I. INTRODUCTION

The coming use of multiprocessor systems necessitates the development of a general methodology for utilizing such systems efficiently, something like the methodology of algorithm design and analysis for single-processor systems. There is a considerable added difficulty, however, in that the underlying processor interconnection may now be part of the design, and different interconnections may cause performance to vary drastically. As a first step towards such a methodology, it seems important to identify the parameters that capture the performance of parallel algorithms in a machine-independent fashion (and could therefore be used in optimizing machine architecture).

We use time and communication as our principal performance criteria.

To begin with, we model the computational problem to be solved as a directed acyclic graph (dag), with nodes corresponding to computed values, and arcs denoting dependencies (i.e., the children of a node are the values used to compute the node). Naturally, in most cases this dag is part of the algorithm design sought, but it seems reasonable, at least at this point, to consider the dag fixed and given. We model the calculation of such a dag in a multiprocessor system by assigning to each node one or more processors that compute the node. (Note: Our proofs could be simplified considerably if we assigned a unique processor to each node, but as we shall observe, allowing several processors to compute the same value can save communication at no additional time delay.) The time to compute a dag is determined when we assign a schedule to the nodes and processors, subject to the obvious constraints that (1) a node cannot be computed unless all its children have been computed at previous time units, and (2) no processor can compute two different nodes at the same time.

The communication required by a particular schedule is the number of processor-node-child triples \((P, n, c)\), such that \(P\) computes the value of \(n\) at a time when child \(c\) of \(n\) has not yet had its value computed by \(P\). Such a node \(n\) is called a communication node. Note that a communication node may contribute either 1 or 2 (or even more) to the total communication. However, since we obtain only an order-of-magnitude result, we shall lower bound the communication by counting the communication nodes. The number of communication nodes captures one measure of communication, namely the total message traffic generated by the algorithm. Another interesting measure is the total elapsed time due to communication, measured in length of chains of interdependent messages. This matter is taken up in Section III.

Example 1: In Fig. 1 we see the \(2 \times 2\) diamond, i.e., grid turned so the children of each node are below the node. Suppose we wish to compute this dag in time 3. We might use the two-processor schedule

\[
\begin{array}{c|ccc}
\text{Time} & 1 & 2 & 3 \\
\hline
P_1 & a & b & d \\
P_2 & c & & \\
\end{array}
\]

This schedule uses two units of communication; the value \(a\) is passed from \(P_1\) to \(P_2\) after time 1, and the value of \(c\) is passed from \(P_2\) to \(P_1\) after time 2.

Note that if we allow \(P_2\), which is idle at time 1, to compute \(a\), then the communication is reduced to 1.

† We do not care whether or not \(c\) is later computed by \(P\).

Fig. 1. Diamond dag.
This dag is an example where allowing more than one processor to compute a node is essential for minimizing the communication, given a time limit.

For many computation dag there is no real issue regarding the best way to trade communication for time.

**Example 2:** (Ernst Mayr) Consider dag that are complete binary trees of \( n \) nodes. If we allow \( k \) processors to cooperate on the computation, the time is constrained by \( t = \Omega(n^2/k) \). As each processor must either compute the root or compute a value used by some other processor, the communication is constrained by \( c \geq k - 1 \). Thus, \( ct = \Omega(n) \) whenever more than one processor is used to compute the tree.

That is really all there is to the complete binary tree, because we can find a partition of the nodes so that both bounds, on \( c \) and \( t \), are met simultaneously.

---

**The Diamond Dag**

The diamond dag is the generalization of Fig. 1 to an \( n \times n \) square rotated 45 degrees, where the children of each node are the nodes immediately to the southeast and southwest, if they exist. It turns out that the diamond does not share with the tree of Example 2 the nice property that the best lower bounds on time and communication can be met simultaneously. Rather, there is a law about the product \( ct \) that is stronger than what is implied by the best lower bounds on \( c \) and \( t \) individually.

This interesting tradeoff law is one of a number of surprising properties of this dag. For example [11K] showed a tradeoff involving time and the amount of external memory needed to compute the diamond on a chip. In [CS] and [K], the "pyramid," which is the upper half of the diamond, is studied from the point of view of single processor computations. The pyramid represents the straightforward algorithm for estimating the \( n \)th derivative at a point, while the diamond represents the straightforward way of computing longest common subsequences (see [AHU], e.g.).

Suppose we compute an \( n \times n \) diamond using \( k \) processors. We can argue that \( t \geq n^2/k \), since at least one processor must compute \( n^2/k \) nodes. We can meet this lower bound, to within a constant factor, by the stripes method shown in Fig. 2. There, the first processor begins to compute the rightmost diagonal of the first stripe. After \( n/k \) time units, it has reached the end of that diagonal, and the second processor can begin. Similarly, within \( n/k \) time units, all the processors are working on their stripes, and values will always be available so no processor is idle until it finishes its stripe.

Now, let us consider a lower bound on communication. If we divide responsibility for computation of the nodes among \( k \geq 2 \) processors evenly, then each computes \( n^2/k \) nodes, and it is not hard to show that there must be among the nodes computed by any one of these \( k \) processors at least \( \sqrt{n^2/k} = n/k \) communication nodes. Thus, \( c = \Omega(n/k) \).

