Parallel Hashing: An Efficient Implementation of Shared Memory

ANNA R. KARLIN

Stanford University, Stanford, California

AND

ELI UPFAL

IBM Almaden Research Center, Almaden, California

Abstract. A central issue in the theory of parallel computation is the gap between the ideal models that utilize shared memory and the feasible models that consist of a bounded-degree network of processors sharing no common memory. This problem has been widely studied. Here a tight bound for the probabilistic complexity of this problem is established.

The solution in this paper is based on a probabilistic scheme for implementing shared memory on a bounded-degree network of processors. This scheme, which we term *parallel hashing*, enables n processors to store and retrieve an arbitrary set of n data items in $O(\log n)$ parallel steps. The items' locations are specified by a function chosen randomly from a small class of universal hash functions. A hash function in this class has a small description and can therefore be efficiently distributed among the processors. A deterministic lower bound for the point-to-point communication model is also presented.

Categories and Subject Descriptors: C.1.2 **[Processor Architectures]**: Multiple Data Stream Architecture—*parallel processors*; D.4.2 **[Operating Systems]**: Storage Management—*distributed memories*; F.1.2 **[Computation by Abstract Devices]**: Modes of Computation—*parallelism*; *probabilistic computation*; *relations among modes*; F.2.2 **[Analysis of Algorithms and Problems]**: Nonnumerical Algorithms and Problems—*routing and layout*; *sorting and searching*

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Feasible model, parallel algorithm, probabilistic computation, randomized algorithm, routing, simulation between models, theoretical model

1. *Introduction*

1.1 MOTIVATION. Information exchange between processors is essential for any efficient parallel computation. Typically, a logical step in any synchronous parallel algorithm starts when processors specify data items they wish to access. Execution of the logical step ends only after all requests are satisfied.

A preliminary version of this paper appeared in the *Proceedings of the 18th Annual ACM Symposium on Theory of Computing* (Berkeley, Calif., May 28–30). ACM, New York, 1986, pp. 160–168.

The research of Anna R. Karlin was supported by Office of Naval Research grant 00014-85-C-0731 and by an IBM doctoral fellowship.

Authors' addresses: A. R. Karlin, Computer Science Department, Princeton University, Princeton, NJ 08544; E. Upfal, IBM Almaden Research Center, Almaden, CA 95120.

Most of the research in the theory of parallel computation assumes models similar to parallel random access machines (PRAM) in which each such logical step is executed in $O(1)$ steps. These models idealize communication and allow us to focus on the computation. Indeed, they are very convenient for programming, as well as for the design and analysis of parallel algorithms.

Unfortunately, no realization of such models seems feasible in the foreseeable technologies. The only current feasible models consist of a bounded degree network of processors [11]. Information in such models is exchanged only by messages between directly connected processors. Thus, in practice, processors in a large network can communicate directly only with a few neighbors, and direct access to a shared memory is excluded.

To bridge this fundamental gap between the desirable and the feasible in parallel computation one has to study the complexity of simulating a PRAM by a bounded-degree network of processors. A tight bound for such a simulation gives practical significance to the vast number of results that have been obtained for the PRAM and related models. Moreover, an efficient and simple-to-program simulation proves the feasibility of the following attractive scheme for using general purpose parallel computers. Users may write their programs for an abstract and convenient model of parallel computation such as the PRAM, and an automatic device will then translate these programs to some feasible parallel machine commands, just as today's compilers translate high-level languages to sequential machine commands, without significant loss of efficiency.

Of special interest are simulations between two models with the same number of processors, since an equal number of resources in both models enables us to study their relative computation power. In this work we concentrate on simulations of this type.

Formally, an (n, m)-PRAM is a set of n processors (RAMS) sharing a random access common memory of size m. At each step, each processor can either perform some internal computation or access one arbitrary variable in the common memory. Our goal is to execute a PRAM program on a bounded-degree network of n processors. In this model, at each step each processor can either perform some internal computation or it can send one message to one direct neighbor in the network. Since there is no shared memory, all the information is stored in the processors' local memory. We assume that the computation power of individual processors in both models is equal, and that an atomic message in the bounded-degree model can carry a single variable address and a single variable value.

This problem has been addressed in numerous papers. First heuristics were given tailored to particular families of programs (see [8] and the references there), and recently the research has focused on algorithms that can simulate an arbitrary PRAM program. In a sequence of papers [1, 3, 9, 13, 15, 17], the probabilistic and deterministic upper bounds for simulating T arbitrary PRAM steps were reduced from $O(Tn)$ to $O(T(\log n)^2)$ (with probability tending to 1) in the probabilistic case [13] and to $O(T(\log n)^2)$ in the deterministic case [1] where the number of shared variables is polynomial in n.

1.2 New Results. Our main result is a tight bound for the probabilistic complexity of this problem when m is polynomial in n.

Theorem 1.1. *The probabilistic complexity of simulating an arbitrary T step (n, m)-PRAM program, m polynomial in n, on a bounded-degree network of n processors is* $\Theta(T \log n)$.

The $\Omega(T \log n)$ probabilistic lower bound is easy and is presented in Section 3. The probabilistic upper bound is a corollary of the following theorem.

THEOREM 1.2. *Any sequence of $T(n, m)$-PRAM steps can be simulated in $O(T \log m)$ steps on a bounded-degree network with probability tending to 1 as n and/or T go to ∞.*

The simulation algorithm we present is a Las Vegas algorithm. It always executes the simulation correctly, and it always terminates. Only its run-time is probabilistically analyzed over all possible random choices made during its execution. No assumptions are made about the input PRAM program.

All previous algorithms perform the simulation (explicitly or implicitly) in two phases. First, the use of shared memory is eliminated, so the PRAM program can be simulated on a model consisting of a complete network of processors sharing no central memory (sometimes referred to as an MPC). Then a parallel routing algorithm is used to simulate every MPC step on the bounded-degree network.

Any (deterministic or probabilistic) routing algorithm that simulates an arbitrary MPC step requires $\Omega(\log n)$ steps. Thus it is very unlikely that using similar simulation techniques one can reach the $O(T \log n)$ lower bound. To overcome this problem we propose a probabilistic method that handles both tasks simultaneously.

Our PRAM simulation is based on a scheme we call *parallel hashing*. This scheme combines the use of Universal Hash Functions [4] for distributing the variables among the processors and a probabilistic routing algorithm for executing memory requests. Simulation of a PRAM step involves communication between each processor that wishes to access a variable and that processor storing the variable. The analysis of the algorithm has to show that when the variables are distributed according to a function randomly chosen from a small set $\mathscr{H}$ of hash functions, the communication required for simulating an arbitrary PRAM instruction is executed in $O(\log n)$ steps with high probability.

Previous analyses of routing algorithms have assumed that either the origins and destinations of messages are evenly distributed among the processors, or they are randomly chosen, independent of one another.

