The Complexity of Deterministic PRAM Simulation on Distributed Memory Machines[1]

Andrea Pietracaprina

Dipartimento di Matematica
Pura e Applicata
Università di Padova
I35131 Padova, Italy

Geppino Pucci

Dipartimento di Elettronica
e Informatica
Università di Padova
I35131 Padova, Italy

CONTACT AUTHOR:

Geppino Pucci
Dipartimento di Elettronica e Informatica
Università di Padova
Via Gradenigo 6/A
I35131-Padova, Italy
E-mail: `geppo@artemide.dei.unipd.it`
Phone: +39 49 8287726
Fax: +39 49 8287699

[1]A preliminary version of this work was presented at the *2nd European Symposium on Algorithms*, Papendal NL, September 1994 [14]. This research was supported in part by MURST of Italy and by the ESPRIT III Basic Research Programme of the EC under contract No. 9072 (Project GEPPCOM).

Abstract

In this paper we present lower and upper bounds for the deterministic simulation of a Parallel Random Access Machine (PRAM) with n processors and m variables on a Distributed Memory Machine (DMM) with $p \leq n$ processors. The bounds are expressed as a function of the redundancy r of the scheme (i.e., the number of copies used to represent each PRAM variable in the DMM), and become tight for any m polynomial in n and $r = \Theta(1)$.

1 Introduction

An (n, m)-PRAM consists of n processors that have direct access to m shared *variables*. The important feature of this model, which makes it very attractive for the design of parallel algorithms, is that in a PRAM *step*, executed in unit time, any set of n variables can be read or written in parallel by the processors. For large values of m and n, however, this assumption represents a serious obstacle to any realistic implementation. In practice, one has to simulate the PRAM on a more feasible machine, where the shared variables are distributed among memory modules local to the processors. In such a machine, the modules become a bottleneck, since only one item per module can be accessed in unit time.

In this paper we study the complexity of simulating an (n, m)-PRAM on a p-DMM, which consists of $p \leq n$ processors, each provided with a local memory module, communicating through a complete interconnection. In a DMM *step*, each processor can issue an access request for an arbitrary module, but only one request per module is served. Although this model is still unfeasible, in that it idealizes interprocessor communication, it provides a good framework to assess the difficulty of memory distribution in a topology-independent setting.

This paper deals with deterministic PRAM simulations. In order to avoid trivial worst-case scenarios, in which a few modules are overloaded with requests, it is necessary to replicate each variable into a number of *copies* stored in distinct modules, so that, during the simulation of a PRAM step, contention at the modules can be reduced by carefully choosing the copies that have to be accessed. The number of copies used for each variable is called the *redundancy* of the simulation scheme. We say that an (n, m)-PRAM can be simulated on a p-DMM with *slowdown* s, if any $T \geq 1$ PRAM steps can be simulated in $O(Ts)$ DMM steps, in the worst case.

1.1 Previous Work

In the last decade, a large number of randomized and deterministic PRAM simulation schemes have been developed in the literature. All randomized schemes are based on the use of universal classes of hash functions to allocate the variables to the modules. The distribution properties of these functions yield very efficient simulations in the probabilistic

sense. For instance, using a few copies per variable, work-efficient simulations exhibiting triply logarithmic slowdown can be achieved with high probability [3].

In contrast, the development of fast deterministic PRAM simulations appears to be much harder. The pioneering work of Mehlhorn and Vishkin [10] introduced the idea of representing each variable by several copies, so that a read operation needs to access only one (the most convenient) copy. For $m = O(n^r)$, they present a scheme that uses r copies per variable and allows a set of n reads to be satisfied in time $O\left(rn^{1-1/r}\right)$ on an n-DMM. However, the execution of write operations, where all the copies of the variables must be accessed, is penalized and requires $O(rn)$ time in the worst case.

Later, Upfal and Wigderson [17] proposed a more balanced protocol requiring that, in order to read or write a variable, only a majority of its copies be accessed. They also represent the allocation of the copies to the modules (*Memory Organization Scheme* or, for short, MOS) by means of a bipartite graph $G = (V, U)$, where V is the set of PRAM variables, U is the set of DMM modules, and r edges connect each variable to the modules storing its r copies. For m polynomial in n and $r = \Theta(\log n)$, the authors show that there exist suitably expanding graphs that guarantee a worst-case $O\left(\log n (\log\log n)^2\right)$ slowdown on an n-DMM. Using a more complex access strategy, Alt *et al.* [1] improved the bound to $O(\log n)$. In all these schemes, the existence of the underlying MOS graphs is only proved by counting arguments, but no efficient constructions are known. Several authors devised similar, nonconstructive schemes for bounded-degree network machines [9, 4, 5, 6].