We can meet this lower bound by the partition suggested in Fig. 3. There, the number of communication nodes for each processor's territory is \( O(n/k) \), so the total amount of communication is \( O(n^2/k \). However, it is possible to show that only one processor in any column can be active at any time, so the duration of the computation is \( O(n^2/k \), rather than \( O(n^2/k \) as we might have hoped.

Note that both Fig. 2 and Fig. 3 represent schedules where exactly one processor computes each node. However, no modification of Fig. 2 where we allow several processors to compute a node can improve the communication in Fig. 2, nor can it improve the time in Fig. 3.

---

**II. THE MAIN THEOREM**

We prove the following result.

**Theorem 1:** Any schedule and assignment of processors that computes all the nodes of an \( n \times n \) diamond dag, uses communication \( c \), and takes time \( t \), where \( t = o(n^2) \), must satisfy \( ct = \Omega(n^2) \).

(Note: If we relax the condition that \( t = o(n^2) \), then
we could use one processor and have $c = 0$. However, another useful way to state the bound is $(c + n)t = \Omega(n^3)$, so we cover the case $t = n^2$. In fact, this latter statement of the Theorem has an intuitive interpretation, in that we can think of $c + n$ as the total communication of the computation, including communication due to inputting the $n$ values at the lower boundaries. This latter kind of communication is present even in the single-processor case.

We shall prove the theorem by a series of lemmas. The argument centers around a partition of the nodes of the diamond into squares of side $m$ in the manner suggested by Fig. 4.\footnote{However, there is no relationship between $m$ and $k$, the number of processors used (compare with Fig. 3). In fact, it turns out that we shall ultimately wish to choose $m = 32t/n$, which makes a plausible partition of the diamond, since $t = o(n^3)$ is assumed and $t \geq n$ is obvious.}

In what follows, we focus on an arbitrary $m \times m$ subdiamond in the $n \times n$ diamond.

First note that the bottom node of a subdiamond must be computed before any other, and the top node of a subdiamond cannot be computed until all other nodes have been computed by at least one processor. The time between the first computation of the bottom node and the first computation of the top node will assume special significance; we call it the window for that subdiamond. Note that processors may continue to compute nodes of a subdiamond after the window has passed, but no node is computed by any processor before the window opens.

**Lemma 1:** Suppose that the duration of the window for some subdiamond is at most $m^2/8$. Then there are at least $m/4$ communication nodes in that subdiamond.

**Proof:** (sketch) We consider only those processors that compute one or more nodes of the upper quadrant of the subdiamond during the window. Each such processor $P$ computes at most $m^2/8$ nodes during the window, because there isn’t time to compute more. We consider two cases, depending on whether or not $P$ computes an entire diagonal from the border of the subdiamond

![Fig. 4. The partition for the Proof of Theorem 1.](image1)

![Fig. 5. The proof of Lemma 1.](image2)

![Fig. 6. A column of subdiamonds.](image3)
with window durations of at most $4mt/n$. □

In order to apply Lemma 1 to these subdiamonds, we need to know that their window duration is at most $n^2/8$, that is,

$$\frac{4mt}{n} \leq \frac{m^2}{8}$$

from which we derive $m \geq 32t/n$.

If we take $m = 32t/n$, we can apply Lemma 1 to the $n^2/4m^2$ subdiamonds that we know exist by Lemma 3. Each of these, by Lemma 1, has at least $m/4$ communication nodes, so we know

$$c \geq \left(\frac{n^2}{4m^2}\right) \left(\frac{m}{4}\right) = \frac{n^2}{16m} = \frac{n^2}{512(t/n)} = \frac{n^3}{512t}$$

from which $ct = O(n^3)$ is immediate. □

**Corollary 1**: The stripes method of Fig. 2 gives, to within a constant factor, the smallest communication and the smallest number of processors for a given time limit. The method of Fig. 3 gives the smallest communication, but not the smallest number of processors, for a given time limit. □

**III. A BOUND ON THE COMMUNICATION DELAY**

We next study a tradeoff between processing time and communication delay. The communication delay $d$ of a computation is the length of the longest chain of communication nodes such that each node depends on the previous one in the computation.

**Theorem 2**: If $d$ is the communication delay of a schedule for the diamond taking time $t = o(n^2)$, then $dt = \Omega(n^2)$.

**Proof**: The computation defines node-processor pairs with dependencies between them. Let $(a, P)$ be such a pair, so that $a$ is on the main (vertical) diagonal of the diamond (see Fig. 7). Define $s(a)$ to be the largest integer so that all nodes in the (singly or doubly) shaded area are computed by the same processor $P$, with no communication (except, perhaps for the boundaries). Notice that, by the definition of $s(a)$, there will always be a communication node in the shaded area with a child $c$ just beyond the lower-left boundary of the shaded region (unless, of course, the shaded region extends to the lower edge of the diamond). Also, $s(a)$ could be one, in which case the shaded region would contain just $a$.

