In *Proc. 1985 International Conference on Parallel Processing*, pp. 697–699.

Varying Diameter and Problem Size in Mesh-Connected Computers

(Preliminary Version)

Russ Miller*
Mathematical Sciences
State University of New York
Binghamton, NY 13901 USA

Quentin F. Stout†
Elec. Eng. and Comp. Sci.
University of Michigan
Ann Arbor, MI 48109 USA

Abstract

On a mesh-connected computer, moving data across the mesh is the most time-consuming operation in many algorithms. This time can be reduced by using a mesh with smaller diameter, i.e., with fewer processing elements. To accomodate inputs of the same size, this requires that the processors have more memory. For image processing and graph theoretic algorithms we analyze the time as a function of the mesh diameter and problem size. We show that for many problems, smaller diameters can yield faster algorithms, and that there is a choice of diameter that is simultaneously best for several of these problems. Further, for these problems and this number of processing elements (or any smaller number), the mesh is an optimal interconnection scheme.

1 Introduction

Mesh-connected computers (hereafter called *meshes*) have long been suggested for a variety of problems, and several meshes have been built [DL,HF,Re,Ro]. Their nearest neighbor connections reduce the interconnection costs, and many input formats, such as matrices or digitized pictures, map naturally onto the mesh. However, the mesh is often critisized as being too slow, having a communication diameter which grows as the square root of the number of processing elements, instead of logarithmically as occurs in some other interconnection schemes. In this paper we show that if one reduces the communication diameter by using fewer processing elements, then many problems can be solved more quickly. While this fact is common knowledge for simple problems such as finding a minimum, we show that it is also true for higher level problems in image processing and graph theory.

The standard (2-dimensional) *mesh of size n* (n a perfect square) is an SIMD machine in which n *processing elements* (*PEs*) are arranged in a square lattice. For $1 \leq i, j \leq n^{1/2}, PE(i,j)$ is connected via unit time communication links to $PEs(i \pm 1, j)$ and $PEs(i, j \pm 1)$, if they exist. Each PE has a fixed number of words of memory, each of length $\Omega(\log n)$, and all operations take unit time. This model appears in [DR,MS,NS,Ro,RS,St,TK].

There are many algorithms for a mesh of size n which finish in $\Theta\left(n^{1/2}\right)$ time [DR,MS,NS,RS,St,TK]. This is not surprising, for in this time the n PEs can do $\Theta\left(n^{3/2}\right)$ operations, and for many of the problems considered there are serial algorithms needing only $\Theta(n)$ or $\Theta(n * \log n)$ operations. Unfortunately, it takes $\Omega\left(n^{1/2}\right)$ time for information to cross the mesh, so the $\Theta\left(n^{1/2}\right)$ time is necessary. The only way to reduce the running time while keeping all the advantages of a mesh is to use a smaller mesh. (One could add new interconnections or busses, but these involve additional cost.) That is, if the original problem had n pieces of data initially distributed one per PE in a mesh of size n, we now consider what happens if it is in a mesh of size s, $1 \leq s \leq n$, where each PE is initially given n/s pieces of data. This reduces the diameter to $2\left(s^{1/2} - 1\right)$, but it sacrifices some of the parallelism. For several problems, we analyze the fastest possible algorithm as a function of n and s, and determine the optimal value of s. To make the comparisons meaningful, the speed of the PEs is fixed, and only the amount of memory given each PE is changed, along with the number of PEs.

Throughout, all analyses consider worst-case time. Because of space limitations, proofs have been omitted or sketched. Complete proofs will appear in the final version.

2 Bounds

For simple semigroup problems such as summing, finding the minimum, parity, etc., there is an easy lower bound. First, no

*Current address: Department of Computer Science, 226 Bell Hall, State University of New York, Buffalo, NY 14260 USA

†Research supported by National Science Foundation Grant MCS-83-01019 and Naval Research Lab. Contract 65-2068-85.