Recently, Pietracaprina and Preparata gave the first constructive deterministic schemes for the n-DMM that exhibit sublinear slowdown for both read and write operations. In [12], two PRAM simulations with $m = O(n^2)$ and $m = O(n^3)$ variables are given, respectively with $O\left(n^{1/2}\right)$ and $O\left(n^{2/3}\right)$ slowdown and redundancy $r = 3$ and $r = 5$. In [13], an $O\left(n^{1/3}\right)$ slowdown is achieved with $m = O\left(n^{3/2}\right)$ and $r = 3$. Constructive schemes with nearly optimal slowdown have been recently developed for the mesh topology in [16, 15].

The first significant lower bound for PRAM simulations on the DMM appeared in [17]. The authors tailor their argument to the n-DMM and obtain a bound which is independent of the redundancy of the simulation scheme. However, by a simple modification of their proof,

it can be shown that any simulation with redundancy r and m polynomial in n requires

$$\Omega\left(\frac{\log(m/n)}{\log\log(m/n)} + \min\left\{\frac{n}{r}, \left(\frac{m}{n}\right)^{\frac{1}{r}}\right\}\right) \tag{1}$$

slowdown on an n-DMM. Lower bounds have been also developed for networks of bounded degree in [1, 7, 4, 5]. However, the techniques used to prove such bounds are of a slightly different nature, since bandwidth issues have to be taken into account.

1.2 New Results

The goal of this paper is to study the complexity of deterministic (n, m)-PRAM simulation on a p-DMM, with $p \leq n$, as a function of the redundancy r. We improve upon the result of [17] by providing a tighter lower bound and showing that there exist suitable schemes that achieve optimal slowdown when r is fixed independently of n, m and p. In Section 2, we prove the following

Result 1 *Let $m > n$, $p \leq n$, $r \geq 1$ and $\mu = \lfloor r/2 \rfloor + 1$. Any algorithm which simulates an (n, m)-PRAM on an p-DMM with redundancy r, has worst-case slowdown at least*

$$\Omega\left(\frac{n}{p}\left(\min\left\{\frac{\log(m/n)}{\log\log(m/n)}, \sqrt{p}\right\} + \min\left\{\frac{p}{r}, \left(\frac{m}{n}\right)^{\frac{1}{\mu}}\right\}\right)\right).$$

Note that for $p = n$ and $r = o\left(\log(m/n) / \log\log(m/n)\right)$, this bound improves quadratically upon (1). Moreover, its proof reveals that the majority protocol captures the best trade-off between the complexity of read and write operations, as suggested by intuition. As a by-product, the bound also shows that the explicit schemes of [12] are optimal.

Result 1 shows that work-efficient deterministic simulations are impossible unless $m = \Theta(n)$ or $p = O(1)$, therefore answering a long-standing open question in the PRAM simulation literature. Although the lower bound is proved under the somewhat restrictive assumption that each variable is replicated in a fixed number of copies whose locations do not vary with time (in fact, all the existing deterministic simulation schemes satisfy this assumption), in Section 2.1, we show that work-efficient deterministic simulations are still impossible even under very general assumptions.

In Section 3.1, we propose a scheme to simulate an (n, m)-PRAM on a n-DMM. The scheme relies on suitably expanding MOS's, whose existence is proved within the section. We have:

Result 2 *Let $m = n^{1+\epsilon}$, with arbitrary $\epsilon > 0$. For $r = \Omega((1+\epsilon)\log(1+\epsilon))$, there is a scheme to simulate an (n, m)-PRAM on an n-DMM with redundancy r and slowdown*

$$O\left(r \log^2 r + \log n \log r + r \left(\frac{n}{r}\right)^{\frac{\epsilon}{\mu}} \left(\log r + \frac{1}{\epsilon}\right)\right),$$

where $\mu = \lfloor r/2 \rfloor + 1$.

The simulation slowdown comes very close to the lower bound. In particular, using sublogarithmic redundancy, we obtain a slight improvement on the scheme of [17].

