Proc. 27th Symposium on Foundations of Computer Science, IEEE, 1986, pp. 264-273.

MESHES WITH MULTIPLE BUSES

Quentin F. Stout

Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109 USA

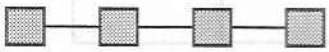
ABSTRACT

This paper considers mesh computers with buses, where each bus provides a broadcasting capability to the processors connected to it. We first disprove a published claim by showing that on a 2-dimensional mesh with a bus for each row, where each row must solve its own problem with data that is independent of all other rows, there are problems where the rows can cooperatively solve all subproblems faster than any single row can solve its own problem. As a corollary we obtain efficient solutions to some graph problems. We also consider the optimal layout of buses for a given dimension and number of buses per processor, where optimality is defined in terms of the time needed to simulate any other machine with the same constraints. Using new families of layouts, optimal or nearly optimal families of layouts are determined for each possible choice of dimension and number of buses per processor.

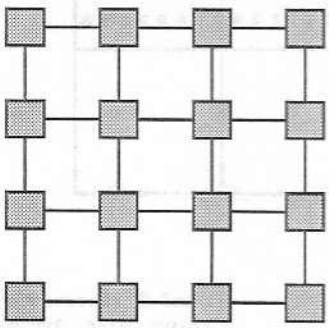
1. INTRODUCTION

Mesh-connected computers have long been of interest because many problems have data which maps naturally onto them, and because the simple interconnections of such computers enables them to be scaled to a large number of processors. A *d-dimensional mesh computer of size n*, where n, d, and $n^{1/d}$ are integers, contains n *processing elements (PEs)* arranged in a d-dimensional grid with $n^{1/d}$ PEs on each edge, with each PE connected to its $2d$ nearest neighbors. (PEs along the edges have fewer connections.) See Figure 1. The PEs are all identical and have a wordsize which is at least logarithmic in n (so that a PE can store its own coordinates), they have a standard instruction set with unit time operations, they can exchange a word of information with all of their neighbors in unit time, and they have unlimited memory. (The last point is used only to simplify the discussion. It would suffice if they had a constant amount of memory, where the constant depended upon the problem.) *Mesh* will mean mesh computer.

The primary deficiency of meshes is that they have large diameters, in that information must traverse $(d\text{-}1)n^{1/d}$ links to travel from one corner to the opposite corner of a d-dimensional mesh of size n. Several additions have been proposed which can reduce this worst-case PE-to-PE communication time. These include attaching trees, pyramids, orthogonal trees, or buses to the processors. This paper analyzes the last suggestion, with an emphasis on multiple buses. For our purposes, a *bus* is a device whereby any PE attached to it can send a word of information to all other attached PEs in unit time. There is no collision resolution, so all algorithms must be written so that no two PEs try to use the same bus at the same time. (This is similar to the EREW PRAM model.) Buses have multiple PEs



a) A 1-dimensional mesh of size 4



b) A 2-dimensional mesh of size 16

Processing Element (PE)

Communication Link

Figure 1.

This research was funded by NSF grant DCR-8507851.

attached to them, and a PE can be attached to more than one bus. A bus, together with all its attached PEs, will be called a *bus system*. This paper considers the use and layout of bus systems.

Three types of layouts of bus systems for meshes have been considered previously: a single global bus [Bok, Jor, Sto1, Sto2], buses on each row and column [PK-R1, PK-R2], and multiple global buses [Agg]. NASA's Finite Element Machine (FEM) is a mesh with a global bus [Jor], and the global or feature of the Massively Parallel Processor (MPP) can also be used as a global bus [Bat].

Algorithms for meshes with buses have concentrated on determining geometric properties of black/white images [PK-R2, Sto1], finding minimums and maximums [Bok, PR-K1, Sto2, Agg], selecting the k^{th} largest value [Sto2, PR-K2], and numerical matrix calculations. Optimal algorithms have been found for a few such problems, but in general proofs of optimality are complicated, especially when there are multiple buses. In particular, in section 2 it is shown that multiple non-overlapping bus systems, each trying to solve a simple problem involving only its data (where the data in each system is independent of all other systems), can solve all their problems more quickly by cooperation than by having each bus system solve its own problem independently. At first glance this result seems impossible, and it contradicts an intuitively appealing argument appearing in [PK-R2]. Here our primary contribution is not the rather straightforward algorithm which proves the theorem, but the recognition that such a cooperative algorithm is possible. To demonstrate that this result can be applied to more general problems, it is also shown that this approach can be used to give efficient solutions for some graph problems which explicitly involve combining information from different rows.

In section 3 we address the problem of finding "optimal" layouts of bus systems. We consider the dimension, size, and maximum number of buses per PE as given constraints, and try to find a layout which is "best" in its ability to rapidly simulate any other layout with the same constraints. This is similar to the approach taken by Leiserson with his Fat-Trees [Leis]. By varying the size one can obtain families of layouts, and we exhibit optimal or nearly optimal families for each choice of dimension and number of buses per PE. These results are similar to, and have important differences from, various results for VLSI layout.

We use the symbols O, Ω, and Θ to mean "order no more than", "order no less than", and "order exactly", respectively. All times are worst case unless stated otherwise. The term "poly-log" means $O((\log n)^k)$ for some k. Because of space limitations, many of the "proofs" are merely sketches of the important ideas.