The routing task in our case is different. Typically $m \gg n$, and a single instruction might involve several variables stored at the same processor. Furthermore, neither the origins nor the destinations of the messages are probabilistically independent of one another since they were fixed by the same hash function. Therefore, a different analysis is required here based on the special properties of appropriately chosen hash functions.

Typical results in the theory of hash functions deal with their average performance, primarily with the average number of keys sent to one location. These results are not sufficient for our purpose. Instead, we need to prove a stochastic upper bound on the distribution of the number of variables being requested, out of arbitrary set of n, that are stored in one location. We then extend this result, using a shrinking lemma, to bound the number of requested variables stored in certain portions of the network. Using these tools we are able to prove that if the variables are distributed by a function randomly chosen from a relatively small set of hash functions $\mathscr{H}$, where $|\mathscr{H}| \leq m^{O(\log m)}$, then with high probability, the simulation of an arbitrary PRAM instruction takes $O(\log m)$ steps.

In Sections 3 and 4, we present two lower bounds for simulating a PRAM on a bounded degree network of processors. Theorem 1.1 gives a $\Theta(\log n)$ tight

bound for probabilistic simulations. The probabilistic lower bound is presented in Section 3. The obvious remaining question is whether such a run time can be achieved by a deterministic computation. The last section presents a lower bound that gives some evidence to the contrary. In [15] it was proved that under very natural assumptions any deterministic on-line simulation of a T-step PRAM program on a complete network of processors (MPC) requires $\Omega(T \log n/\log\log n)$ steps. We extend this technique to prove that under certain restrictions on the communication between processors any deterministic simulation of a T-step PRAM program on a bounded-degree network of processors requires $\Omega(T(\log n)^2/\log\log n)$ steps (for T superlinear). This lower bound almost matches the known upper bound that assumes the same restriction on communication. A similar lower bound was proven independently in [1].

2. *The Probabilistic Upper Bound*

2.1 AN INFORMAL DESCRIPTION OF THE ALGORITHM. Without loss of generality we can assume that we simulate an Exclusive-Read-Exclusive-Write-PRAM (EREW-PRAM) program (no variable is simultaneously accessed by two processors). An easy reduction [12, p. 239][1] enables us to simulate any other PRAM model, adding $O(\log n)$ steps to the simulation time of each instruction.

Throughout the execution of the algorithm the m shared variables are distributed according to a randomly chosen hash function h from the following class of universal hash functions:

$$\mathcal{H} = \left\{ h \mid h(x) = \left(\left(\sum_{0 \le i < \zeta} a_i x^i \right) \bmod p \right) \bmod n, \right.$$
$$\left. \text{for any } a_i \in [0 \cdots p-1],\ a_0 \in [1 \cdots p-1] \right\},$$

where p is a prime, $p \ge m$, and $\zeta = a \log m$ for some constant a fixed in the proof. The hash function h is known to all processors. When a processor wishes to read or write a variable v, it computes $h(v)$ in order to determine which processor it needs to communicate with. This computation adds $O(\log m)$ steps to the simulation of each PRAM instruction.

A single PRAM instruction is executed by sending messages from each processor to the processor storing the variable it wishes to access and back in case of a read instruction. The communication protocol for doing this will be shown to terminate in $O(\log m)$ steps with high probability. If, however, the communication has not been completed in the allotted time, the current attempt at simulating the instruction is aborted.[2] A new random hash function $h' \in \mathcal{H}$ is then chosen and distributed to all processors, say by processor 1, and all the variables are moved to their new locations as specified by h'. The simulation process then resumes, with a new attempt at executing the failed instruction. We prove that the combination of the class of hash functions and the communication algorithm we use guarantees that the costly operation of rehashing is executed only rarely.

[1] The key idea in the reduction is for all the processors requesting the same variable to elect a leader responsible for accessing that variable. This is done by sorting the requests. The leader then distributes the result using a fast data distribution algorithm.

[2] Since all communication is synchronous and processors know when each instruction begins, failure can be detected by having each processor stop after the alloted time and send a message to processor 1 indicating if they have not yet received the item requested.

The communication protocol we use is similar to the algorithm presented in [14] for routing a permutation. Only the analysis is different. The algorithm is best described on the butterfly network. For simplicity, we assume that the number of processors is $n = r2^r$. Separate the processors's numbers, in their binary representation into two parts. The r right bits are called the *address*, and the other part is called the *prefix*. The connections in the butterfly network are as follows. There are two edges directed out of the processor with the address $b_0 \cdots b_\alpha \cdots b_{r-1}$ and the prefix α. The other endpoints of these two edges are the processors with the addresses $b_0 \cdots b_\alpha \cdots b_{r-1}$ and $b_0 \cdots \overline{b_\alpha} \cdots b_{r-1}$ and prefixes $(\alpha + 1) \bmod r$. Thus each group of processors having the same prefix α forms one stage in the network and is connected only to processors with prefixes $(\alpha + 1) \bmod r$. The last stage is connected to the first one.

The process of routing a message to its destination is performed in two phases. First, the message is sent to a random location in the network, then it is routed to its destination by the shortest path. To send a message to a random location, the processor storing that message picks a random number l in the range $r, \ldots, 2r - 1$, and the message takes l random transitions, where in a random transition the message chooses randomly a line among the lines leaving its current location.

Each transition a message takes will have a *priority* associated with it. The priority is the number of transitions the message has already performed. If several messages are queued up at a processor, that message waiting for a transition with the least priority number (thus a message that has travelled less) is the first to leave. Thus, the priority numbers are used to speed up slower messages at the expense of faster ones.

2.2 Analysis of the Algorithm. In this section, we shall show that the algorithm described in the previous section simulates a T step PRAM, program in $O(T \log m)$ steps on a butterfly network with high probability. We begin by describing the intuition behind the proof.

At the highest level, we shall show that each PRAM instruction is simulated in $O(\log m)$ steps. In particular, the probability that the simulation of a particular instruction requires more than $O(\log m)$ steps will be shown to be so small that rehashing operations are rare. Thus, the time required to execute all rehashing operations that ever occur is subsumed by the time required to simulate all the PRAM instructions.

We bound the failure probability of the single-step simulation, wherein a set S (of cardinality n) variables are read or written to, by considering all possible paths in the network with which variables may ever coincide as they travel to their destinations. We can show that the probability that too many variables in S "touch" any of these paths is very small. The relatively small congestion this implies yields the overall time bound on the simulation of this step.

The overlap probability for a particular path is split into two disjoint cases, one in which we show that the probability that a variable hits the path for the first time is small, and the other in which we show that the probability that a variable stays in or returns to the path (after it has hit it for the first time) is small. The latter computation is simple. For the former, the probability that too many variables hit the path for the first time, we again split the event into two cases. This time, we are able to show that either many variables were mapped by the random hash function to a relatively small set of specific locations, or that few variables were

mapped to these locations, but those that were took very specific (are unlikely) routes as they took their random walk through the network. Both of these events can be shown to have very low probability.

