeddings which are an improvement in two respects. All tree paths are mapped into edge-disjoint paths in the mesh, and the maximum path length from a leaf to the root is asymptotically optimal for such embeddings. These improvements are obtained without compromising the efficiency of the embedding as specified in the last paragraph.

2. The embedding

In this section we prove our main result, giving the details of our construction for the binary tree embedding. The mesh which is the target graph for the embedding has a pair of anti-parallel directed edges between each two adjacent nodes. The complete binary trees which are to be mapped into the mesh have their edges directed towards their root.

Theorem 1 *For all $m \geq 1$, there are embeddings of the complete binary trees with 2^{2m} and 2^{2m+1} leaves, in a $2^m \times 2^m$ mesh and a $2^m \times 2^{m+1}$ mesh respectively, with the following properties:*

(i) each mesh node is assigned exactly one leaf node of the tree,

(ii) each mesh node, except one, is also assigned exactly one internal node of the tree,

(iii) distinct tree edges are mapped onto edge-disjoint (possibly null) directed paths in the mesh,

(iv) the length of the image in the mesh of the tree path from a leaf to the root is at most $2^m + O(m)$ mesh steps for the 2^{2m}-leaf tree, and $\frac{3}{2}2^m + O(m)$ for the 2^{2m+1}-leaf tree.

Proof : We begin with the embedding in a square for trees with 2^{2m} leaves. The case $m = 1$ is easy. For $m = 2$, Figure 1 shows one possible embedding in the 4×4 mesh. In this figure, the internal tree nodes and the paths corresponding to the tree edges are drawn with increasing boldness from leaves to root. The leaf nodes are not shown explicitly since there is one at each mesh node. Note that some tree edges, incident with the leaves, are mapped to null paths, indicated by loops in the figure. The root is embedded on the left side, but the heavy path shown from this to the top-left corner is used later in larger embeddings. The node distinguished with a dotted square in the figure is that unique node which has not yet been assigned an internal tree node. The small diagram to the right gives the salient features of this embedding, A_2, for use in the recursive construction. The

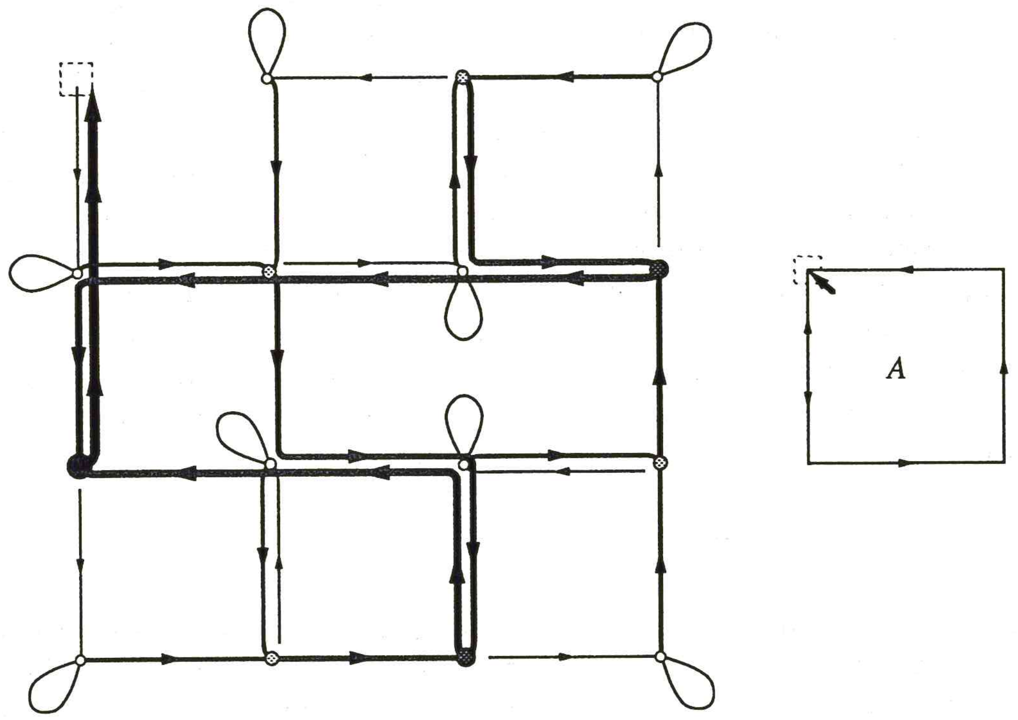

Figure 1. Layout A_2

arrows on the perimeter indicate the usage so far of the outside edges, and show that all the clockwise outside edges on three of the sides are as yet unused.

