INFORMATION AND COMPUTATION 119, 124–128 (1995)

A Lower Bound for the Emulation of PRAM Memories on Processor Networks*

TORBEN HAGERUP

Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany

We show a lower bound of $\Omega(\min\{\log m, \sqrt{n}\})$ on the slowdown of any deterministic emulation of a PRAM memory with m cells and n I/O ports on an n-processor bounded-degree network. The bound is weak; unlike all previous bounds, however, it does not depend on the unnatural assumption of point-to-point communication which says, roughly, that messages in transit cannot be duplicated by intermediate processors. For m sufficiently large relative to n, the new bound implies the optimality of a simple emulation on a mesh-of-trees network.

1. INTRODUCTION

Parallel random access machines (PRAMs) and bounded-degree processor networks (BDNs) are synchronous parallel machines employing as computing agents a finite collection of sequential processors, each equipped with a local memory. The processors of a PRAM communicate via a shared global memory, while each processor of a BDN can communicate directly with only a constant number of neighboring processors.

The powerful PRAM model is attractive to the algorithm designer, but the direct physical realization of its shared memory with as many simultaneously active I/O ports as there are processors poses difficult or unsolvable problems. Processor networks, in contrast, are routinely built using current technology, but they are significantly less convenient to program because interprocessor communication, easy on the PRAM, represents a nontrivial problem that must be dealt with explicitly. For this reason, much effort has gone into bridging the gap between the unfeasible PRAM and the feasible BDN by means of emulations of the shared PRAM memory on a BDN (see Fig. 1).

Consider in the sequel an emulation on an n-processor BDN of a PRAM memory with m cells or *PRAM variables* and n I/O ports. Each BDN processor is connected to one I/O port in a bijective fashion. In the beginning of each memory cycle, each BDN processor obtains over its I/O port from an unspecified external agent (representing the PRAM program) either a write request, i.e., a request to update a specified PRAM variable with a specified value, or a read request, i.e., a query asking for the value of a specified PRAM variable. We can assume that the read and write requests issued within a single memory cycle all pertain to distinct variables (i.e., the memory access pattern is that of an EREW PRAM).

The task of the emulation is to satisfy all read requests; i.e., each BDN processor receiving at the beginning of a memory cycle a query for a variable x must at the end of the cycle transmit over its I/O port the current value of x, defined in the obvious way, or some fixed value if x has never received a value. The *slowdown* of a particular memory cycle is the number of network steps needed to emulate that cycle, and the (worst-case) slowdown of the emulation is the maximum slowdown of any memory cycle. Note that we require the emulation to handle arbitrary sets of read and write requests for distinct variables correctly, which corresponds to allowing nonuniform PRAM computations, i.e., computations described by a different program for each combination of values of n and m.

As demonstrated by Karlin and Upfal (1988) and Ranade (1991), there are randomized emulations with expected slowdown $O(\log n)$. Since the diameter of any n-processor BDN is $\Omega(\log n)$, this is clearly optimal. We consider here deterministic emulations, for which the picture is less clear. The best currently known upper bound of $O(\log n \log m/\log\log n)$ was given by Herley and Bilardi (1994). A lower bound of $\Omega(\min\{\sqrt{n\log n}, \log n \log m/\log\log m\})$ for the case $m = n^{2+\Omega(1)}$ was found by Alt *et al.* (1987) and, independently, by Karlin and Upfal (1988). The latter bound was extended by Herley and Bilardi to smaller values of m relative to n. In conjunction, the known results amount to a lower bound of

$$\Omega\left(\min\left\{\sqrt{n\log n}, \frac{\log n \log m}{\log\log m}, \frac{(\log(m/n))^2}{\log\log(m/n)}\right\}\right),$$

for arbitrary combinations of n and m.

* Supported in part by the Deutsche Forschungsgemeinschaft, SFB 124, TP B2, VLSI Entwurfsmethoden und Parallelität, and in part by the ESPRIT II Basic Research Actions Program of the EC under Contract 3075 (Project ALCOM).

0890-5401/95 $6.00

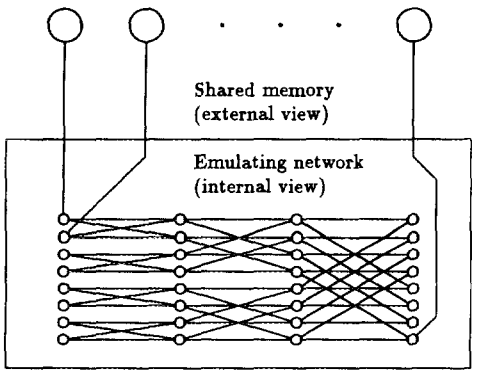

FIG. 1. Emulation of a shared PRAM memory on a bounded-degree network.