THEOREM 2.1. *Any given PRAM program with n processors, m shared variables and T instructions, can be simulated on an n node butterfly network in $O(T \log m)$ steps with probability tending to 1 as n and/or T go to ∞ (m is assumed greater than n).*

THEOREM 2.2. *For any constant $c > 2 + \epsilon$ there is a constant c' such that the simulation algorithm above executes an arbitrary single PRAM instruction on a butterfly network in less than $c' \log m$ steps with probability greater than $1 - (1/m^c)$.*

For the analysis we need some facts from probability theory.

Fact 1 [2, 5]. For every $n > 0$, $0 < p < 1$ and $0 < \beta < 1$,

$$\sum_{0 \le k \le \lfloor (1-\beta)np \rfloor} \binom{n}{k} p^k (1-p)^{n-k} \le \exp\left(-\frac{\beta^2 np}{2}\right).$$

Denote by $B(m, N, p)$ the probability of at least m successful trials in a sequence of N independent Bernoulli trials each with probability p.

Fact 2 [7]. If T is the number of successes in N independent Poisson trials with probability $p_1, \ldots, p_N$, then if $\sum_i p_i = N_p$ and $m \ge Np + 1$ is an integer, then

$$\Pr(T \ge m) \le B(m, N, p).$$

Fact 3 [5]. If $m \ge Np$ is an integer, then

$$B(m, N, p) \le \left(\frac{Np}{m}\right)^m \exp(m - Np).$$

Assuming Theorem 2.2, we prove Theorem 2.1.

PROOF OF THEOREM 2.1. We distinguish two cases.
If $T \le m^{c-1}$ then

$$\begin{aligned} &\Pr(T \text{ steps take more than } c'T \log m) \\ &\quad \le \sum_{1 \le i \le T} \Pr(\text{single step takes more than } c' \log m \text{ steps}). \\ &\quad \le \frac{T}{m^c} \le \frac{1}{m} \end{aligned}$$

by Theorem 2.2.

The other case is when $T > m^{c-1}$. We show that with high probability no more than $8T/m^{c-1}$ rehashing operations occur. Since each rehashing operation takes $O(m \log n)$ time (in the worst case each data item is transmitted sequentially) and $c > 2 + \epsilon$, all the rehashing operations take $O(T \log n)$ time. Hence, as each attempted PRAM simulation step runs for at most $c' \log m$ steps, the total simulation time is $O(T \log m)$.

We say that a PRAM instruction is successfully simulated if it terminates in $c' \log m$ steps with no rehashing operation necessary. We group sets of consecutive instructions whose simulations are successful into *subsequences*. The probability

that more than

$$4\left(\frac{T}{m^{c-1}/2}\right)$$

rehashing operations occurs is bounded by the probability that more than half of the subsequences run for less than or equal to $m^{c-1}/4$ steps. This follows from the fact that if more than half of the subsequences run for more than $m^{c-1}/4$ steps, then $8T/m^{c-1}$ rehashings would suffice to simulate all T instructions.

Thus we need to bound the probability that more than half of the subsequences run for at most $m^{c-1}/4$ steps. In order to do this, we consider each subsequence as a Bernoulli trial that succeeds if it terminates after at most $m^{c-1}/4$ steps. By Theorem 2.2, the probability of a success is at most $1/4m$. Hence, the probability that more than $8T/m^{c-1}$ rehashing operations occurs is bounded by the probability of more than half the trials succeeding in at least $8T/m^{c-1}$ Bernoulli trials with probability at most $1/4m$ of success. By Chernoff's bound, this latter probability is at most

$$\begin{aligned}
&\sum_{N>(8T/m^{c-1})} \left(\frac{N/4m}{N/2}\right)^{N/2} \exp\left(\frac{N}{2} - \frac{N}{4m}\right) \\
&\quad \le \sum_{N>(8T/m^{c-1})} (2m)^{-N/2} \exp\left(\frac{N}{2}\right) \\
&\quad \le \sum_{N>(8T/m^{c-1})} \left(\frac{m}{2}\right)^{-N/2} \le \left(\frac{m}{2}\right)^{-4T/m^{c-1}}.
\end{aligned}$$

□

PROOF OF THEOREM 2.2. Simulating a single PRAM instruction involves routing four types of communication requests. First, up to n messages are sent from distinct locations to randomly chosen locations. Then, these messages are sent from their random locations to destinations defined by a randomly chosen hash function. On their way back, messages are first sent from locations defined by the hash function to random locations and then from random locations to distinct addresses.

The communication algorithm was analyzed in [14] for the first and the last types of communication requests. It was proved there that the algorithm executes each of these requests in $O(\log n)$ steps with high probability. Thus, we only have to analyze the communication algorithm for the communication requests that depend on the hash function. Intuitively, we need to show that the level of randomization given by the hash function is sufficient for the success of the algorithm with high probability. Since the analysis of the two cases is similar, we consider here only the process of transmission of up to n messages from locations specified by the hash function to randomly chosen locations.

For the purposes of the proof we consider the following modified algorithm. No processor ever transmits a message of priority i before it has transmitted all messages of priority less than i that it will ever transmit. Such an execution clearly takes at least as long as an execution of the original algorithm.

With respect to one execution of the modified algorithm, as in [14] we define the notion of a *critical delay sequence*. First, a *delay sequence* is a sequence of processors $y_0, \ldots, y_{2r-1}$, such that for any $i < 2r - 1$ either $y_i = y_{i+1}$ or y_i is one of the two processors with edges to y_{i+1}. A delay sequence is *critical* if y_{2r-1} was one of the last processors to transmit a message in the execution of the algorithm, and for each $0 \le i < 2r - 1$, y_i was one of the last processors to transmit messages with priority i from among the two processors with edges to y_{i+1} and y_{i+1} itself.

Intuitively, a delay sequence represents a path in the network with which messages may coincide as they travel to their destinations. We are interested in measuring the congestion along these paths, that is, how many messages overlap with it. The critical delay sequence will be that path which is the bottleneck. In other words, the delay on the critical delay sequence will determine the overall running time of one simulation step. For a given critical delay sequence D, define two random variables:

f_i^D is the number of messages with priority i leaving y_i, and

t_i^D is the time when all messages of priority j leaving y_i, for $j \leq i$ have been transmitted.

If $t_0^D = 0$, then

$$t \leq \sum_{1 \leq i \leq 2r-1} (t_i^D - t_{i-1}^D) \leq \sum_{0 \leq i \leq 2r-1} f_i^D.$$

Hence, defining F^D to be $\Sigma_{0 \leq i \leq 2r-1} f_i^D$, we are interested in bounding the probability of an execution that has a critical delay sequence D with $F^D \geq c' \log m$.