Our construction requires also an alternative 4×4 embedding, B_2, shown in Figure 2. The root here is embedded in the interior of the square but there is an outgoing path from it to the lower-right corner. This time, all the clockwise edges on the top, left and bottom sides are free.

The next stage in our construction, the embeddings for $m \geq 3$, is shown in Figure 3. Three A_{m-1}'s and one B_{m-1} are combined to give embeddings of the 2^{2m}-leaf tree in a $2^m \times 2^m$ mesh. The three new internal tree nodes required are shown by white and black circles, and are connected by paths of appropriate weight. For the recursion, the embedding is continued in two different ways. The black root node can be connected to the top-left corner by one of the hatched paths shown, or joined to the lower-right corner by another hatched path. The first alternative yields an embedding A_m which has the edge characteristics of type A given by the small diagram in Fig. 1, while the second similarly yields B_m. The arrangement shown in Figure 3 therefore represents a recursive step by which the construction can be continued indefinitely. The third path illustrated, with different hatching, to the lower-left corner, will be used in the 2^{2m+1}-leaf embedding.

We will show that the maximum distance, $D(m)$, from a leaf to the root of the tree is $2^m + O(m)$ mesh steps. It is a trivial matter to construct an embedding for $m = 1$ for which this distance is 2. For $m = 2$, we see by inspection of the embedding B_2 of Fig. 2 that this maximum distance is 6 mesh steps and so we have $D(2) = 6$. For $m > 2$ the proof proceeds by induction on m. Consider square meshes with $n = 2^{2m}$ nodes, $m \geq 2$. Let $A(m)$, $B(m)$, and $C(m)$, be the maximum distances from a leaf to the output from