The upper and lower bounds cited above coincide for a large range of values of m relative to n. All lower bounds mentioned above, however, were derived under the assumption of so-called *point-to-point communication*. This assumption is best visualized by imagining the values of PRAM variables to be contained in sealed envelopes. An unlimited supply of such envelopes for a variable x, provided by the external agent at the relevant processor, takes the place of a write request for x, and a read request for x must be satisfied by the presence of at least one envelope for x at the relevant processor. Envelopes can be arbitrarily routed through the network, but copies of envelopes cannot be created by the network. Note that this may force the network to work harder to achieve a certain distribution of copies of the variables, i.e., of knowledge of their current values. By way of example, suppose that n processors are interconnected in the form of a complete binary tree. If the root processor wants to broadcast the value of a single variable to all other processors, this can be done in $O(\log n)$ time with no restrictions on communication, but the broadcast takes $\Omega(n)$ time under the assumption of point-to-point communication, since $n-1$ copies of the variable must leave the root processor.

Most of today's large computer networks operate according to the assumption of point-to-point communication; e.g., if a person in New York wants to send the same E-mail message to ten recipients in Europe, ten copies of the message must cross the Atlantic. Nevertheless, we argue that the assumption of point-to-point communication is quite artificial in the context of a tightly coupled multiprocessor. This is evidenced already by the difficulty of describing it in natural terms. Computers do not communicate via sealed envelopes, copying of arbitrary data items is as easy, up to a constant factor, as passing them on unread to a neighboring processor, and we have not been able to think of a physical realization of a network that would of necessity restrict itself to using point-to-point communication. The assumption of point-to-point communication was evidently introduced to make the model of computation fit the existing proof, and it might be claimed that the previous lower bounds say more about our inability to analyze the general situation than about the difficulty of the emulation.

Throughout the paper, for any $s>0$, $\log s$ denotes $\max\{\log_2 s, 1\}$. The assertion $E_1 = O(E_2)$, where E_1 and E_2 are expressions, is intended to mean that for some constant $c>0$, $E_1 \leq cE_2$ for *all* legal combinations of values of the variables occurring in E_1. The meaning of $E_1 = \Omega(E_2)$ and $E_1 = \Theta(E_2)$ is defined analogously.

2. THE NEW LOWER BOUND

In this paper, we prove the first lower bound on slowdown, apart from the trivial bound of $\Omega(\log n)$, that does not depend on the assumption of point-to-point communication. The bound, $\Omega(\min\{\log m, \sqrt{n}\})$, is quite weak. In particular, when m is polynomial in n, it coincides with the trivial bound of $\Omega(\log n)$. Its proof is a variation of the proof of (Alt *et al.*, 1987), the crucial difference being a new definition of the *redundancy* of a variable. In general terms, the redundancy of a variable is a measure of the abundance of *copies* of the variable. If the redundancy of a variable is high, reading the variable may be easy, but updating it is expensive, a trade-off that is exploited by the proofs. The previous proofs define the redundancy of a variable as the number of copies of the variable stored outside a "ball" of radius q in the network, for suitably chosen q. Under the assumption of point-to-point communication, this forces the network to spend $\Omega(rq)$ time per memory cycle maintaining an average redundancy of r, but this is not true in the case of unrestricted network communication. Our main contribution is an alternative definition that works even without the assumption of point-to-point communication. We define the redundancy of a variable simply as the number of processors holding a copy of the variable (technically, we assume the subnetwork spanned by the set of such processors to be connected; this is no restriction, however, since we can simply assume that once a processor "knows" the value of a particular variable, it never again "forgets" that value). With this definition, $\Omega(r)$ time per memory cycle is necessary to maintain average redundancy r, and we will show in Section 4 that the emulation is extremely slow unless $r = \Omega(\log m)$.

A lower bound on the slowdown of an emulation is a statement about the worst case of what may happen within a single memory cycle. It does not rule out the possibility that bad cases might be quite infrequent; indeed, even an emulation with constant amortized slowdown is still possible. Our proof, however, yields something stronger than merely a bound on the (worst-case) slowdown. Given a particular emulation and a positive integer T, let $S(T)$ be the maximum number of network steps needed to emulate

T consecutive memory cycles. What we prove is that $S(T)/T = \Omega(\min\{\log m, \sqrt{n}\})$ if T is sufficiently large; i.e., the amortized slowdown is no better than our lower bound on the worst-case slowdown.