2. COOPERATIVE SPEEDUP

In [Bok, Sto1, Sto2] it was shown that a 1-dimensional mesh of m PEs with a single global bus, where each PE starts with a value in some set S, if * is a semigroup operation on S which can be computed in unit time, then the result of applying * to all the PEs' values can be determined in $\Theta(m^{1/2})$ time, and further, this time is the best possible. In addition, the proof of optimality showed that almost any nontrivial problem must take $\Omega(m^{1/2})$ time. (An example of a trivial problem is "Send the value in the leftmost PE to all other PEs".)

Suppose we try to apply this result to a 2-dimensional mesh of size n, where each row is a bus system and there are no other buses. Suppose, for example, that each PE has either a 0 or 1, and in each row we wish to compute the location of the leftmost 1. (If there is no 1 in the row, then the answer is $n^{1/2}+1$.) Since the location of 1s in different rows is completely independent, and a separate answer is needed for each row, it seems that the best that can be done is to have each bus system compute its own answer. Each row has $n^{1/2}$ PEs, so this will take $\Omega(n^{1/4})$ time. This argument was put forth in [PK-R2], but it is incorrect.

Theorem 2.1: Given a 2-dimensional mesh of size n, where each row is a bus system and each PE has either a 0 or a 1, the problem of determining the location of the leftmost 1 in each row (and having each PE know the answer for its row) can be solved in $\Theta(n^{1/6})$ time. Further, this time is optimal if there are no additional bus systems.

Proof: Our algorithm follows a common pattern of algorithms for meshes with buses, in that it includes a portion which is purely a mesh algorithm and a portion which is purely broadcasting, with the times of these portions balanced. Partition the mesh into $\lceil n^{1/6} \rceil \times \lceil n^{1/6} \rceil$ subsquares, called *blocks*. The set of blocks are partitioned into rectangles 1 block high and $\lceil n^{1/6} \rceil$ blocks wide, called *groups*. A group contains $\lceil n^{1/6} \rceil$ rows of PEs, and approximately $n^{1/3}$ columns. Notice that no more than $n^{1/6}$ groups share the same *band* of $\lceil n^{1/6} \rceil$ rows. Within each band the leftmost group will use the topmost row bus system for its internal use, the second from the left group will use the second row bus system, and so on. See Figure 2.

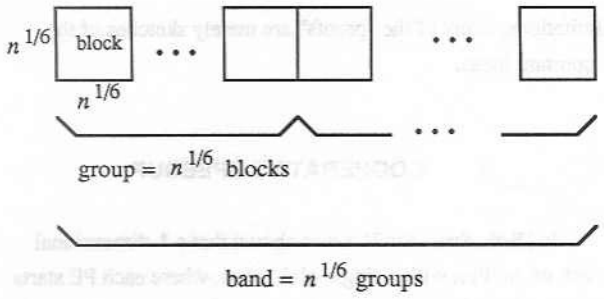


Figure 2

The algorithm starts by using a mesh algorithm within each block to locate the leftmost 1 in each row, if any, and move this information to the leftmost PE along the block's group's bus. Then, within each group, the leftmost block starts to use the bus, followed by the block to its right, and so on until finished. When a block's turn comes, it broadcasts the location of the leftmost 1 in each row for which it has a 1 but no previous block does. When it is finished it broadcasts a signal that it is done and the next block can proceed. Notice that the left-to-right ordering of the blocks' use of the bus gaurantees that when the location of a 1 is broadcast, it is indeed the leftmost 1 for its row, at least among the columns in the group.

When all groups are done, they have determined the leftmost position of a 1 in each of their rows, if there is any such 1. The last block in each group has been recording this information and storing each row's answer in the last PE in that row. Now the blocks, in left-to-right turns, broadcast their answer, or a signal that they have no 1, in all rows simultaneously. The PEs in each row just record the first broadcasted location of a 1 in their row, or, if no location is broadcasted, they use the value $n^{1/2}+1$.

The time for the mesh algorithm is $\Theta(n^{1/6})$, and since there are no more than $\lceil n^{1/6} \rceil$ groups on each row, the time for the final round of broadcasting is also $\Theta(n^{1/6})$. To see that the broadcasting within each group takes $\Theta(n^{1/6})$ time, notice that there are exactly two types of messages being broadcast. One is the signal for the next block to proceed, and the other is the location of a leftmost 1 in some row (leftmost among those in the group). There are only $\lceil n^{1/6} \rceil$ blocks in a group, so there are only $\lceil n^{1/6} \rceil$ signals for the next block to proceed. Further, for each row in a group there is at most 1 broadcast giving the leftmost location of a 1 in that row, so no more than $\lceil n^{1/6} \rceil$ locations are broadcasted.