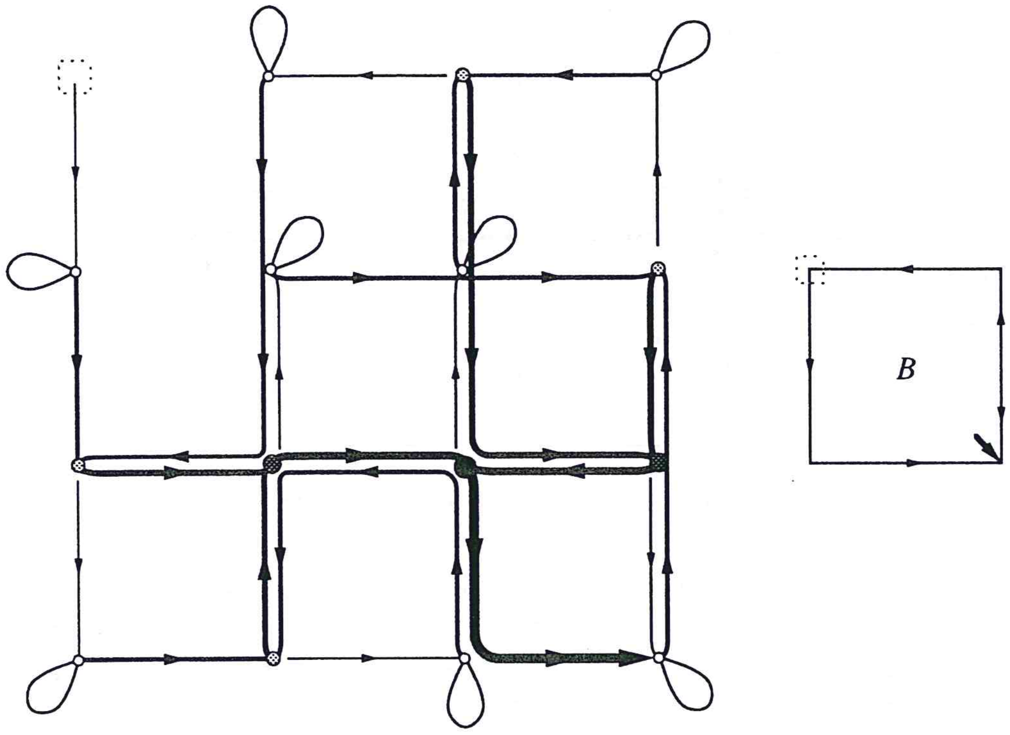

Figure 2. Layout B_2

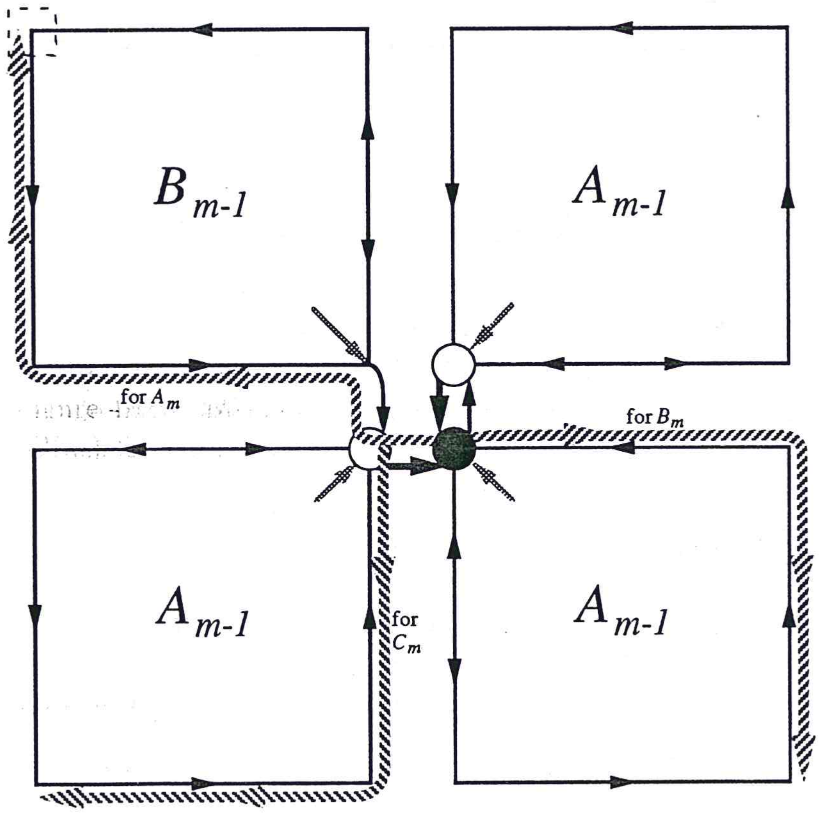

Figure 3. The recursive construction using A_{m-1} and B_{m-1}

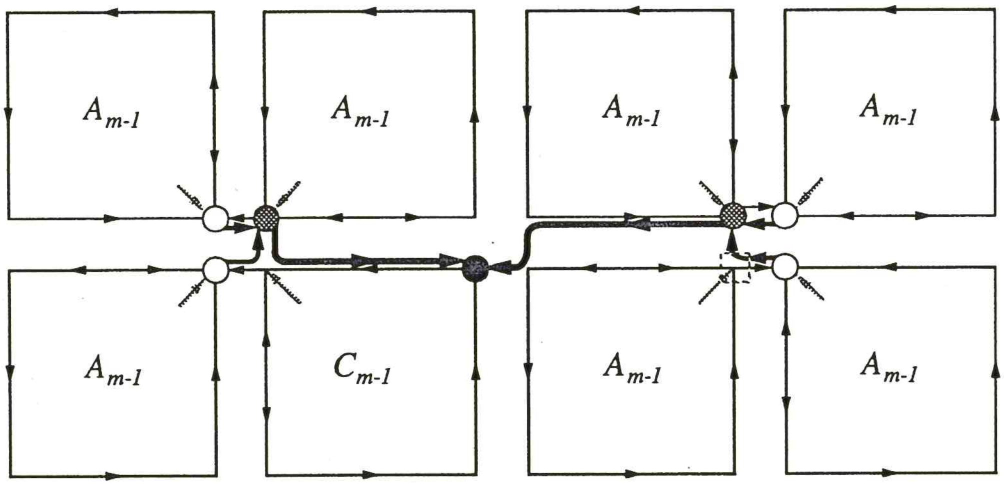

Figure 4. The embedding in a $2^m \times 2^{m+1}$ mesh

the top-left corner of pattern A_m, the lower-right corner of pattern B_m and the lower-left corner of pattern C respectively, for the $2^m \times 2^m$ array. From Figures 1 and 2, we see that $A(2) = 10$ and $B(2) = 8$. A corresponding layout C_2 with $C(2) = 9$ is easy to derive from B_2. We can verify from Figure 3 the following recurrence equations for $m \geq 3$:

$$\begin{aligned}
A(m) &= D(m) + 2^m \\
B(m) &= D(m) + 2^m - 2 \\
C(m) &= D(m) + 2^m - 1 \\
D(m) &= \max\{A(m-1)+2, B(m-1)+2\} = D(m-1) + 2^{m-1} + 2.
\end{aligned}$$