3. A GAME MODEL OF THE EMULATION

In the interest of precision and simplicity, we model the emulation by a game played between player A (the "adversary" or "algorithm") and player E (the "emulation"). Although some parallels to the emulation are drawn in the description of the game, we leave to the reader the complete translation between the two contexts. The description of the game should be taken as the final authority concerning what is and what is not allowed in the emulation. For instance, the use of separate tokens for each variable rules out the possibility of combining values of many variables into a single data item that can be routed at unit cost. Whereas it is not clear how to achieve such a combination, schemes such as that of Rabin (1989) show that this restriction may not be vacuous.

The game is played by altering configurations of *tokens* placed at the vertices of an undirected graph according to rules detailed below. Each token is labeled by an integer, and for each label x, there are two kinds of tokens labeled x, *copy tokens* and *query tokens*. A token labeled x will be called a token *for* x. The vertices of the graph correspond to the processors of the emulating network, each edge $\{u, v\}$ corresponds to a (bidirectional) communication link between the processors represented by u and v, and each label x represents a PRAM variable. In the remaining description, we will identify each vertex with the corresponding processor and each label with the corresponding PRAM variable. A copy token for a variable x at a vertex v represents the fact that v "knows" the current value of x. A query token for x at a vertex v indicates that the current value of x has been requested at v. In the remainder of the paper, let $d \geq 2$ be a fixed integer.

Let $n, m \in \mathbb{N} = \{1, 2, ...\}$. The *$(n, m)$-game* between players A and E is played as follows: In his first move, player E chooses an undirected graph $G = (V, E_G)$ with n vertices and maximum degree d, and a copy token for each element of $X = \{1, ..., m\}$ is placed at each vertex $v \in V$. X represents the set of variables, and the initial configuration is designed to maximize the advantage of player E, while eliminating a technical difficulty related to unitialized variables. The rest of the game is an infinite sequence of *rounds*, where each round consists of one move by player A followed by zero or more moves by player E. Player A chooses freely between *write moves* and *read moves*. A write move has the form *Write*(Y), where Y is a subset of X with $|Y| \leq n$, and consists in the following: For each $x \in Y$, first all tokens for x are removed from the graph, after which player A places a single copy token for x at some vertex $v \in V$. Within a single write move, each vertex $v \in V$ may receive at most one copy token. Y represents the set of PRAM variables updated in a given step, old tokens for variables in Y are removed because they represent copies that are invalidated by the update, and a copy token for x is placed at the single processor that initially knows the new value of x, for each $x \in Y$.

A read move has the form *Read*(Y), where Y as before is a subset of X with $|Y| \leq n$, and consists in the following: For each $x \in Y$, player A places a query token for x at some vertex $v \in V$, with no single vertex receiving more than one query token within a single read move.

Each move by player E consists in the following: For each edge $e \in E_G$, player E chooses one variable $x \in X$. If a copy token for x is present at exactly one endpoint of e, a new copy token for x is created and placed at the other endpoint of e. This models the communication of the value of x from one processor to a neighboring processor. Player E must continue to execute moves until each vertex that holds a query token also holds a copy token for the same variable, i.e., until all read requests have been satisfied. Informally, player A aims to maximize the number of moves executed by player E, while player E tries to minimize this quantity. For all $n, m, T \in \mathbb{N}$, let

$$S(n, m, T) = \min\{S \in \mathbb{N} \mid \text{there is a strategy for player } E \text{ that executes at most } S \text{ moves during the first } T \text{ rounds of any play of the } (n, m)\text{-game}\}.$$

It is easy to see that $S(n, m, T)$ is well-defined for all $n, m, T \in \mathbb{N}$. By the interpretation of the game, $S(n, m, T)$ bounds from below the minimum total slowdown over T successive memory cycles achievable by any deterministic emulation. Our aim in the next section is to prove a lower bound on $S(n, m, T)/T$ that holds for all sufficiently large values of T.

By modifying the rules of the game slightly, we can express the assumption of point-to-point communication. Suppose that in each write move *Write*(Y) and for all $x \in Y$, player A places not a single copy token, but an infinite supply of copy tokens for x at some vertex $v \in V$, with each vertex still receiving tokens for at most one variable. Suppose further that in each move by player E, a single copy token may be *moved* across each edge, as opposed to being *copied* from one endpoint of the edge to the other. This modified game models an emulation that operates according to the assumption of point-to-point communication. For $n, m, T \in \mathbb{N}$, let

$$S'(n, m, T) = \min\{S \in \mathbb{N} \mid \text{there is a strategy for player } E \text{ that executes at most } S \text{ moves during the first } T \text{ rounds of any play of the modified } (n, m)\text{-game}\}.$$

Again, $S'(n, m, T)$ is easily seen to be well-defined, and clearly $S(n, m, T) \leqslant S'(n, m, T)$ for all $n, m, T \in \mathbb{N}$. The earlier lower bounds mentioned in the introduction can now be expressed as

$$S'(n, m, T) = \Omega\left(\min\left\{\sqrt{n \log n}, \frac{\log n \log m}{\log\log m}, \frac{(\log(m/n))^2}{\log\log(m/n)}\right\}\right),$$

for all sufficiently large values of T.