As [PK-R1] noted, optimality can be shown by mimicing the proof of optimality for 1-dimensional meshes. Consider the simpler problem of deciding if all of the values in the first row are 0 or not. Using only mesh connections, at time t any PE can know the initial values in at most $2t+1$ of the PEs on the first row. Therefore at time t any PE can broadcast information involving the initial values of at most $2t+1$ PEs on the first row which have not been involved in any previous broadcast. Further, if the answer is always known by time T and broadcasts involving row r are used, then they must be completed by time $T-r+1$, and involve information leaving row 1 by time $T-2r+2$. Since there are $n^{1/2}$ PEs in the first row, to finish by time T one must have

$$\sum_{r=1}^{T/2} \sum_{t=1}^{T-2r+2} 2t+1 \geq n^{1/2}$$

Therefore $T = \Omega(n^{1/6})$. ∎

Corollary 2.2: Given a 2-dimensional mesh of size n , where each row is a bus system:

a) If each PE has either a true or a false , then the problem of determining the or (or and) of all values in each row (and having each PE know the answer for its row) can be solved in $\Theta(n^{1/6})$ time. Further, this time is optimal if there are no additional bus systems.

b) If each PE has an integer in the range $\{0..r\}$ for some integer r , then the problem of determining the largest (or smallest) value in each row can be solved in $\Theta(n^{1/6}\log^{2/3}r)$ time.

c) If each PE has some value, then the problem of determine the largest (or smallest) value in each row can be solved in $\Theta(n^{1/6}\log^{1/3}n)$ expected time, if all orderings of the values in each row are equally likely.

Sketch of proof: a) The or is true if and only if there is a leftmost true, and the and is false if and only if there is a leftmost false.

b) In each row use a binary search. Using block, group, and band sizes as in the previous theorem would give a time of $\Theta(n^{1/6}\log r)$. To reduce this slightly, use square blocks of edgelength $\Theta(n^{1/6}\log^{2/3}r)$ and put $\Theta(n^{1/6}/\log^{1/3}r)$ blocks per group. This gives $\Theta(n^{1/6}/\log^{1/3}r)$ groups per band, where a band has $\Theta(n^{1/6}\log^{2/3}r)$ rows, so each group can use $\Theta(\log n)$ buses for its internal use. After the initial mesh algorithm to determine the largest entry in each row of each block, each iteration of the binary search will take $\Theta(n^{1/6}/\log^{1/3}r)$ time.

c) Use blocks of edgelength $\Theta(n^{1/6}\log^{1/3}n)$ and $\Theta(n^{1/6}\log^{1/3}n)$ blocks per group, giving $\Theta(n^{1/6}/\log^{2/3}n)$ groups per band and $\Theta(\log n)$ buses per group. Have each block broadcast its largest value in each row where its value is larger than any value previously broadcasted for that row. ∎

Corollary 2.3: Given the adjacency matrix of an undirected graph G, stored one entry per PE in a 2-dimensional mesh of size n where each row and each column is a bus system, the components of G can be labeled and a spanning forest marked in $\Theta(n^{1/6}\log^{2/3}n)$ time.

Sketch of proof: Use blocks of edgelength $\Theta(n^{1/6}\log^{2/3}n)$ and $\Theta(n^{1/6}/\log^{1/3}n)$ blocks per group, giving $\Theta(n^{1/6}/\log^{1/3}n)$ groups per band and $\Theta(\log n)$ buses per group. Use the parallel component labeling algorithm in [HCS], which has a logarithmic number of stages in which each row (representing a vertex) must locate an edge (if any) to a vertex of a different label. (Finding just a different label, instead of the more usual minimal label, slightly reduces the time of this algorithm. However, for spanning trees it will result in just finding an arbitrary spanning tree, instead of one of minimal weight.) An initial mesh algorithm reduces all the connectivity information in each block down to $\Theta(n^{1/6}\log^{2/3}n)$ edges, and each subsequent stage needs only $\Theta(n^{1/6}/\log^{1/3}n)$ time. ∎

Once a spanning forest has been found, there are several graph properties which can be quickly determined. Other problems which can be solved in a similar amount of time include marking all the articulation vertices of G, marking all bridge edges of G, computing the cyclic index of G, finding the biconnected components of G, and deciding if G is bipartite. See [A-H, A-K, Sto3, T-V] for relevant algorithms which can be adapted for used here. It should be noted that there is a slight gap between the times of the above results, and the general lower bound of $\Omega(n^{1/6})$ applicable to such meshes. It appears to be extremely difficult to prove a larger lower bound.

2-dimensional meshes with row and column bus systems were apparently first studied in [PK-R1], where $\Theta(n^{1/6})$ algorithms were given for a few problems which explicitly involved combining information from all rows. Theorem 2.1 shows that, even though the bus systems have independent data, and hence one row learning of another row's values is of no use in solving its problems, nonetheless the structure of the communication requirements of certain types of problems allow rows to help each other. However, the above technique cannot be used on closely related problems such as determining the number of 1s in each row, or the parity of the number of 1s in each row.

Open Question: For what problems $\mathcal{P}$ is it true that if each row of a 2-dimensional mesh with a bus per row must solve its own instance of $\mathcal{P}$, then the rows can cooperatively solve all problems significantly faster than single rows can solve their own problems independently? (Significantly faster means faster by more than a constant multiple.)

Even if one knew the answer to the above question, it is not clear that one would know how to use such row operations to help solve problems that explicitly involve combining information from all rows. The corollaries above, and the algorithms in [PK-R1, PK-R2], give some examples where row and column bus systems can improve a 2-dimensional mesh for specific problems. On the other hand, simple counting arguments show that such bus systems cannot help problems such as sorting.

Open Question: For what problems $\mathcal{P}$ is it true that a 2-dimensional mesh with row and column bus systems can use the buses to solve $\mathcal{P}$ significantly faster than when not using them?