The solution to these equations is:

$$D(m) = 2^m + 2m - 2, \text{ for } m \geq 1.$$

For the case $n = 2^{2m+1}$, if $m \geq 3$ we can connect seven copies of A_{m-1} with one copy of C_{m-1} as shown in Figure 4. Let $D'(m)$ be the corresponding maximal leaf-to-root distance. We may verify in Figure 4 that $D'(m) = \max\{A(m-1)+4, C(m-1)+3\} + 2^{m-1}$ and so, $D'(m) = 3.2^{m-1} + O(m)$. The cases where $m < 3$ are simple. □

3. Lower bounds

For many applications, an important efficiency measure for a tree-embedding is the maximum length in the embedding of the path from a leaf to the root. For any given class of embeddings, we will denote this maximum by $d(n)$ for trees with n leaves.

Let B_r be the set of vertices of the mesh graph $\mathbb{Z} \times \mathbb{Z}$ within path length r of the origin. Then $|B_0| = 1$, $|B_1| = 5$, and in general $|B_r| = 1 + \sum_{i=0}^{r} 4i = 2r(r+1)+1$ for $r > 0$. For any injective mapping of n leaves into the mesh, if $n > |B_r|$ then some vertex has to be mapped to a mesh node at distance greater than r from the root.

For embeddings onto a specified region R, $d(n)$ is bounded below by the *radius* of R, i.e., the minimum distance r such that for some 'central' node c, every node has distance at most r from c.

These observations are enough to prove the following result.

Theorem 2 *Consider leaf-disjoint embeddings in the mesh of treese with n leaves.*

(i) For an arbitrary embedding,

$$d(n) \geq \sqrt{n/2} - O(1),$$

(ii) for an embedding in an $r \times s$ rectangle where $n = rs$,

$$d(n) \geq \left\lceil \frac{r-1}{2} \right\rceil + \left\lceil \frac{s-1}{2} \right\rceil = (r+s)/2 - O(1).$$

Corollary 1 *The values of $d(n)$ attained by our constructions are asymptotically optimal for embeddings into squares and rectangles of the prescribed type.*

A natural question to consider is whether the pairs of anti-parallel edges are necessary. Can complete (undirected) trees be densely embedded in the usual undirected mesh? Each mesh node (except one) is host to one leaf vertex, with degree one, and one internal vertex, with degree three, and so has a total of at least four embedded edges incident with it. Note that some of the edges adjacent to leaves can be mapped into paths of length zero, the loops in our figures, and so some mesh nodes may require only *two* of their incident mesh edges. Thus there is no immediate contradiction from degree considerations. However, we now consider local details and easily find a contradiction.

Consider boundary mesh nodes, away from the one special node that does not host an internal tree vertex. Any such node has degree less than four and so must have a loop in the embedding. It therefore is host to a leaf vertex and the internal node adjacent to that leaf, and requires one incoming path from another leaf, and one outgoing path to the parent vertex. Since the neighbouring boundary nodes are in the same predicament, there is an impossible situation at the boundary, even worse if it is at a corner.

4. Algorithmic issues

The maximum distance from a leaf to the root in the embedding is a measure of the routing time required for a single vertical sweep of the balanced binary tree when it is employed in mesh implementations of P-RAM algorithms which use this algorithmic structure. For an efficient embedding of the tree, the $O(\log n)$ P-RAM algorithmic time translates to $O(\sqrt{n})$ time on the square mesh. This is optimal to within a constant factor. The constant of 3.54 in [4] is improved to 1 by the embedding described in the previous section. Thus we demonstrate a speed-up by the constant factor of approximately 3.54 in routing time on the mesh. This would normally be the dominating term in the overall time complexity for a single vertical sweep of the tree, certainly for implementations of P-RAM algorithms in the class NC, where at worst polylogarithmic time computations would be performed at internal nodes of the tree.