Since there are no more than $r2^r 3^{2r} \leq m^5$ possible delay sequences, it is enough to show that for a given delay sequence $\Pr\{\sum_i f_i^D \geq c' \log m\} < m^{-(5+c)}$. The remainder of the proof consists of showing that for an arbitrary chosen delay sequence D consisting of processors $y_0, \ldots, y_{2r-1}$, this bound holds.

In order to perform this analysis, we present each f_i^D as the sum of two random variables g_i^D and h_i^D. Let s_i^D be the set of messages leaving y_i with priority i.

g_i^D is the number of messages in s_i^D not appearing in $\bigcup_{0 \leq j \leq i-1} s_j^D$, and
h_i^D is $f_i^D - g_i^D$.

Let $G^D = \sum_i g_i^D$ and $H^D = \sum_i h_i^D$. Then $F^D = G^D + H^D$. Owing to the structure of the graph, in $2r - 1$ transitions a message can reenter the delay sequence only once. Once a message touches the delay sequence, it continues to overlap with it for a number of steps that is distributed geometrically with parameter $\frac{1}{2}$. But indeed H^D counts the number of steps each of the distinct messages stays in the delay sequence (after the first). Hence, H^D is bounded by a negative binomial distribution with parameters $2G^D$ and $\frac{1}{2}$ and with high probability H^D is bounded by $8G^D$. This is formally presented in Lemma 2.5 below.

It remains to bound G^D, the number of distinct messages involved in a critical delay sequence D. Let S be the set of variables being requested by the PRAM instruction ($|S| = n$) and let X_i^D be the number of variables in S assigned by the hash function to the 2^i processors (2^r processors for $i \geq r$) that can contribute messages of priority i to y_i's queue. Since each of the X_i^D messages has probability 2^{-i} (2^{-r} for $i \geq r$) of leaving y_i with priority i, $\mathrm{E}(G^D) = \sum_{0 \leq i < r} X_i^D/2^i + \sum_{r \leq i < 2r-1} X_i^D/2^r$.

Let $b > c + 11$ be a constant. We say that *some* X_i^D is big if any of the following events occurs:

$$X_i^D > \begin{cases} b \log m & \text{for } 0 \leq i < \log\log m; \\ b2^i & \text{for } \log\log m \leq i < r; \\ b2^r & \text{for } r \leq i < 2r. \end{cases}$$

If none of these events occur, we say that *all* X_i^D*'s are small.*

If all X_i^D's are small, then $\mathrm{E}(G) = \sum_i X_i^D 2^i + \sum_{r \leq i < 2r-1} X_i^D/2r \leq 4b \log m$. We now extend this to a high probability argument.

LEMMA 2.1. *For any constants $b > 2$ and $b' > 26b$*

$$Pr\left(\sum_{0\le i<2r} g_i^D \ge b' \log m \mid \text{all } X_i^D\text{'s are small}\right) \le \frac{1}{m^b}.$$

PROOF. First, note that $\Pr(\sum_{0\le i<2r} g_i^D \ge b'\log m \mid \text{all } X_i^D \text{ are small})$ is maximized when the X_i^D random variables achieve their upper limit because of the straightforward dependence between these random variables. Hence, we perform the computation assuming that

$$X_i^D = \begin{cases} b\log m & 0 \le i < \log m; \\ b2^i & \log m \le i < r; \\ b2^r & r \le i < 2r. \end{cases} \qquad (*)$$

Let $Y_1, \ldots, Y_n$ be a set of indicator random variables defined with respect to the delay sequence D such that Y_j is 1 if for some i, packet j leaves y_i with priority i, and is 0 otherwise. (Each of these packets is an element of S.) Once $(*)$ is given, the n packets in S have been placed into the local memories in some fashion. In particular, any two packets j and k ($j, k \in S$) now have their starting point fixed, and the stochastic process from this point on is independent for different packets. (The routing process operates on packets independently.) Hence, $Y_j \mid (*)$ is independent of $Y_k \mid (*)$ for all pairs of packets $1 \le j < k \le n$.

We know that $\{\sum_{0\le i<2r} g_i^D \mid (*)\} = \sum_{1\le j\le n} Y_j \mid (*)$. Letting $p_j = \Pr(Y_j = 1 \mid (*))$, we have

$$\mathrm{E}\left(\sum_{0\le i<2r} g_i^D \,\middle|\, (*)\right) = \sum_{1\le j\le n} p_j.$$

But we also know that

$$\mathrm{E}\left(\sum_{0\le i<2r} g_i^D \,\middle|\, (*)\right) = \sum_{0\le i<r} \left(\frac{X_i^D}{2^i} \,\middle|\, (*)\right) + \sum_{r\le i<2r} \left(\frac{X_i^D}{2^r} \,\middle|\, (*)\right) \le 4b\log m.$$

Hence $\sum_{1\le j\le n} p_j \le 4b\log m$. Applying the Hoeffding theorem [7], we get that for $a > 4b\log m$, $\Pr(\sum_{0\le i<2r} g_i^D \ge a \mid (*)) \le B(a, n, 4b\log m/n)$. Thus, $\Pr(\sum_{0\le i<2r} g_i^D \ge b'\log m \mid (*)) \le B(b'\log m, n, 4b\log m/n)$. Applying fact 3 to this equation, we get

$$\Pr\left(\sum_{0\le i<2r} g_i^D \ge b'\log m\right) \le \left(\frac{4b\log m}{b'\log m}\right)^{b'\log m} \exp(b'\log m - 4b\log m)$$
$$\le \left(\frac{13b}{b'}\right)^{b'\log m} \le m^{-b}. \qquad \square$$

Thus, the crux of the proof is to show that with high probability all the X_i^D's for a delay sequence D are indeed small. To prove this, we need good stochastic bounds on the number of elements in S assigned by the hash function to processors' sets of size 2^i, $i = 0, \ldots, 2r-1$, that contribute messages of priority i to y_i.

We first prove a simple *shrinking lemma* that reduces the problem of bounding the number of elements mapped by the hash function to a relevant set of buckets to the problem of bounding the number of elements mapped by a related hash function to a single bucket in a smaller table. Then we establish the desired bounds.

Let Z_i^S be the maximum number of data items in the set S of n items that are mapped to any one bucket, where the number of buckets is $n/2^i$, $0 \le i < r$ ($n/2^r$ for $r \le i < 2r$) and the mapping is performed by choosing a random function

in the class

$$\mathscr{H}^{(i)} = \begin{cases} \left\{h \mid h(x) = \left(\left(\sum_{0 \le l < \zeta} a_l x^l\right) \bmod p\right) \bmod \frac{n}{2^i},\ a_l \in [0 \cdots p-1]\right\} & 0 \le i < r; \\ \left\{h \mid h(x) = \left(\left(\sum_{0 \le l < \zeta} a_l x^l\right) \bmod p\right) \bmod \frac{n}{2^r},\ a_l \in [0, \cdots p-1]\right\} & r \le i < 2r. \end{cases}$$

Recall that functions in $\mathscr{H}$ are polynomials of degree at most $\zeta - 1$ over $\mathscr{Z}_p$ taken modulo n. Functions in $\mathscr{H}^{(i)}$ are also polynomials of degree $\zeta - 1$ over $\mathscr{Z}_p$, but they are taken modulo $n/2^i$. Note that for this proof, we shall need to set ζ (the degree $+1$ of the polynomials in $\mathscr{H}$) to be $O(\log m)$. Nonetheless, we present the following two lemmas in the most general possible form.