3. LAYOUT OF BUS SYSTEMS

A global bus is promoted as being useful because it enables any PE to communicate with all others in unit time, but it seems that this capability is of less importance than several other factors. As was noted above, in [Sto2, Bok] it was shown that a 1-dimensional mesh of size n with a global bus system must take $\Omega(n^{1/2})$ time to solve a simple problem such as having each PE start with a value and then applying a semigroup operation to these values. However, if smaller, "local" bus systems are allowed, where each PE is still in at most one bus system, then this problem can be solved much faster.

For example, partition the PEs into consecutive triples, where each triple will represent a node of a balanced binary tree. One PE in each triple is used for communicating with the parent, one with the right child, and the remaining one with the left child. If a node p has a right child q, then a bus system consisting of the parent PE of q and the right child PE of p is formed, and connections to left children are formed similarly. To apply a semigroup operation to the values, the 3 PEs in a leaf combine their values and use the bus to the parent to send it. Any intermediate node combines values received with its initial values and then sends the total up on its parent bus. The entire algorithm takes $\Theta(\log n)$ time, and uses only one bus per PE.

It is clear from this example that a mesh of size n can use nonoverlapping bus systems to simulate any parallel computer with an interconnection scheme which is a connected graph with no more than $n/2$ edges, where the simulation time depends

upon the maximum degree of the graph. While this gives the potential of very fast algorithms, arbitrary buses also destroy the primary virtue of meshes, namely the fact that they can be laid out without long wires (at least for lower dimensional meshes). To preserve this important feature, **from now on all bus systems will be contiguous groups of PEs.** All previously studied bus systems satisfied this constraint.

Our goal is to find layouts of bus systems which are "good". The first consideration is what design parameters one can work with. In our case, if we take the basic instruction set and speed of the processors as given, then there are four main parameters: the dimension d, the number of PEs n, the maximum number of buses per PE b, and the actual layout of the buses (that is, the specifcation of which PEs belong to which bus systems). Given d, b, and n, $L(d,b,n)$ denotes all possible bus layouts.

Given d, b, and n, one could employ a problem-oriented approach which tries to find optimal layouts in $L(d,b,n)$ to solve a given problem. In general this is quite difficult for nontrivial problems, and while this is sometimes of use for important problems such as the FFT, it runs into the additional difficulty that an optimal layout for one problem may not be optimal even for closely related problems. Further, any nontrivial layout will be optimal for some problem. For example, in $L(1,1,n)$, suppose M is a layout where the buses partition the mesh into intervals of length greater than 2. Then the problem of sending a word of information from each PE which is leftmost in a bus system in M to the rightmost PE in the same bus system can be solved in unit time by M, but by no other member of $L(1,1,n)$. Hence M is a unique optimum layout for this problem.

This leads one to believe that perhaps a better approach is to find a layout of buses which is somehow "good" for all problems, or, more correctly, which isn't very "bad" for any problem. We will define this by the time it takes to simulate all other members of $L(d,b,n)$, somewhat in the manner of Leiserson [Leis] finding Fat-trees of "equivalent" hardware resources to simulate a given parallel computer. To have M1 simulate M2, we will assign a unique label to each bus of M2, and each PE of M1 will start with the list of buses it would be attacted to in M2.

Let T(M1,M2) denote the worst-case time for M1 to simulate a step of M2, where M1 and M2 are in $L(d,b,n)$, and let $T(M1) = \max\{T(M1,M2) : M2 \in L(d,b,n)\}$. (Technically T should have a subscript to indicate d, b, and n, but as their values will always be clear from the context, they will be omitted.) Then *M is an optimal layout in $L(d,b,n)$* if $T(M) = \min\{ T(M1) : M1 \in L(d,b,n) \}$.

In general it is extremely difficult to find an optimal layout in any given $L(d,b,n)$. Of the three parameters, d and b are the most critical, in that a processor designed for given values of d, b, and n may be suitable for a wide range of n values, but will not be suitable for a wide range of d and b values since they are directely tied to the number and type of I/O ports. Therefore we will consider the values of d and b as being fixed, consider n as varying, and try to find an optimal family of layouts with a member for each n. Let $\mathcal{F}$ be a family of machines having a member $\mathcal{F}(n)$ in $L(d,b,n)$ for each n. We weaken the definition slightly and say that *$\mathcal{F}$ is an optimal family for dimension d and b buses per PE* if there is a constant C, independent of n, such that $T(\mathcal{F}(n)) \le C*T(M(n))$ for all n, where $M(n)$ is an optimal layout in $L(d,n,b)$. Thus, in this well-defined sense, an optimal family is never very bad for any problem or any number of processors. Optimal families provide a systematic way of choosing a fairly good general purpose layout as soon as the desired number of PEs is determined. The remainder of this paper will concentrate on finding optimal or nearly optimal families.

3.1 1-DIMENSIONAL MESHES

Our first result is an optimal family for $d=b=1$.

Theorem 3.1: Let $\mathcal{F}$ be the family of 1-dimensional meshes with 1 bus per PE, where the member $\mathcal{F}(n)$ of size n has bus systems which partition the PEs into intervals of length $\lceil n^{2/3} \rceil$ (except for the rightmost interval, which has length $n-\lceil n^{2/3} \rceil*(\lfloor n^{1/3} \rfloor -1)$). Then $\mathcal{F}$ is an optimal family for $d=b=1$, and $T(\mathcal{F}(n)) = \Theta(n^{1/3})$.