4. THE PROOF

This section establishes the lower bound $S(n, m, T)/T = \Omega(\min\{\log m, \sqrt{n}\})$, valid for $T \geqslant 4m$. In fact, we exhibit an explicit strategy for player A that realizes the lower bound. Without loss of generality, we will assume that $m \geqslant 4n^4$, since otherwise the lower bound to the shown coincides with the trivial lower bound of $\Omega(\log n)$.

For a particular play of the (n, m)-game, let $G = (V, E_G)$ be the graph chosen by player E in his first move. For $x \in X = \{1, ..., m\}$ and $t \in \mathbb{N}$, denote by $\Gamma_x(t) \subseteq V$ the set of vertices that hold a copy token for x at the beginning of round t and call $|\Gamma_x(t)|$ the *redundancy* of x at that time. The strategy for player A can now be described as follows: For $t = 1, 3, 5, ...$, the move by player A in round t is of the form *Read*(Y), where Y is an n-element subset of X that minimizes $|\bigcup_{x \in Y} \Gamma_x(t)|$ over all such subsets; for $t = 2, 4, 6, ...$, the move by player A in round t is of the form *Write*(Y), where Y is an n-element subset of X that maximizes $\sum_{x \in Y} |\Gamma_x(t)|$ over all such subsets.

In order to analyze the resulting play, let us introduce some more notation. For $t \in \mathbb{N}$, let $s(t)$ be the number of moves by player E in round t, and call $R(t) = \sum_{x \in X} |\Gamma_x(t)|$ and $r(t) = R(t)/m$ the *total redundancy* and the *average redundancy*, respectively, at the beginning of round t. Assume that T is even, let $\hat{r} = (1/T) \sum_{t=1}^{T} r(t)$ be the average redundancy over the first T rounds and let $S = \sum_{t=1}^{T} s(t)$ be the total number of moves executed by player E over the first T rounds.

LEMMA 1. *If $T \geqslant 4m$ and $\hat{r} \geqslant 2$, then $S \geqslant T\hat{r}/(8d)$.*

Proof. Fix $t \in \{2, 4, ...\}$. Since $r(t)$ is the average redundancy at the beginning of round t, there clearly is an n-element subset Y of X such that $\sum_{x \in Y} |\Gamma_x(t)| \geqslant nr(t)$. The move *Write*$(Y)$ by player A reduces the redundancy of each element of Y to 1. Hence the move by player A in round t reduces the total redundancy R by at least $n(r(t) - 1)$. On the other hand, a single move by player E increases R only by the number of new tokens created, i.e., by at most dn. Since R has the value mn initially and remains nonnegative,

$$S \cdot dn + mn \geqslant \sum_{i=1}^{T/2} n(r(2i) - 1).$$

Since only write moves by player A decrease redundancies, $r(2i) \geqslant r(2i - 1)$ for $i = 1, 2, ...$, and hence

$$\sum_{i=1}^{T/2} r(2i) \geqslant \frac{1}{2} \sum_{t=1}^{T} r(t) = \frac{T\hat{r}}{2}.$$

Using the assumptions $T \geqslant 4m$ and $\hat{r} \geqslant 2$, we then obtain

$$S \geqslant \frac{1}{d}\left(\frac{T(\hat{r} - 1)}{2} - m\right) \geqslant \frac{T\hat{r}}{8d}. \quad \blacksquare$$

LEMMA 2. *For $t = 1, 3, ...$, if $r(t) \leqslant (1/8) \log_d m$, then $s(t) \geqslant n/(4\, dr(t)) - 1$.*

Proof. Note first that $r(t) \leqslant (1/8) \log_d m$ implies $d^{4r(t)} \leqslant \sqrt{m}$ and hence

$$\frac{m}{2n\, d^{4r(t)}} \geqslant \frac{\sqrt{m}}{2n} \geqslant n$$

(recall the assumption $m \geqslant 4n^4$).