In Section 3.2, we modify the access protocol of the scheme to run on a p-DMM, for any $p \leq n$, obtaining the following result:

Result 3 *Let $m = n^{1+\epsilon}$, with arbitrary $\epsilon > 0$, and let $p \leq n$. For $r = \Omega((1+\epsilon)\log(1+\epsilon))$, there is a scheme to simulate an (n, m)-PRAM on an p-DMM with redundancy r and slowdown*

$$O\left(\min\left\{n, \frac{n}{p}\left(r \log r \log n + \log^2 n + r\left(\frac{n}{r}\right)^{\frac{\epsilon}{\mu}}\left(\log r + \frac{1}{\epsilon}\right)\right)\right\}\right).$$

where $\mu = \lfloor r/2 \rfloor + 1$.

Compared to the lower bound, the above slowdown becomes optimal for any $p \leq n$, when m is polynomial in n and r is a constant. An important property of this second scheme is that it can be implemented with no loss of efficiency on a suitable bounded-degree network. Therefore, Result 3 also holds for such sparser (and feasible) interconnection.

2 Lower Bound

A number of literature papers contain lower bounds on the slowdown of deterministic PRAM simulations on the DMM [17] or on bounded-degree networks [1, 7, 4, 5]. All such bounds are expressed in terms of the parameters n and m, and implicitly optimize on the value of the redundancy. Since all deterministic simulation schemes developed until now use a fixed

number r of copies per variable, we think useful to estimate the best possible slowdown achievable as a function of r. Specifically, we present a general lower bound on the slowdown of any deterministic simulation of an (n, m)-PRAM on a p-DMM, as a function of n, m, p and the redundancy r. The lower bound is proved by refining the standard approach of [17] and adopted thereafter. Our argument relies upon the following assumptions:

1. For each variable, the number and location of the copies do not vary with time. Moreover, without loss of generality, we assume that copies of the same variable are stored in distinct modules (which requires $p \geq r$).

2. Simulations are *on-line*, that is, the simulation of a step starts only after the simulation of the previous step is completed.

We first need two technical lemmas based on a well known combinatorial argument.

Lemma 1 *Given $m \geq n$ PRAM variables, each with at most $\rho \geq 1$ updated copies distributed among p DMM modules ($p \leq n$), there exists a set of n variables requiring*

$$\Omega\left(\frac{n}{p} \cdot \min\left\{\frac{p}{\rho}, \left(\frac{m}{n}\right)^{\frac{1}{\rho}}\right\}\right)$$

DMM steps to be read.

Proof: It is sufficient to show that there exist n variables whose updated copies are all concentrated into at most $\max\left\{2\rho, p\,(n/m)^{1/\rho}\right\}$ modules. Let t be the maximum value for which no set of t modules stores all the updated copies of n variables. If $t < 2\rho$ we are done. Let $t \geq 2\rho$. Consider a matrix with $\binom{p}{t}$ rows indexed by all subsets of t modules, and m columns indexed by the variables. Entry (i, j) of this matrix is 1 if the j-th variable has all its updated copies stored in modules of the i-th subset, 0 otherwise. Each row accounts for at most $n - 1$ 1's. Each column accounts for at least $\binom{p-\rho}{t-\rho}$ 1's. Thus,

$$\binom{p}{t}(n-1) \geq m\binom{p-\rho}{t-\rho} \ ,$$

which implies that $t = O\left(p\,(n/m)^{1/\rho}\right)$. □

Lemma 2 *Consider $m \geq n$ PRAM variables, each represented by r copies distributed among $p \leq n$ DMM modules, and let $\mu = \lfloor r/2 \rfloor + 1$. There exists a set of n variables Φ, and a set of modules S such that each variable in Φ has at least μ copies stored in modules of S, and*

$$|S| = O\left(\max\left\{r, p\left(\frac{n}{m}\right)^{\frac{1}{\mu}}\right\}\right).$$

Proof: Let t be the maximum value for which no set of t modules stores μ copies of each of n variables. If $t < r$ we are done. Let $t \geq r$. Consider a matrix with $\binom{p}{t}$ rows indexed by all subsets of t modules, and m columns indexed by the variables. Entry (i, j) of this matrix is 1 if the j-th variable has at least μ copies stored in modules of the i-th subset, 0 otherwise. Each row accounts for at most $n - 1$ 1's. Each column accounts for exactly $\sum_{y=\mu}^{r} \binom{r}{y}\binom{p-r}{t-y}$ 1's. Therefore, it must be

$$\binom{p}{t}(n-1) \geq m \sum_{y=\mu}^{r} \binom{r}{y}\binom{p-r}{t-y},$$

which implies that $t = O\left(p\,(n/m)^{1/\mu}\right)$. □

Theorem 1 *Let $m > n$, $p \leq n$, $r \geq 1$ and $\mu = \lfloor r/2 \rfloor + 1$. For any algorithm which simulates an (n, m)-PRAM on a p-DMM with redundancy r, there exists a sequence of $T = \Theta(m/n)$ PRAM steps requiring simulation time*

$$\Omega\left(T \cdot \frac{n}{p}\left(\min\left\{\frac{\log(m/n)}{\log\log(m/n)}, \sqrt{p}\right\} + \min\left\{\frac{p}{r}, \left(\frac{m}{n}\right)^{\frac{1}{\mu}}\right\}\right)\right).$$