Proof: To prove that the family is optimal, suppose M is a member of $L(1,1,n)$. If M has k bus systems, then it must take at least k time units to transmit any information from one end of M to the other, which can be done in a single time unit with a global bus. Therefore, if $k \ge n^{1/3}$ then $T(M) \ge n^{1/3}$. If $k < n^{1/3}$, then at least one of the bus systems has more than $n^{2/3}$ PEs. In this bus system, if we consider the system as partitioned into intervals of length $n^{1/3}$, then the problem of sending a word of information from the leftmost PE in each interval to the rightmost one takes $\Omega(n^{1/3})$ time for M, but can be accomplished in a single time unit by an appropriate member of $L(1,1,n)$. Therefore T(M) is $\Omega(n^{1/3})$.

To see that $T(\mathcal{F}(n)) = \Theta(n^{1/3})$, fix n and let M be a member of $L(1,1,n)$. Assign unique labels to the buses of M and let each member of $\mathcal{F}(n)$ start with the label of the bus it

would belong to in M. To simulate a single time unit of M, each PE first performs the internal calculations it would do in M, and exchanges data with its neighbors as it would do in M. This takes only constant time. To simulate the broadcasting, if a PE was going to broadcast on a bus, it generates a record containing the bus' label and the data it would have broadcasted. This is circulated with its bus system in $\mathcal{F}(n)$, where each PE on the simulated bus performs its appropriate action upon receipt. If a PE on either end of the bus system is on the simulated bus, then it copies the circulating record. This has taken $\Theta(n^{1/3})$ time.

Now the following steps are repeated $n^{1/3}$-1 times. Each PE at the right end of a bus of $\mathcal{F}(n)$ which is holding a copy of a circulating record sends the record to its left neighbor. If the neighbor is in the same simulated bus it keeps the record, otherwise it just ignores it. The PE at the left end of each bus of $\mathcal{F}(n)$ similarly sends any held record to its right neighbor. Now any PE at the right end of a bus of $\mathcal{F}(n)$ which has just received a record broadcasts it, and if the PE at the left end of the bus is on the same simulated bus it records it, and all PEs on the same simulated bus perform their appropriate action. Then any PE at the left end of a bus of $\mathcal{F}(n)$ performs a similar broadcast.

Each step takes unit time, so this part has taked $\Theta(n^{1/3})$ time. At its end, the simulation is completed. ∎

To find optimal families for d=1 and b>1 there are many more possibilities, but they can be reduced to a managable form. The following lemma shows that members of $L(1,b,n)$ can be transformed into other members where the bus systems are more structured, without significantly altering the time to perform any algorithm. The proof is quite straightforward and its proof will be ommited. Figure 3 illustrates its use.

Lemma 3.2: Fix b. There is a constant C (depending only on b) such that, for any n and any machine M in $L(1,b,n)$, there are machines M1 and M2 in $L(1,b,n)$ such that

a) the bus systems of M, M1, and M2 can be assigned to layers 1..b ;
b) for each of M, M1 and M2, within each layer the bus systems are nonoverlapping;
c) Each bus system of M1 is a subsystem of a bus system of M at the same layer;
d) The bus systems of M1 and M2 are the same, but may be in different layers;
e) bus systems of M2 at layers with smaller numbers either don't overlap with, or completely contain, bus systems at layers with larger numbers; and
f) M2 can simulate a single time step of M in C time steps.

∎

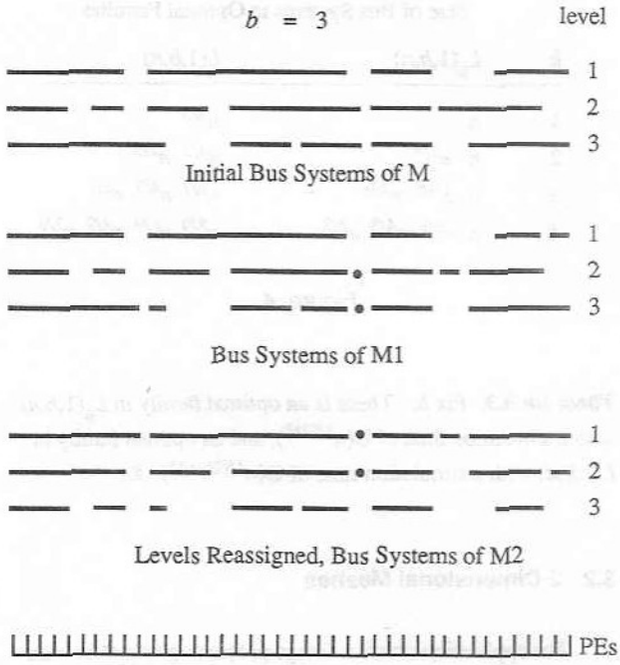


Figure 3

Using this lemma simplifies the analysis of optimal layout of bus systems where each PE can belong to at most b >1 bus systems. Let $L_g(1,b,n)$ denote those elements of $L(1,b,n)$ where at least one of the buses is a global bus. One first finds an optimal family within $L_g(1,b,n)$ by using an optimal family in $L(1,b\text{-}1,n)$, and then uses this to find an optimal family in $L(1,b,n)$.