PE can receive data from all other PEs in less than $2*\lfloor s^{1/2}/2 \rfloor$ time units. (Only a center PE can communicate with all others this rapidly.) Second, it takes each PE at least n/s time to examine all of its initial data. Thus any nontrivial semigroup operation requires at least $\max\{2*\lfloor s^{1/2}/2 \rfloor, n/s\}$ time. Further, this bound holds for any problem of size n (i.e., initially having n words of data) in which each PE must examine all of its initial data, and in which there is at least one PE which has to receive information which might originate in any of the PEs. Such a problem is said to require *global communication*. This bound is minimized when $s = \Theta(n^{2/3})$, so we obtain:

2.1 Proposition *On a mesh of size s, for any problem of size n requiring global communication, at least $\max\{2 * \lfloor s^{1/2}/2 \rfloor, n/s\}$ steps are needed. Therefore, for a problem of size n, no matter what size mesh is used, $\Omega(n^{1/3})$ time is needed.* □

If the problem is to compute a commutative semigroup operation, and if the operation can be computed in unit time, then for all n and s this time can be achieved. This is accomplished by having each PE first apply the operation to all of its initial data, and then sending these values towards a center PE, combining them in transit. For any n and s, this computes the semigroup operation in $\Theta(n/s + s^{1/2})$ time, and in particular finishes in $\Theta(n^{1/3})$ time when $s = \Theta(n^{2/3})$. This observation is a sort of folk theorem.

Unfortunately, not all problems with global communication can achieve this lower bound. For example, suppose n initial values are to be sorted. Even in the expected case, half of the values on the right half of the mesh must cross to the left half, and vice versa. There are only $s^{1/2}$ wires crossing the centerline, so $\Omega(n/s^{1/2})$ time is needed. Since $n \geq s$, this is minimized when $s = n$, for which a $\Theta(n^{1/2})$ algorithm is known [TK]. Using $s = \Theta(n^{2/3})$ takes $\Theta(n^{2/3})$ time.

Hence no single value of s as a function of n can give optimal solutions to all problems. However, we will show that $s = \Theta(n^{2/3})$ is optimal for many nontrivial problems, and therefore seems to be a good choice.

3 Images

In this section, the input to each problem is an $n^{1/2} \times n^{1/2}$ array of black/white *pixels* (picture elements), which is thought of as a white background with black figures. This is a *picture of size n*. When stored in a mesh of size s, each PE receives a $(n/s)^{1/2} \times (n/s)^{1/2}$ subsquare. Two black pixels are *adjacent* if they share an edge, and are *connected* if there is a path of adjacent black pixels from one to the other. The *component labeling problem* is to assign a label to each black pixel, where two black pixels receive the same label if and only if they are connected. The set of black pixels with the same label is a *figure*. [NS] gives a $\Theta(n^{1/2})$ time component labeling algorithm for the mesh of size n. Notice that this problem requires global communication.

3.1 Theorem *There is an algorithm, independent of s and n, that labels the components in $\Theta(n/s + s^{1/2})$ time when given a picture of size n stored in a mesh of size s, $1 \leq s \leq n$.*

Outline of the algorithm: Each PE uses a serial algorithm to label its subsquare in $\Theta(n/s)$ time. Then the subsquares are connected up, correcting labels of figures which are in two or more subsquares. Such figures must touch the boundary of the subsquare, so each PE need only correct $\Omega((n/s)^{1/2})$ labels in its subsquare. As in [NS], information concerning such labels is joined together in increasingly larger squares. The [NS] algorithm must be modified to account for the fact that a PE may have many labels to correct, but this can be done in the indicated time bounds. □

Two important features of the above algorithm are the linear-time serial algorithm which can be applied to initial subsquares, and the fact that squares can be combined by considering an amount of data which grows as the perimeter of the squares, i.e., as the square root of the size of the square. These two features are present in many other problems.

For problems involving distance or convexity, $PE(i, j)$ is identified with the integer lattice point (i, j). Any l_p metric can be used in the algorithms below, although in practice usually just the l_1 ("taxi-cab" or "city block"), l_2 ("Euclidean"), or l_∞ ("chessboard") metrics are used. The *distance between sets A and B* is defined to be $\min\{d(a, b) : a \in A,\ b \in B\}$, where d is the given metric. Given a black/white picture with its components labeled, the *nearest neighbor problem* is to determine, for each component, the closest figure to it. The *diameter problem* is to determine the external diameter of each figure, where the external diameter of a set A is $\min\{d(a, b) : a, b \in A\}$.

The *smallest box problem* [FS] is to determine a (not necessarily unique) smallest rectangle containing each figure. The *area problem* is to determine the number of pixels in each figure, and the *perimeter problem* is to determine the number of pixels on the border of each figure.