LEMMA 2.2.

$$Pr(X_i^D \ge j) \le Pr(Z_i^S \ge j).$$

PROOF. We design a numbering scheme for the processors with the property that the set of processors in each of the residue classes $\bmod n/2^i$ is one of the sets of 2^i processors that contributes to X_i^D. This numbering works simultaneously for all D and for all i. Formally, a node with prefix α and address $b_0 \cdots b_{r-1}$ will be numbered $b_\alpha r \cdot 2^{r-1} + b_{\alpha+1} r \cdot 2^{r-2} + \cdots + b_{(\alpha+k) \bmod r}\, r \cdot 2^{r-k-1} + \cdots + b_{\alpha-2} r \cdot 2 + b_{\alpha-1} r + \alpha$.

Suppose the ith node in the delay sequence, y_i has prefix β and address $c_0 \cdots c_{r-1}$. Then the 2^i nodes that contribute to X_i^D all have prefix $(\beta - i) \bmod r$ and addresses that agree with $c_0 \cdots c_{r-1}$ on all bits except for $c_{(\beta-i) \bmod r} \cdots c_{(\beta-1) \bmod r}$. Applying the numbering of the previous paragraph to these nodes and noting that $n/2^i = r2^{r-i}$, we see that all the terms on which the numbering of these nodes differ are multiples of $n/2^i$. Therefore, they are all in the same residue class modulo $n/2^i$. □

LEMMA 2.3. *For* $j > max(\zeta, 2^i)$,

$$Pr(Z_i^S \ge j) \le n\left(\frac{2^i}{j - \zeta}\right)^\zeta.$$

PROOF. Let $N_i^l(S)$ be the set of functions $h \in \mathscr{H}^{(i)}$ mapping at least j elements in S to position l and let $p_i^l(S)$ be the probability that a random function $h \in \mathscr{H}^{(i)}$ maps at least j elements in S to position l, that is, $p_i^l(S) = |N_i^l(S)| / |\mathscr{H}^{(i)}|$. We shall count the number of $(2\zeta + 1)$ tuples $(x_1, \ldots, x_\zeta, y_1, \ldots, y_\zeta, h)$ such that

(1) For $1 \le i \le \zeta$, $x_i \in S$;
(2) $x_1, \ldots, x_\zeta$ are distinct;
(3) For $1 \le k \le \zeta$, y_k is congruent to $l \bmod n/2^i$ and $0 \le y_k < p$; and
(4) $h \in \mathscr{H}^{(i)}$ such that $h(x) \equiv P(x) \bmod n/2^i$ and $P(x_i) \equiv y_i \bmod p$, $1 \le i \le \zeta$ (P is a polynomial of degree $\zeta - 1$ over $\mathscr{Z}_p$).

Let A be the set of $(2\zeta + 1)$ tuples satisfying conditions (1)–(4).

We know that for each set of ζ elements $x_1, x_2, \ldots, x_\zeta \in S$ and $y_1, y_2, \ldots, y_\zeta$ with $y_k \equiv l \bmod n/2^i$, for $1 \le k \le \zeta$, there is at most one polynomial P of degree at most ζ over the field $\mathscr{Z}_p$ with $P(x_k) = y_k$, for k, $1 \le k \le \zeta$. There are $\binom{n}{\zeta}$ choices for $x_1, \ldots, x_\zeta$ and at most $\lceil p/n/2^i \rceil^\zeta$ choices for $y_1, \ldots, y_\zeta$. Hence $|A| \le \binom{n}{\zeta} \lceil p/(n/2^i) \rceil^\zeta$. Each $h \in N_i^l(S)$, however, has a set of $j > \zeta$ points $x_1, \ldots, x_j$ and j

values $y_1, \ldots, y_j$ such that $y_k \equiv l \bmod n/2^i$ for $1 \le k \le \zeta$ and a polynomial P over $\mathcal{Z}_p$ with $P(x_k) \equiv y_k \bmod p$, for k, $1 \le k \le j$. Since to each function h there corresponds a polynomial P with $h(x) \equiv P(x) \bmod n/2^i$, each of the $\binom{j}{\zeta}$ different subsets of ζ (x_k, y_k) pairs uniquely determines the function P and hence h as well. But each of these choices is in A, and so $|A| \ge \binom{n}{\zeta}\lceil p/(n/2^i)\rceil^{\zeta}$. Hence, we must have

$$\binom{j}{\zeta} |N_i^l(S)| \le \binom{n}{\zeta}\left\lceil \frac{p}{n/2^i}\right\rceil^{\zeta}.$$

Since $|H^{(i)}| = p^{\zeta}$ and $p_i^l(S) = |N_i^l(S)|/|H^{(i)}|$, we have that for all S

$$p_i^l(S) \le \frac{\binom{n}{\zeta}}{\binom{j}{\zeta}}\left(\frac{2^i}{n}\right)^{\zeta} \le \left(\frac{2^i}{j-\zeta}\right)^{\zeta}.$$

As $\Pr(Z_i^S \ge j) \le \sum_{1\le i\le n/2^i} p_i^l(S)$, the lemma is proven. □

Combining Lemmas 2.2 and 2.3 and appropriately fixing ζ to be $O(\log m)$, we can now show that the probability that there exists a delay sequence for which some X_i^D is big is sufficiently small.