For example, to find an optimal family in $L_g(1,2,n)$, one will partition the n PEs into submeshes of length x, which in turn will be arranged as an optimal member of $L(1,1,x)$. To choose x, the global bus can transfer values (as part of the simulation of broadcasts) from each submesh to the next in $\Theta(n/x)$ time, while Theorem 3.2 shows that within each submesh the simulation will take $\Theta(x^{1/3})$ time. Balancing these two gives $x = \Theta(n^{3/4})$, and a simulation time of $\Theta(n^{1/4})$. Straightforward arguments then show that this is indeed optimal. Figure 4 shows optimal families for small values of b.

We note that the proof of Theorem 3.1 used information about optimal families of $L_g(1,1,n)$ to find an optimal family of $L(1,1,n)$, although it was not stated in such terms. Once the proof was reduced to the consideration of the existence of a bus system of size at least $n^{2/3}$, the analysis of this bus system is really just a consideration of an optimal layout in $L_g(1,1,n^{2/3})$.

Size of Bus Systems in Optimal Families

b	$L_g(1,b,n)$	$L(1,b,n)$
1	n	$n^{2/3}$
2	n $n^{2/4}$	$n^{4/5}$ $n^{2/5}$
3	n $n^{4/6}$ $n^{2/6}$	$n^{6/7}$ $n^{4/7}$ $n^{2/7}$
4	n $n^{6/8}$ $n^{4/8}$ $n^{2/8}$	$n^{8/9}$ $n^{6/9}$ $n^{4/9}$ $n^{2/9}$

Figure 4

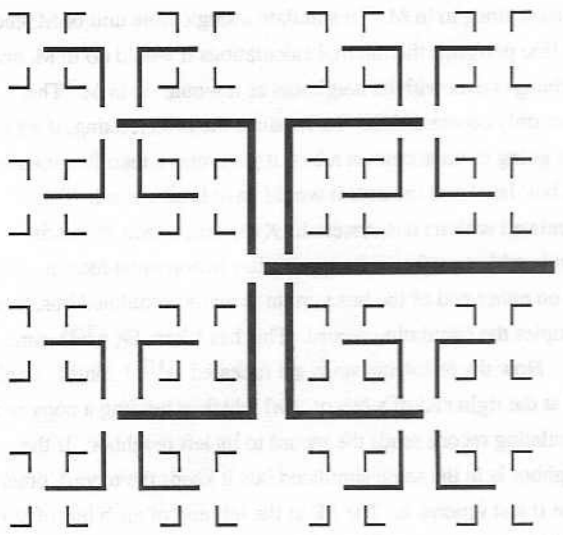
Theorem 3.3: Fix b. There is an optimal family in $L_g(1,b,n)$ with a simulation time of $\Theta(n^{1/(2b)})$, and an optimal family in $L(1,b,n)$ with a simulation time of $\Theta(n^{1/(2b+1)})$. ∎

3.2 2-Dimensional Meshes

For applications such as image processing or matrix manipulation, 2-dimensional meshes are of more interest than 1-dimensional ones, and large meshes such as the MPP [Bat] are 2-dimensional. Keeping the requirement that bus systems are connected, there are suddenly many more possibilities. For example, in $L(1,1,n)$, if each PE starts with a value and some unit-time semigroup operation is to be applied to all the values, then it is easy to show that $\Omega(n^{1/3})$ time is required. (This can be attained using the family in Theorem 2.1 .) Moving up a dimension, in $L(2,1,n)$ this can be computed in tree-like fashion in $\Theta(\log n)$ time if the bus systems are arranged in an "H-tree" configuration shown in Figure 5, where only the PEs at the ends of buses actually broadcast on, or listen to, their bus.

To find an optimal family in $L(2,1,n)$, we first note that the problem of moving a value from the leftmost PE in each row to the rightmost one can be accomplished in unit time if there is a global bus per row, and a similar fact is true for columns. A counting argument can be used to show that any layout in $L(2,1,n)$ must take $\Omega(n^{1/4})$ worst-case time to simulate these two, so any optimal family must have a simulation time of $\Omega(n^{1/4})$.

One family which achieves this simulation time is based on the pyramid computer. The pyramid has a series of layers, each of which is a 2-dimensional mesh. Each layer has 1/4 as many PEs as the layer above, and each PE is connected to 4 PEs below (its children) and one PE above (its parent), in addition to the mesh connections on the same level. The pyramid is not naturally a member of $L(2,1,n)$ since normally one obtains an embedding of a pyramid computer into the plane by adding



Heavier lines indicate buses nearer to the root.

"H-tree" layout of buses in 2-dimensional mesh

Figure 5

mesh connections between the nodes at each level of the H-tree. This introduces crossings of the edges of the embedded pyramid, so the resulting layout would either not have contiguous bus systems, or would have more than 1 bus per PE. To avoid this, wherever there is a crossing, the simulated wire corresponding to a parent/child connection stays as it was, while the wire corresponding to a mesh connection is broken into two pieces.

It is easy to show that no simulated wire is broken more than once. To simulate a step on the pyramid computer, information traveling along a broken wire uses the real mesh connections to pass from one piece (bus) to another, so a time step in the pyramid computer takes a constant amount of time to simulate.

Theorem 3.4 Bus versions of pyramid computers are an optimal family in $L(2,1,n)$, with a simulation time of $\Theta(n^{1/4})$.