The *convex hull problem* is to find the extreme points of each figure, i.e., the corners of the smallest convex polygon containing the set. (Other definitions of convexity are possible [KR].) The *disjoint convex hulls problem* is to decide, for each figure, if its convex hull intersects the convex hull of any other figure.

For a picture of size n stored in a mesh of size n, all of the above problems can be solved in $\Theta(n^{1/2})$ time [DR,MS]. These problems have the two properties mentioned above for the component labeling problem, which allows us to build the following algorithms.

3.2 Theorem *For each of the nearest neighbor, diameter, smallest box, area, perimeter, convex hull, and disjoint convex hull problems, there is an algorithm, independent of n and s, that solves the problem in $\Theta(n/s + s^{1/2})$ time when given a picture of size n stored in a mesh of size s, $1 \leq s \leq n$.* □

4 Graphs

In this section the input is an adjacency or weight matrix for a graph of $n^{1/2}$ vertices, where each PE receives a $(n/s)^{1/2} \times (n/s)^{1/2}$ submatrix. This input format is a *graph of size n*. Only undirected graphs with positive weights will be considered. The *component labeling problem* for graphs is to assign a label to each vertex, with two different vertices receiving the same label if and only if there is a path between them. The *minimal spanning forest problem* is to determine the edges in a minimal weight spanning tree of each component. It has been noted that an algorithm for one of these problems is easily modified to solve the other [CLC,SJ].

On a standard mesh of size n, both of these problems can be solved in $\Theta(n^{1/2})$ time [AK], and on a serial computer a simple depth-first search solves them in $\Theta(n)$ time. On a mesh of size s, if one uses the serial algorithm on the subsquares followed by the [AK] algorithms to combine squares, the time will be $\Theta(n^{3/2}/s)$ because of the matrix calculations in [AK]. Replacing the matrix calculations with the edge-based graph algorithms in [RS] allows one to rapidly combine the squares, giving:

4.1 Theorem *For the component labeling and minimal spanning forest problems, there is an algorithm, independent of n and s, that solves the problem in $\Theta(n/s + s^{1/2})$ time when given a graph of size n stored in a mesh of size s, $1 \leq s \leq n$.* □

There are several other graph problems which can be solved in the same time, again using edge-based graph algorithms to combine squares. We refer to [St] for definitions and the appropriate edge-based graph algorithms.

4.2 Theorem *There are algorithms, independent of n and s, that take a graph G of size n on a mesh of size s, $1 \leq s \leq n$, and decide if G is bipartite, find the cyclic index of G, find all bridge edges of G, find all articulation points of G, and decide if all components of G are biconnected, in $\Theta(n/s + s^{1/2})$ time.* □

5 Optimality

There are several ways to express the optimality of the results obtained in the previous sections. First we show there is a choice of s which is simultaneously optimal for all of the problems considered, and then we show that the smaller diameter mesh is an optimal interconnection scheme.

5.1 Theorem *For each of the component labeling, nearest neighbor, diameter, smallest box, area, perimeter, convex hull, disjoint convex hulls, component labeling for graphs, minimal spanning forest, bipartite, cyclic index, bridge edge, articulation point, and biconnectedness problems, if $s = n^{2/3}$ then there is an algorithm solving the problem in $\Theta(n^{2/3})$ time. Since each of these problems requires global communication, this is an optimal choice of s for all of them.* □

Suppose an algorithm A takes $T(n, s)$ time to solve a problem of size n on a mesh of size s. A is said to have *linear speed-up in the range* $1 \leq s \leq U(n)$ if there are positive constants C and D, independent of n and s, such that $C * s \leq T(n, 1)/T(n, s) \leq D * s$ for arbitrary n and $1 \leq s \leq U(n)$. An algorithm A is (worst-case) *optimal* for a uniprocessor if there is a positive constant C so that, for any size i of input, for any algorithm B solving the same problem, the worst-case time for B on an input of size i is at least C times the worst-case time for A on an input of size i.