We are going to define a sequence of such nodes $a_j$, as follows: $a_1$ is the top node of the diamond, without loss of generality computed by just one processor, $P_1$. For a general node $a_j$, in the sequence, consider a processor $P_j$ among those that compute it, the corresponding area with side $s_j$, and, if the lower boundary of the area is not the boundary of the diamond, consider the node $c_j$ on the other side of the boundary, computed by some processor $Q_j$. We let $a_{j+1}$ and $Q_{j+1}$ be the node on the main diagonal, and the corresponding processor, so that the pair $(c_j, Q_j)$ depends on $(a_{j+1}, P_{j+1})$, for some processor $P_{j+1}$. We continue until the construction stops, after $m$ steps, because the lower boundary of the diamond has been reached.

Notice that, by our construction, each node in the doubly shaded region (see Fig. 7) that corresponds to $a_j$—call it $R_j$—depends on each node of $R_{j+1}$, and thus, if by $|A|$ we denote the area of $A$,

$$t \geq \sum_{j=1}^{m} |R_j| = \frac{1}{8} \sum_{j=1}^{m} s(a_j)^2.$$

Also, notice that $\sum_{j=1}^{m} s(a_j) = n$. Finally, the message exchanged at $c_j$ depends on that at $a_{j+1}$, and thus $d \geq m$. We conclude from these three inequalities that $dt \geq \frac{1}{8} n^2$. □

**IV. APPLICATIONS**

Let us consider a computation of the diamond dag (which we may by Corollary 1 assume is by the stripes method) using a collection of processors linked by a local area network. Such a network has a communication cost that involves a large amount of overhead to open communication between two processors, plus a smaller cost to communicate each of a number of values once the overhead is paid. Theorem 2 says that there must be some path along which $d = \Omega(n^2/t)$. Thus, $d$ overhead charges must be paid. On the other hand, Theorem 1 implies a lower bound on the total number of values communicated, and therefore, no matter how many values are bundled into one message, Theorem 1 implies a lower bound on the amount of time spent shipping values across the net. Depending on the overhead and on $n$, either Theorem 1 or Theorem 2 may...
put a more stringent limit on the number of processors that may effectively share the calculation of the diamond dag.

For a more concrete example, suppose that the overhead is \( C \), constant and independent of the traffic in the network. If we use \( k \) processors, then the "real" time needed for the computation is something like

\[
T = \frac{n^2}{k} + C \frac{n^2}{n^2 / k}
\]

(constants are ignored in these calculations). The first term is the delay due to computation time, the second is the total communication delay, by Theorem 2. The optimum number of processors to use is \( k \approx n / \sqrt{C} \), for a total time \( T \approx n \sqrt{C} \).

Usually the communication overhead depends on the traffic of the network; for example, the ethernet throughput is measured in messages exchanged in unit time. If we assume that our computation will distribute traffic evenly throughout its duration (as the stripes method nearly does), then the "real" time becomes

\[
T = \frac{n^2}{k} + \frac{n^2 / k}{b - \frac{n^2 / k}{n^2 / k}}.
\]

The first term is again the computation time, while the second captures delays due to communication. The numerator is the number of distinct message transmissions, by Theorem 2, while the denominator is the limit \( b \) minus the average message traffic (by Theorem 1, \( n^3 / t^2 \)). The optimum choice of number of processors is \( k \approx \sqrt{nb} \), for \( T \approx n^3 / b \).

From this discussion, we observe that results such as Theorems 1 and 2 do lead to a kind of detailed analysis of the efficient use of multiprocessors to solve a particular problem, as well as its limitations, that would be impossible without these results. We hope that such results, for wider classes of problems, will lead to a theory of parallel algorithms nearly as comprehensive as the one we have for sequential ones.

V. FURTHER REMARKS

It is interesting to consider what other dags exhibit communication-time tradeoffs similar to the diamond. We already mentioned the \( ct = \Omega(n) \) result of Mayr for the complete binary tree. Maria Klawe and Mike Paterson have shown a \( ct = \Omega(n^2) \) result for the dynamic programming dag, where the children of node \((i,j)\) are all the nodes \((k,j), k < i\) and \((i,l), l < j\). This graph models the computation of many "dynamic programming" algorithms [AHU], and can be calculated in \( O(n^2) \) sequential time. Antivahis has improved this tradeoff to \( c^2t = \Omega(n^2) \) [A]. An important question that remains open is to state generalized such tradeoffs, that hold for any dag, and involve certain graph-theoretic parameters of the dag (like the bisection width used for VLSI lower bounds [T]). Another interesting question is deriving a more detailed tradeoff, involving time, number of messages, and granularity of messages, i.e., number of values that can be sent in a single message.

ACKNOWLEDGEMENTS

We would like to thank the following people for comments on this work and on the earlier proof of a weaker theorem that assumed no node was computed by more than one processor. Thanks go to Dimitri Antivahis, Peter Hochschild, Don Knuth, Harry Mairson, Ernst Mayr, and Mike Paterson.

REFERENCES