LEMMA 2.4. *For $\zeta = a \log m$, and $b > a + 1$, Pr(some X_i^D is big) $\le 1/m^{a\log(b-a)-2}$.*

PROOF

$$\begin{aligned}\Pr(\text{some } X_i^D \text{ is big}) &\le \Pr(\text{some } Z_i^S \text{ is big}) \qquad \text{by Lemma 2.2}\\ &\le \sum_{0\le i\le \log m} \Pr(Z_i^S \ge b\log m)\\ &\quad + \sum_{\log\log m < i \le r} \Pr(Z_i^S \ge b2^i) + \sum_{r<i<2r} \Pr(Z_i^S \ge b2^r)\end{aligned}$$

Applying Lemma 2.3, recalling that for $i \ge r$, Z_i^S counts the number of items in S mapping to one bucket, where the number of buckets is $n/2^r$, we obtain

$$\begin{aligned}\Pr(\text{some } X_i \text{ is big}) &\le \sum_{0\le i\le \log\log m} n\left(\frac{2^i}{b\log m - a\log m}\right)^{a\log m}\\ &\quad + \sum_{\log\log m < i \le r} n\left(\frac{2^i}{b2^i - a\log m}\right)^{a\log m}\\ &\quad + \sum_{r<i<2r} n\left(\frac{2^r}{b2^r - a\log m}\right)^{a\log m}\\ &\le \sum_{0\le i<2r} n\left(\frac{1}{m}\right)^{a\log(b-a)}\\ &\quad \text{(since } i > \log\log m \text{ in the second and third summations)}\\ &\le \frac{1}{m^{a\log(b-a)-2}}.\end{aligned}$$

□

LEMMA 2.5. *$Pr(H^D \ge 16d\log m \mid G^D \le d\log m) \le m^{-d}$.*

PROOF. Since $G^D \le d\log m$, at most $d\log m$ distinct messages enter the delay sequence. Once in the delay sequence, each message stays in for a number of steps that is distributed geometrically with parameter $\frac{1}{2}$. Owing to the structure of the graph, each of these messages may reenter the delay sequence one more time after

leaving it, and again stays in for a number of steps that is distributed geometrically with parameter $\frac{1}{2}$. Therefore, H^D is the sum of at most $2d\log m$ geometric random variables, that is, H^D is bounded by a negative binomial random variable with parameters $2d\log m$ and $\frac{1}{2}$ (see, e.g., [6, pp. 164–167]). Hence,

$$\begin{aligned}\Pr(H^D \geq 16d\log m \mid G^D \leq d\log m) &\leq \sum_{0\leq i\leq 2d\log m} \binom{16d\log m}{i}\left(\frac{1}{2}\right)^{16d\log m} \\ &\leq \exp^{-(3/4)^2 4d\log m} \qquad \text{(by fact 1)} \\ &\leq m^{-9/4d} \leq m^{-d}.\end{aligned}$$

□

Putting everything together and taking $a \geq c + 7$ (and hence $\zeta \geq (c + 7)\log m$), $b = a + 4$, $d > 26(c + 7)$ and $c' = 17d$, we obtain

$$\begin{aligned}\Pr\left(\sum_i f_i^D \geq c'\log m\right) &= \Pr(G^D + H^D \geq 17d\log m) \\ &\leq \Pr(G^D \geq d\log m) + \Pr(H^D \geq 16d\log m \mid G^D \leq d\log m) \\ &\leq \Pr(G^D \geq d\log m \mid \text{all } X_i^D\text{'s are small}) + \Pr(\text{some } X_i^D \text{ is big}) \\ &\quad + \Pr(H^D \geq 16d\log m \mid G^D \leq d\log m) \\ &\leq \frac{1}{m^{c+6}} + \frac{1}{m^{a\log(b-a)-2}} + \frac{1}{m^d} \qquad \text{by Lemmas 2.1, 2.4, 2.5} \\ &\leq \frac{1}{m^{c+5}}.\end{aligned}$$

□

3. *A Probabilistic Lower Bound*

THEOREM 3.1. *Let $\mathscr{A}$ be any on-line probabilistic algorithm for simulating an arbitrary (n, m)-PRAM program, m polynomial in n, on a bounded degree network of n processors with no shared memory. Then there exists a T-step PRAM program P on which the average running time of $\mathscr{A}$ is $\Omega(T\log n)$ steps.*

PROOF OF THEOREM 3.1. The probabilistic algorithm takes as input a PRAM program. We use the following reduction due to Yao.

THEOREM 3.2 [18]. *Let T_1 be the expected running time for a given probabilistic algorithm solving problem P, maximized over all possible inputs. Let T_2 be the average running time for a given input distribution, minimized over all possible deterministic algorithms to solve P. Then $T_1 \geq T_2$.*

Let $\mathscr{A}$ be the best deterministic algorithm. There are two cases to consider. If $\mathscr{A}$ keeps an average of $\log n$ copies of each variable, then on average every write instruction takes time $\Omega(\log n)$, since each processor storing a copy of a variable must receive a message with the updated value. If, on the other hand, $\mathscr{A}$ keeps less than an average of $\log n$ copies of each variable, then at least half of the items have less than $2\log n$ copies. If d is the maximum degree of any node in the network, and if v is one of these sparsely-stored variables, then at most $2\sqrt{n}\log n$ of the processors are within a distance of $\log_d n/2$ away from some copy of v.

A random read request (specified by a vector of variables of length n, the ith variable requested by the ith processor) has on average half of the variables requested among those with less than $2\log n$ copies. Each of these variables will be requested with high probability by a processor whose distance from each copy of the variable is more than $\log_d n/2$. Hence, with high probability, each random read request will take $\Omega(\log n)$ time.

Therefore, if we consider the PRAM program that consists of alternating read requests and write requests, with each variable requested chosen uniformly at random, the best deterministic algorithm will on average take $\Omega(T \log n)$ time and hence any probabilistic algorithm will have some input on which it will take an expected $\Omega(T \log n)$ steps. □

4. *A Deterministic Lower Bound*

Though Theorem 3.1 gives a tight bound for the probabilistic complexity of simulating a T step PRAM program on a bounded-degree network of processors, the deterministic complexity of this problem is still open. The best upper bound is $O(T(\log n)^2)$ [1], and the best lower bound is $\Omega(T \log n/\log\log n)$ [15]. (The lower bound in [15] is for simulating a PRAM program on a *complete* network of processors containing no shared memory.) All known algorithms for this problem (deterministic or probabilistic) assume a *point-to-point communication model* in which a processor has to send a separate message in order to update each copy of each variable.[3] In this section we extend the proof in [15] to show that for programs of superlinear length in this model of communication, an $O(T \log n)$ upper bound is not attainable.

It has been proved in [15] that any polylogarithmic deterministic simulation must store several copies of most of the variables. We show that copies of variables written by processor i will need to be distributed amongst processors relatively far away from i in the network. Intuitively, if these variables are not stored at processors a large distance from processor i, then they have all their copies in just a small number of memories and so a read request for them is overly time consuming. Suppose that, at time t, processor i has updated the set of $\Omega(n^{1+\epsilon})$ variables V_t. We define the *redundancy* r_i^t to be the average number of copies of variables in V_t that are stored at processors a distance at least $\log_d n/4$ away from processor i (where d is the maximum degree in the network).