Now choose $Z \subseteq X$ with $|Z| \geqslant m/2$ such that $|\Gamma_x(t)| \leqslant 2r(t)$ for all $x \in Z$. By the rules of the (n, m)-game, the subgraph G_x of G induced by $\Gamma_x(t)$ is connected, for arbitrary $x \in X$. Hence for each $x \in Z$ there is a path in G of length at most $2(2r(t) - 1)$ that contains all vertices in $\Gamma_x(t)$ (take any Euler tour of a spanning tree of G_x). But there are at most $n \cdot d^{4r(t)}$ such paths in G, i.e., at least

$$\frac{|Z|}{n\, d^{4r(t)}} \geqslant \frac{m}{2n\, d^{4r(t)}} \geqslant n$$

elements of Z share the same path, and this path contains at most $4r(t)$ vertices. If the move by player A in round t is *Read*(Y), it is therefore clear that $|\Gamma_Y| \leqslant 4r(t)$, where $\Gamma_Y = \bigcup_{x \in Y} \Gamma_x(t)$. Since the moves by player E in round t must cause at least $n - |\Gamma_Y|$ tokens for variables in Y to be transmitted over edges joining Γ_Y to the rest of the graph, it follows that

$$s(t) \geqslant \frac{n - |\Gamma_Y|}{d\,|\Gamma_Y|} \geqslant \frac{n}{4dr(t)} - 1. \quad \blacksquare$$

MAIN THEOREM. *If $T \geqslant 4m$, then $S(n, m, T)/T = \Omega(\min\{\log m, \sqrt{n}\})$.*

Proof. The set $I = \{t \in \mathbb{N} \mid t \leqslant T \text{ and } r(t) \leqslant 2\hat{r}\}$ is of size at least $T/2$. Furthermore, since $r(2i) \geqslant r(2i - 1)$ for $i = 1, 2, ...$, I contains at least $T/4$ odd integers. But then either $\hat{r} > (1/16) \log_d m$, in which case the claim of the theorem follows from Lemma 1, or Lemma 2 implies that

$$S \geqslant \frac{T}{4}\left(\frac{n}{8d\hat{r}} - 1\right).$$

If $\hat{r} < 2$, the claim follows. Otherwise, combine the inequality with Lemma 1 to obtain

$$\frac{S}{T} \geqslant \max\left\{\frac{n}{32d\hat{r}} - 1, \frac{\hat{r}}{8d}\right\} = \Omega(\sqrt{n}). \quad \blacksquare$$

5. CONCLUSION

There seems to be a substantial amount of slack in our proof. We expect that the lower bound can be significantly strengthened using more sophisticated combinatorial techniques.

By the main theorem, $\sup_T S(n, m, T)/T = \Omega(\sqrt{n})$ for m sufficiently large relative to n, more precisely, for $m = 2^{\Omega(\sqrt{n})}$. Alt *et al.* (1987) describe an emulation on a mesh-of -trees network with slowdown $O(\sqrt{n})$ for arbitrary m (p. 829), which is therefore optimal for this range of m. The only emulations previously known to be optimal without the restriction of point-to-point communication match the trivial lower bound of $\Omega(\log n)$ and work only for $m = O(n \log n)$ (see Peleg and Upfal, 1989). Still for m sufficiently large relative to n, Alt *et al.* (1987) establish upper and lower bounds on the slowdown of emulations obeying the restriction of point-to-point communication of $O(\sqrt{n} \log n)$ and $\Omega(\sqrt{n \log n})$, respectively. In this case, the insistence on point-to-point communication hence worsens the obtainable slowdown by a factor of between $\Theta(\sqrt{\log n})$ and $\Theta(\log n)$. It is unknown, and an interesting open problem, whether a similar phenomenon occurs for smaller values of m.

ACKNOWLEDGMENT

The author is grateful to the referees for several suggestions that improved the presentation.

Received June 24, 1990; final manuscript received December 15, 1993

REFERENCES

Alt, H., Hagerup, T., Mehlhorn, K., and Preparata, F. P. (1987), Deterministic simulation of idealized parallel computers on more realistic ones, *SIAM J. Comput.* **16**, 808–835.

Herley, K. T., and Bilardi, G. (1994), Deterministic simulations of PRAMs on bounded degree networks, *SIAM J. Comput.* **23**, 276–292.

Karlin, A. R., and Upfal, E. (1988), Parallel hashing: An efficient implementation of shared memory, *J. Assoc. Comput. Mach.* **35**, 876–892.

Peleg, D., and Upfal, E. (1989), The token distribution problem, *SIAM J. Comput.* **18**, 229–243.

Rabin, M. O. (1989), Efficient dispersal of information for security, load balancing, and fault tolerance, *J. Assoc. Comput. Mach.* **36**, 335–348.

Ranade, A. G. (1991), How to emulate shared memory, *J. Comput. System Sci.* **42**, 307–326.