For some algorithms, the edge-disjointness property of the embedding described in this paper yields further complexity gains. Occasionally it is useful for all nodes in the tree, not just those at the same level, to pass messages simultaneously to their sons in such a way that this continues until all messages (including that from the root) reach the leaves of the tree. An example of such a *cascading* requirement is provided within the bracket matching algorithm detailed in [4]. This can be simulated in the embedding of [4] by first allowing the messages from the internal tree nodes adjacent to leaves to be passed (by copying to descendant nodes) to the leaves, then subsequently messages from the nodes at the next level are sent to the leaves and so on, until finally the message from the root is allowed to be copied down to all descendants. In this way, only tree edges at the same level are being employed at the same time and the lack of disjointness of all paths from the root to the leaves is no hindrance. In the embedding of [4], the routing time for such a process would be $3.54\sqrt{n}(1 + 1/2 + 1/4 + 1/8 + ...) \sim 7.08\sqrt{n}$. The successive terms arise from routing from successive tree levels starting at the root. For the embedding of this paper, the path-disjointness property allows messages to be passed down the tree simultaneously from all levels, and so the routing time for the *cascading* requirement is just that for passing a message from the root to the leaves (this masks the time for message passing from all other internal nodes) which is $\sqrt{n} + O(\log n)$, a speed-up by a constant factor of 7.08.

5. Summary and open problems

We have described how the complete binary tree may be embedded in the mesh such that an algorithmically important parameter, the maximum distance from a leaf to the root, is (asymptotically) optimised. Optimality was obtained for 2-dimensional rectangular meshes having a minimum length perimeter consistent with the requirement that all the mesh nodes are host to a leaf of the tree and all (except one) are also host to an internal node of the tree. Thus trees with $n = 2^m$ leaves are mapped into a square mesh (if m is even) or a rectangular mesh with length twice its width (if m is odd).

Because of the logarithmic lower order term in our maximum leaf-to-root distance there is a small gap between the distance obtained here and the naive lower bound. Whether this gap can be closed, from either side, is an open question.

One of the most useful methods of addressing mesh nodes for algorithmic purposes is to employ the *shuffled-row major ordering* (see, for example. [2, 1, 10, 5]). In this scheme, the 2^m consecutively numbered processors (starting at suitable processors) cover sub-areas of the mesh which are squares for even m, and rectangles with length twice the height for odd m. Our embeddings are therefore natural in this sense also. For other processor addressing schemes or machines with a fixed square mesh of few processors, it may prove useful to embed into a smallest square of processors for all m, but with not all mesh sites necessarily playing host to tree nodes. An optimum solution to minimising $d(n)$ would give

$d(n) \equiv \sqrt{n}$. Of course, the embedding described in this paper gives such a solution for even m. However, to obtain such an embedding for odd m is an open problem.

In Section 3 we showed that boundary effects prevented an embedding in the undirected mesh. A mesh architecture sometimes used is in the form of a torus, with no boundary. It seems unlikely that a complete binary tree with 2^{2m} leaves could be embedded in the undirected $2^m \times 2^m$ torus, but we have not been able to prove this.

The question of how to find similarly dense embeddings of complete binary trees in meshes of higher dimension remains unanswered. Similarly, the problem of finding dense embeddings of complete trees with fixed higher degree in meshes of any dimension is unsolved. From a graph-theoretic point of view, dense embeddings of arbitrary trees in meshes present a challenge, although these problems may prove to be of less general algorithmic importance.

References

[1] A.M. Gibbons, *A tutorial introduction to distributed memory models of parallel computation, Research report 185, Department of Computer Science, University of Warwick, May 1991.*

[2] A.M. Gibbons and W. Rytter, *Efficient Parallel Algorithms, Cambridge University press, Cambridge (1988).*

[3] A.M. Gibbons and Y.N. Srikant, *A class of problems efficiently solvable on the mesh-connected computer including dynamic expression evaluation,* Information Processing *Letters 32 (1989) 305-311.*

[4] A.M. Gibbons and R. Ziani, *The balanced binary tree technique on mesh-connected computers, Information Processing Letters 37 (1991) pp. 101–109.*

[5] D. Nassimi and S. Sahni, *An optimal routing algorithm for mesh-connected parallel computers, Journal of the ACM, Vol.27, No.1, January 1980, 6-29.*

[6] D. Nassimi and S. Sahni, *Finding connected components and connected ones on a mesh-connected computer, SIAM J. Computing (1980) 744-757.*

[7] D. B. Skillicorn, *Architecture-independent parallel computation,* IEEE Computer, De*cember, 23:38-50, 1990.*

[8] C. Thompson and H.T. Kung, *Sorting on a mesh-connected parallel computer,* Com*munications of the ACM, 1987, 263-271.*

[9] L.G. Valiant, *General purpose parallel architectures,* Chapter 18, Handbook of Theo*retical Computer Science, Volume A, Jan van Leeuwen (ed), Elsevier Science Publishers (1990).*

[10] L.G. Valiant and G.J. Brebner, *Universal schemes for parallel computation,* Proceed*ings 13th ACM Symposium on Theory of Computing, 1981, 263-277.*