LEMMA 4.1. *Consider a bounded-degree network with n processors where processor i has written $m/n = \Omega(n^{1+\epsilon})$ variables with redundancy r_i^t (with $r_i^t \le \log^2 n$). Then there exists a sequence of n read requests $R_1, R_2, \ldots, R_n$ such that R_i requires at least $\min(n^{\epsilon/2r_i^t}, \log^2 n)$[4] parallel steps (for some $\epsilon > 0$).*

PROOF. The ith read request R_i will have each processor read a variable that was written by processor i. Let P_i be the set of nodes whose minimum distance from node i is at least $\log_d n/4$, where d is the maximum degree of any node in the network. Define p_i to equal $|P_i|$. Clearly $p_i \ge n - n^{1/4}$. Note that r_i^t is the redundancy of those items whose origin is at processor i and which are located at nodes in P_i. Also since r_i^t is the average number of updated copies of data items in this scheme, there are at least $m/2n$ data items originating at processor i with no more than $2r_i^t$ copies. We consider only these data items for the remainder of the proof.

We begin by showing that among processors in P_i there is a particular set of processors storing all the copies of some p_i items. Suppose that y is the smallest number such that there is no set of $\lceil p_i/y \rceil$ processors storing all the copies of some

[3] The most general model would minimize communication by computing the smallest tree (composed of nodes and edges in the network) connecting the writing processor to the variables' locations, and then sending one copy down each branch of this tree.

[4] The $\log^2 m$ term here could be replaced by something stronger. Such a replacement, however, will not be useful in the proof of the general lower bound.

p_i items. Consider a matrix whose rows are indexed by sets of processors of size $\lceil p_i/y \rceil$ and whose columns are indexed by the $m/2n$ items that have redundancy at most $2r_i'$. The i, j element of this matrix is 1 if all of the copies of item j are stored in the set of $\lceil p_i/y \rceil$ processors indexed by i, and 0 otherwise. Then the number of 1's in each column is at least

$$\binom{p_i - 2r_i'}{\lceil p_i/y \rceil - 2r_i'},$$

and so the number of 1's in the whole matrix is at least

$$\frac{m}{2n}\binom{p_i - 2r_i'}{\lceil p_i/y \rceil - 2r_i'}.$$

On the other hand, if none of these sets of processors contains all the copies of some p_i items, each row can contain at most $p_i - 1$ 1's, and so the whole matrix contains at most

$$\binom{p_i}{\lceil p_i/y \rceil}(p_i - 1) \text{ 1's.}$$

Hence

$$\binom{p_i}{\lceil p_i/y \rceil}(p - 1) \geq \frac{m}{2n}\binom{p_i - 2r_i'}{\lceil p_i/y \rceil - 2r_i'}.$$

Manipulating this equation, we get for some $\epsilon' > 0$, $y \geq n^{\epsilon'/2r_i'}$. Hence, for $x = \lfloor n^{\epsilon/2r_i'} \rfloor$ $(\epsilon < \epsilon')$, there *is* some set of nodes of size $\lceil p_i/x \rceil$ in P_i containing all the copies of some p_i items.

Consider a read request by processors in P_i for this set of p_i items. Suppose that the number of parallel steps needed to satisfy it is less than $\log^2 n$. Then at most $\lfloor n^{1/4}\log^2 n \rfloor$ items can be satisfied by copies which come from the processors a distance less than $\log_d n/4$ from processor i (these copies are not counted in the redundancy). Hence, there is a set of $\lceil p_i/x \rceil$ processors in P_i containing all the copies of some $p_i - \lfloor n^{1/4}\log^2 n \rfloor$ items being requested and at least one copy of each of these must exit this set of processors in order to satisfy the read request. Therefore, the time needed to get them all out is at least

$$\frac{p_i - \lfloor n^{1/4}\log^2 n \rfloor}{\lceil p_i/x \rceil} = \lfloor n^{\epsilon/2r_i'} \rfloor (1 - o(1)). \qquad \square$$

If the simulation keeps many copies of each variable in the network (high redundancy), then writes are very time consuming. If on the other hand, the redundancy is low, Lemma 4.1 illustrates that there are very time-consuming reads. We utilize this trade-off to obtain a lower bound on the overall simulation time.

THEOREM 4.1. *Any on-line simulation of T (n, m)-PRAM instructions $(m = \Omega(n^{2+\epsilon}), T = \Omega((1 + \epsilon')m/n))$ on a bounded degree network of n processors in the point-to-point communication model requires $\Omega(T(\log n)^2/\log\log n)$ parallel steps.*

PROOF. We construct an EREW-PRAM program of length T as follows: The first $\tau = m/n$ instructions assign new values to all the data items. Subsequent instructions alternate between a sequence of n *difficult reads* and n *difficult writes*. We think of one sequence of n difficult reads followed by one sequence of n difficult writes as a superinstruction. *Difficult reads* correspond to the sequence of

reads defined by the lemma. *Difficult writes* have each processor assign new values to the n items with the highest number of copies among those m/n items that that processor initially wrote to. Notice that when a processor writes to an existing data item, the simulation algorithm is not obliged to update all the copies currently in existence. However, only those that are updated remain "valid" copies and only those are counted in the computation of the redundancy at later times. Therefore, the redundancy changes over time, in accordance with the algorithm.

Let $(r_1^t, r_2^t, \ldots, r_n^t)$ be the vector of redundancies after the execution of the tth (super)instruction. (For the first m/n steps t will enumerate single write instructions and after that t will enumerate superinstructions.) Also let s_t be the number of parallel steps used in executing the tth (super)instruction. Each copy of a data item (that is counted in the redundancy vector) requires at least $\log_d n/4$ sequential steps to create, since only copies residing at least that distance away from the writing processor are counted. Furthermore, if s_t parallel steps are used to simulate the tth instruction, then at most $n \cdot s_t$ total sequential steps are used. Therefore, the number of sequential steps executed in the first m/n parallel steps is at most $n \sum_{1 \le j \le \tau} s_j$; from the statement above, it is also at least

$$\frac{m}{n}\left(\frac{\log_d n}{4} \sum_{1 \le i \le n} r_i^\tau\right).$$

Hence,

$$\frac{1}{n}\frac{m}{n}\frac{\log_d n}{4} \sum_{1 \le i \le n} r_i^\tau \le \sum_{1 \le j \le \tau} s_j, \tag{1}$$

where $\tau = m/n$.