Sketch of proof: The fact that this time is best possible was mentioned above. The final part needed is a pyramid algorithm to accomplish arbitrary simulation of broadcasting. Since each bus in a member of $L(2,1,n)$ is connected, the simulation can be performed by a straightforward use of the image-based pyramid computer algorithm for component labeling [M-S1]. This pyramid algorithm needs $\Theta(n^{1/4})$ worst-case time. ∎

For $b \geq 2$ far faster simulation times are possible. The only lower bound we can prove on is $\Omega(\log n)$, which is slightly less than the best time we can achieve. Therefore we cannot claim to have found an optimal family, but it is nearly optimal. Further, we suddenly have the ability to use a family with only 2 buses per PE to provide nearly optimal simulations of b buses per PE for any $b \geq 2$. One can immitate the "mesh-of-trees" or "orthogonal trees" organization [Lei1, Lei2], using $b=2$, by taking a strip of $\lg(n)$ rows (or columns) and partitioning them into buses, where the first row in each strip has buses of length 2, and each row has buses twice as long as the preceeding one. This is illustrated in Figure 6. Showing a lack of originality, this configuration will be called *orthogonal trees of buses.*

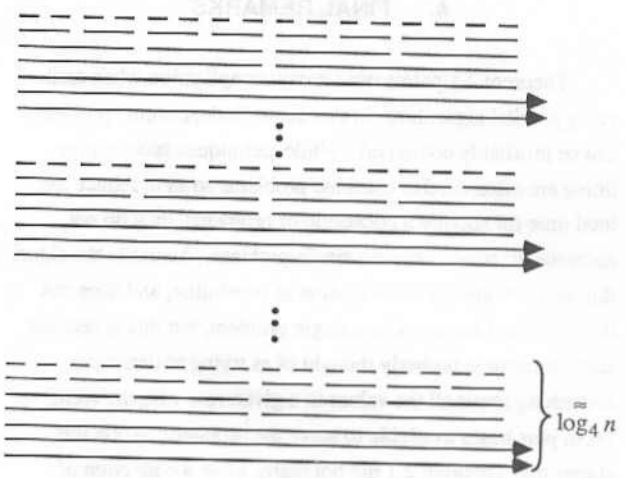
Theorem 3.5: There is a constant C, independent of any parameters, such that if M is a 2-dimensional mesh of size n with contiguous bus systems, where each PE belongs to at most b bus systems, then the orthogonal trees of buses of size n can simulate a step of M in no more than $Cb(\lg n)^2/\lg(\lg n)$ steps.
Sketch of proof: The algorithm uses divide-and- conquer to simulate the passing of messages along buses, and is in fact essentially an algorithm for labeling connected components in digitized images on an orthogonal trees computer [M-S2]. Messages which must pass through the boundaries of squares of size s are passed, and then squares of size $s*\lg(n)$ are used, and so on, in an "upward" movement of messages. Then a "downward" movement is used to distribute messages in a square of size $s*\lg(n)$ to each subsquare of size s which should receive the message. Each stage takes logarithmic time. ∎

Besides providing a nearly optimal family with the ability to also efficiently simulate 2-dimensional meshes with more resources (buses per PE), the orthogonal trees of buses can directly provide fast solutions to a range of problems. As an example, we note the following.

Theorem 3.6: The orthogonal trees of buses can solve the following problems in poly-log time:

a) Given the weight matrix of an undirected graph, label its components and mark a minimal weight spanning forest.
b) Given the adjacency matrix of a directed graph, mark a directed spanning forest.
c) Given the adjacency matrix of an undirected graph, decide if the graph is bipartite, find the cyclic index of the graph, mark all bridge edges of the graph, mark all articulation vertices of the graph, and decide if all components are biconnected.
d) Given a black/white image, label its connected components, find the nearest neighbor of each component, and decide if each component is convex.

Proof: See [Hua, Lei1, Lei2, M-S2] for the orthogonal tree algorithms. ∎



Row bus systems for orthogonal trees of buses.
The column bus systems are the same, but rotated.

Figure 6

3.3 Higher Dimensional Meshes

The orthogonal trees of buses design can be extended to higher dimensions even for $b=1$ since there is enough"room" to have buses going in orthogonal directions without intersecting. Using the same ideas as before, we obtain:

Theorem 3.7: Fix the dimension d. There is a constant C, depending only on d, such that if M is a d-dimensional mesh of size n with contiguous bus systems, where each PE belongs to at most b bus systems, then, after initialization, the d-dimensional orthogonal trees of buses of size n can simulate a step of M in no more than $Cb(\lg n)^2/\lg(\lg n)$ steps. ∎

4. FINAL REMARKS

Theorem 2.1 points out a nasty complication when analyzing parallel algorithms, in that some "independent" problems can be profitably combined. While techniques such as pipelining are often used to combine problems so as to reduce the total time for solving a collection of problems, they do not decrease the time to solve a single problem. Actually, the algorithm in Theorem 2.1 is a variation of pipelining, and does not decrease the time to solve a single problem, but this is because each problem is properly thought of as trying to determine something about all the values in a given row with the entire mesh plus buses available to solve the problem. While this shows that Theorem 2.1 did not really solve a collection of problems faster than any single one could be solved, it does not fully explain the more subtle difference that enables us to find the first 1 in each row faster than the sum of the 1s in each row.