5.2 Theorem *For each of the problems mentioned above, there is an algorithm, independent of n and s, that is optimal for a uniprocessor ($s = 1$) and that exhibits linear speed-up in the range $1 \leq s \leq n^{2/3}$.* □

Perhaps the most surprising feature of these algorithms is that any parallel computer solving any of the problems mentioned above must do $\Omega(n)$ calculations, no matter how many PEs there are nor how they are interconnected. For $1 \leq s \leq n^{2/3}$, a mesh of size s does only $\Theta(n)$ calculations when using the above algorithms, giving the following result:

5.3 Theorem *For each of the problems mentioned above, given a problem of size n, the mesh of size s is an optimal interconnection scheme among all computers with s PEs for $1 \leq s \leq n^{2/3}$, meaning that there is a constant C which depends only upon the problem, such that any other computer with s PEs (with the same instruction set and speed of operation for each instruction) must take time which is at least C times that used by the mesh.* □

6 Summary

For image data and graph data, we have analyzed the effect of using a mesh of size s on a problems of size n, where $1 \leq s \leq n$ and each PE has $\Omega(n/s)$ words of memory. We have shown that, for many problems, there is a speed-up possible by using a mesh with $s < n$, and that choosing $s = n^{2/3}$ gives optimal times. Further, for these problems and this choice of

s (or any smaller value), the mesh achieves linear speed-up and is an optimal interconnection scheme.

For some reason, meshes with extra memory per PE have not had much attention, though there are notable exceptions such as the Illiac IV. In particular, most image processing meshes have been built with very little memory per processor, and discussions of these machines often concentrate on their speed for local tasks, such as edge detection, which do not require global communication. We believe that the problems considered here are more indicative of higher-level problems, and that building image processing machines with more memory per PE will decrease the total time needed to analyze a picture.

However, it must be determined whether using $s < n$ will slow down input/output operations, and for some problems, such as sorting, we have shown that the algorithms must be slower when $s < n$. The question of whether overall, in some "standard" mix of problems, choosing $s = n^{2/3}$ is best will depend on more analysis, and requires a better view of the relative importance of various problems. Whatever the results, it will be useful to have more analyses which consider the situation when input size exceeds the number of PEs, since the owner of any given machine sees the problem sizes increase rapidly.

References

[AK] M.J. Atallah and S.R. Kosaraju, Graph problems on a mesh-connected parallel processor array, *J. ACM* 31 (1984), 649–667.

[CLC] F.Y. Chin, J. Lam, and I.-N. Chen, Efficient parallel algorithms for some graph problems, *C. ACM* 25 (1982), 659–665.

[DL] P.E. Danielsson and S. Levialdi, Computer architectures for pictorial information systems, *IEEE Computer* 14 (1981), 53–67.

[DR] C.R. Dyer and A. Rosenfeld, Parallel image processing by memory augmented cellular automata, *IEEE T. on PAMI* 3 (1981), 29–41.

[FS] H. Freeman and R. Shapira, Determining the minimal-area encasing rectangle for an arbitrary closed curve, *C. ACM* 18 (1975), 409–413.

[HF] K. Hwang and K-s. Fu, Integrated computer architectures for image processing and database management, *IEEE Computer* 15 (1982), 51–60.

[KR] C.E. Kim and A. Rosenfeld, Digital straight lines and convexity of digital regions, *IEEE T. on PAMI* 4 (1982), 149–153.

[MS] R. Miller and Q.F. Stout, Geometric algorithms for digitized pictures on a mesh-connected computer, *IEEE T. on PAMI* 7 (1985), 216–228.

[NS] D. Nassimi and S. Sahni, Finding connected components and connected ones on a mesh-connected parallel computer, *SIAM J. Comput.* 9 (1980), 744–757.

[Re] A.P. Reeves, Parallel computer architectures for image processing, *Comp. Vision, Graphics, and Image Proc.* 25 (1984), 68–88.

[Ro] A. Rosenfeld, Parallel image processing using cellular arrays, *IEEE Computer* 16 (1983), 14–20.

[RS] J. Reif and Q.F. Stout, Optimal component labeling algorithms for mesh computers and VLSI, to appear.

[SJ] C. Savage and J. Ja'Ja, Fast, efficient parallel algorithms for some graph problems, *SIAM J. Comput.* 10 (1981), 682–691.

[St] Q.F. Stout, Tree-based graph algorithms for some parallel computers, *Proc. 1985 Int'l. Conf. on Parallel Proc.*, 727–730.

[TK] C.D. Thompson and H.T. Kung, Sorting on a mesh-connected parallel computer, *C. ACM* 20 (1977), 263–271.