If we let $\beta_{t,j}$ be the number of the s_t parallel steps used to execute the jth read superinstruction ($t > \tau$), then since s_t parallel steps were taken for the sequence of n reads and writes, then at most $(s_t - \sum_{1 \le j \le n} \beta_{t-1,j})n$ total (sequential) steps were taken for the writes, where by the lemma $\beta_{t-1,j} = \min(\lfloor n^{\epsilon/2r_j^{t-1}} \rfloor, \log^2 n)$. In these n writes, the values of n^2 data items are changed. Again, each new copy of the changed data items requires at least $\log_d n/4$ sequential steps to create. Therefore, the redundancies after the tth instruction can be related to those before the tth instruction as follows.

$$(r_1^t + \cdots + r_n^t)\frac{m}{n} \le (r_1^{t-1} + \cdots + r_n^{t-1})\left(\frac{m}{n} - n\right) + \left(\frac{s_t - \sum_{1 \le j \le n} \beta_{t-1,j}}{(\log_d n)/4}\right)n.$$

Rewriting this equation, we obtain

$$\sum_{1 \le i \le n} r_i^t \le \sum_{1 \le i \le n} r_i^{t-1} + \frac{n}{m}\left[\left(\frac{s_t - \sum_{1 \le j \le n} \beta_{t-1,j}}{\log_d n}\right)4n - n \sum_{1 \le i \le n} r_i^{t-1}\right].$$

Summing the last equation over $\tau < t \le \tau + T'$, where $T' = (T - \tau)/n$, and multiplying by m/n^2 we get

$$\begin{aligned}\frac{m}{n^2} \sum_{1 \le i \le n} r_i^\tau + \sum_{\tau+1 \le t \le \tau+T'} \frac{4s_t}{\log_d n} &\ge \frac{m}{n^2} \sum_{1 \le i \le n} r_i^{\tau+T'} \\ &+ \sum_{\tau+1 \le t \le \tau+T'}\left[\frac{4}{\log_d n} \sum_{1 \le j \le n} \beta_{t-1,j} + \sum_{1 \le i \le n} r_i^{t-1}\right].\end{aligned} \tag{2}$$

We now obtain a lower bound on the total number of parallel steps executed.

$$\begin{aligned}\sum_{1\le t\le \tau+T'} s_t &= \sum_{1\le t\le \tau} s_t + \sum_{\tau+1\le t\le \tau+T'} s_t \\ &\ge \frac{m\log_d n}{4n^2} \sum_{1\le i\le n} r_i^{\tau} + \sum_{\tau+1\le t\le \tau+T'} s_t \qquad \text{by (1)} \\ &\ge \sum_{\tau+1\le t\le \tau+T'} \Bigg[\min(\lfloor n^{\epsilon/2r_n^{t-1}}\rfloor, \log^2 n) + \frac{\log_d n}{4} r_1^{t-1} \\ &\qquad + \cdots + \min(\lfloor n^{\epsilon/2r_n^{t-1}}\rfloor, \log^2 n) + \frac{\log_d n}{4} r_n^{t-1}\Bigg],\end{aligned}$$

where the last inequality follows by applying (2) and substituting for $\beta_{t-1,j}$.

Since $[\min(\lfloor n^{\epsilon/2r_i^{t-1}}\rfloor, \log^2 n) + (\log_d n)/4\ r_i^{t-1}]$ is minimized when $r_i^{t-1} = \Omega(\log n/\log\log n)$, we get

$$\sum_{1\le t\le \tau+T'} s_t \ge c \sum_{\tau+1\le t\le \tau+T'} n\frac{\log^2 n}{\log\log n} = \Omega\left[\left(T - \frac{m}{n}\right)\frac{\log^2 n}{\log\log n}\right].$$

Hence, for $m \ge n^{2+\epsilon}$ and $T \ge (1 + \epsilon')(m/n)$, the simulation requires $\Omega(T\log^2 n/\log\log n)$ parallel steps. □

A lower bound similar to that of Theorem 4.2 was proven independently in [1].

ACKNOWLEDGMENTS. We wish to thank K. Mehlhorn for suggesting the idea of combining the use of universal hash functions with probabilistic routing and for useful comments about earlier versions of this paper. We are also extremely grateful to Cynthia Dwork, Richard Karp, Mark Manasse, Nick Pippenger, Jeff Ullman, and Andy Yao for fruitful discussions relating to this work. Part of this work was done while Eli Upfal was visiting Stanford University. Eli Upfal wishes to thank Jeff Ullman and the Computer Science Department at Stanford for their generous hospitality and support.

REFERENCES

1. ALT, H., HAGERUP, T., MEHLHORN, K., AND PREPARATA, F. P. Simulation of idealized parallel computers on more realistic ones. *SIAM J. Comput. 16*, 5 (1987), 808–835.
2. ANGLUIN, D., AND VALIANT, L. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comput. Syst. Sci. 12*, 6 (1979), 155–193.
3. AWERBUCH, B., ISRAELI, A., AND SHILOACH, Y. Efficient simulation of PRAM by Ultracomputer. Preprint, Technion, Haifa, Israel. 1983.
4. CARTER, L., AND WEGMAN, M. Universal classes of hash functions. *J. Comput. Syst. Sci. 18*, 2 (1979), 143–154.
5. CHERNOFF, H. A measure of asymptotic efficiency for tests of hypothesis based on the sum of observations. *Ann. Math. Statistics 23* (1952), 493–507.
6. FELLER, W. *An Introduction to Probability Theory and Its Applications.* Vol. I, 3rd ed. Wiley, New York, 1967.
7. HOEFFDING, W. On the distribution of the number of successes in independent trials. *Ann. Math. Statistics 27* (1956), 713–721.
8. KUCK, D. J. A survey of parallel machines organization and programming. *ACM Comput. Surv. 9*, 1 (1977), 29–59.
9. MEHLHORN, K., AND VISHKIN, U. Randomized and deterministic simulation of PRAMs by parallel machines with restricted granularity of parallel memories. In *Proceedings of the 9th Workshop on Graph Theoretic Concepts in Computer Science.* Fachbereich Mathematic, Universität Osnabruck, Osnabruck, Austria, June 1983.

10. REIF, J., AND VALIANT, L. A logarithmic time sort for linear size networks. In *Proceedings of 15th ACM Symposium on Theory of Computing* (Boston, Mass., Apr. 25–27). ACM, New York, 1983, pp. 10–16.
11. SCHWARTZ, J. T. Ultracomputers. *ACM Trans. Prog. Lang. Syst. 2*, 4 (Oct. 1980), 484–521.
12. ULLMAN, J. D. *Computational Aspects of VLSI.* Computer Science Press, 1984.
13. UPFAL, E. A probabilistic relation between desirable and feasible models of parallel computation. In *Proceedings of the 16th ACM Symposium on Theory of Computing* (Washington, D.C., Apr. 30–May 2). ACM, New York, 1984, pp. 258–265.
14. UPFAL, E. Efficient schemes for parallel communication. *J. ACM 31*, 3 (1984), 507–517.
15. UPFAL, E., AND WIGDERSON, A. How to share memory in a distributed system. *J. ACM 34*, 1 (Jan. 1987), 116–127.
16. VALIANT, L. A scheme for fast parallel communication. *SIAM J. Comput. 11*, 2 (1982), 350–361.
17. VISHKIN, U. A parallel-design distributed-implementation general-purpose computer. *J. Theoret. Comput. Sci. 32*, 1–2 (1984), 157–172.
18. YAO, A. C. A probabilistic computation: Towards a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Foundations of Computer Science.* IEEE, New York, 1977, pp. 222–227.

RECEIVED OCTOBER 1986; REVISED JUNE 1987, DECEMBER 1987, AND APRIL 1988; ACCEPTED APRIL 1988