In section 3 we defined a notion of optimal family of machines subject to resource limitations (the dimension d and number of buses attached to any PE b). This definition is in terms of the maximum time it takes to simulate any single step of any other machine with the same resource limitations, a definition which seems very robust in that an optimal machine will never be very "bad" on any problem. For dimension 1 we gave optimal families with layered buses organized in a tree-like manner, and optimal simulation times decrease like $\Theta(n^{1/(1+2b)})$ as b increases. For dimension 2, b=1 has an optimal family based on pyramid computers, with a simulation time of $\Theta(n^{1/4})$, while for $b\geq 2$ nearly optimal families based on orthogonal trees can achieve simulation times of $\Theta(\lg(n)^2/\lg(\lg(n)))$. Further, orthogonal trees, which have b=2, are nearly optimal at simulating any machine with $b\geq 2$, showing an instance where increasing a resource (the maximum number of buses per PE) does not provide much benefit. A similar situation holds in all higher dimensions for $b\geq 1$.

We note that all of these simulations need only a constant amount of space per PE (depending on b and d), and no extra initiation time. The similations are really all modifications of connected component labeling algorithms, where the difference is that each PE knows the name of the "components" it is in (the bus systems), but needs to find the new "label" of the component (the broadcasted message). Viewed this way, it is easy to see that with slight changes we could relax the assumption that the machine being simulated had no collision resolution on its buses, while still using a simulator with no collision resolution. If the collision resolution was of any simple form such as transmitting the smallest message or a random one (the sort of resolutions used in PRAMs with concurrent write capabilities), then the collision resolution can be incorporated in the simulation algorithm since the collision resolution is essentially just the same as the resolution of deciding what label to assign to a component.

For all of our 1-dimensional results, and for the 2-dimensional results with b=1, this will not alter the optimal simulation times. For the other results, which used orthogonal trees, the times will increase slightly to $\Theta(\log^2 n)$ since this is best known time for component labeling on orthogonal trees [M-S2]. It is an open question whether one must really pay this slight penalty (a factor of $\log(\log n)$) to simulate collision resolution on a machine without it.

REFERENCES

[Agg] A. Aggarwal, "Optimal bounds for finding maximum on array of procesors with k global buses", *IEEE Trans. Computers 35* (1986), pp. 62-64.

[A-H] M.J. Atallah and S.E. Hambrusch, "Solving tree problems on a mesh-connected processor array", *Proc. 26th Symp. on Found. of Comp. Sci.* (1985), pp. 222-231.

[A-K] M.J. Atallah and S.R. Kosaraju, "Graph problems on a mesh-connected processor array", *JACM 31* (1984), pp. 649-667.

[Bat] K.E. Batcher, "Design of a massively parallel processor", *IEEE Trans. Computers 29* (1980), pp. 836-840.

[Bok] S.H. Bokhari, "Finding maximum on an array processor with a global bus", *IEEE Trans. Computers 33* (1984), pp. 133-139.

[HCS] D.S. Hirschberg, A.K. Chandra, and D.V. Sarwate, "Computing connected components on parallel computers", *Comm. ACM 22* (1979), pp. 461-464.

[Hua] M.-D.A. Huang, "Solving some graph problems with optimal or near-optimal speedup on mesh-of-trees networks", *Proc. 26th Symp. on Found. of Computer Sci.* (1985), pp. 232-240.

[Jor] H.F. Jordan, "A special purpose architecture for finite element analysis", *Proc. 1976 Intl. Conf. on Parallel Proc.*, pp. 263-266.

[Lei1] F.T. Leighton, "New lower bound technique for VLSI", *Proc. 22nd Symp. on Found. of Comp. Sci.* (1981), pp. 1-12.

[Lei2] F.T. Leighton, "Parallel computation using meshes of trees", *Proc. 1983 Int.Workshop on Graph Theoretic Concepts in Computer Science* .

[Leis] C.E. Leiserson, "Fat-Trees: Universal networks for hardware-efficient supercomputing", *IEEE Trans. Computers 34* (1985), pp. 892-901.

[M-S1] R. Miller and Q.F. Stout, "Data movement techniques for the pyramid computer", *SIAM J. Computing*, to appear.

[M-S2] R. Miller and Q.F. Stout, "Some algorithms for the mesh-of-trees and hypercube", submitted.

[PK-R1] V.K. Prasanna Kumar and C.S. Raghavendra, "Array processor with multiple broadcasting", Tech. Report, Dept. of Elec. Eng. Systems, Univ. Southern Cal., 1984.

[PR-R2] V.K. Prasanna Kumar and C.S. Raghavendra, "Image processing on enhanced mesh connected computers", *Proc. Comp. Arch. for Pat. Anal. and Image Database Man.*, 1985, pp. 243-247.

[Sto1] Q.F. Stout, "Broadcasting on mesh-connected computers", *Proc. 1982 Conf. on Info. Sci. and Sys.*, pp. 85-90.

[Sto2] Q.F. Stout, "Mesh-connected computers with broadcasting", *IEEE Trans. Computers 32* (1983), pp. 826-830.

[Sto3] Q.F. Stout, "Tree-based graph algorithms for some parallel computers", *Proc. 1985 Intl. Conf. on Parallel Proc.*, pp. 727-730.

[T-V] R.E. Tarjan and U. Vishkin, "Finding biconnected components and computing tree functions in logarithmic parallel time", *Proc. 25th Symp. on Found. of Computer Sci.* (1984), pp. 12-20.

[Ull] J.D. Ullman, *Computational Aspects of VLSI* , Computer Sci. Press, 1984.