Proof: We first determine a sequence of $\Theta(m/n)$ *hard writes* that update a constant fraction of all the variables. Either these writes are expensive to simulate, since they update many copies of most of the variables, or we are able to determine $\Theta(m/n)$ *hard reads* that access variables with few updated copies, therefore establishing the stipulated bound.

The hard writes are determined as follows. Let V be the set of m PRAM variables. By repeatedly applying Lemma 2, we can find $k = \lceil m/2n \rceil$ subsets $\Phi_i \subseteq V$, $1 \leq i \leq k$, of variables, with the following properties:

1. $|\Phi_i| = n$, for $1 \leq i \leq k$;

2. $\Phi_i \cap \Phi_j = \emptyset$ for $1 \leq i \neq j \leq k$;

3. For each Φ_i, with $1 \leq i \leq k$, there exists a set of modules S_i with

$$|S_i| = O\left(\max\left\{r, p\left(\frac{n}{m}\right)^{\frac{1}{\mu}}\right\}\right),$$

such that each variable in Φ_i has at least μ copies stored in modules of S_i.

Observe that each Φ_i is chosen from the set $V - \cup_{j=1}^{i-1}\Phi_j$ containing at least $m/2$ variables.

Consider k PRAM write steps, where the i-th step writes the variables in Φ_i. After the simulation of these steps, partition the set $\cup_{i=1}^{k}\Phi_i$ into three sets V_1, V_2 and V_3 as follows. A variable $v \in \cup_{i=1}^{k}\Phi_i$ is in V_1 if it has less than $(\mu-1)/\lambda$ updated copies (the actual value of λ will be determined later); v is in V_2 if it has at least $(\mu-1)/\lambda$ and at most $\mu-1$ updated copies; v is in V_3 otherwise. Clearly, one of these three sets has cardinality $\Theta(m)$. We distinguish among the following three cases.

Case 1: $|V_1| = \Theta(m)$. By repeatedly applying Lemma 1 we determine $\Theta(m/n)$ PRAM steps that read a constant fraction of the variables in V_1 each requiring simulation time

$$\Omega\left(\frac{n}{p}\min\left\{\frac{p\lambda}{\mu-1}, \left(\frac{m}{n}\right)^{\frac{\lambda}{\mu-1}}\right\}\right)$$

Thus, the overall time needed to simulate the $T = \Theta(m/n)$ write and read steps is

$$\Omega\left(T \cdot \frac{n}{p}\min\left\{\frac{p\lambda}{\mu-1}, \left(\frac{m}{n}\right)^{\frac{\lambda}{\mu-1}}\right\}\right). \tag{2}$$

Case 2: $|V_2| = \Theta(m)$. Since at least $\Omega(m(\mu-1)/\lambda)$ copies have been updated during the simulation of the write steps, the overall time required by these steps is $\Omega((m/p)(\mu-1)/\lambda) = \Omega((m/n)(n/p)(\mu-1)/\lambda)$. Moreover, by repeatedly applying Lemma 1, we can determine $\Theta(m/n)$ PRAM steps that read a constant fraction of variables in V_2, each step requiring simulation time

$$\Omega\left(\frac{n}{p}\min\left\{\frac{p}{\mu-1}, \left(\frac{m}{n}\right)^{\frac{1}{\mu-1}}\right\}\right).$$

Thus, the overall time needed to simulate the $T = \Theta(m/n)$ write and read steps is

$$\Omega\left(T \cdot \frac{n}{p}\left(\frac{\mu-1}{\lambda} + \min\left\{\frac{p}{\mu-1}, \left(\frac{m}{n}\right)^{\frac{1}{\mu-1}}\right\}\right)\right). \tag{3}$$

Case 3: $|V_3| = \Theta(m)$. At least $\Omega(m\mu)$ copies are updated during the simulation of the write steps, hence, as before, the overall time required by these steps is $\Omega((m/n)(n/p)\mu)$. Moreover, there must be $\ell = \Theta(m/n)$ sets $\Phi_{i_1}, \Phi_{i_2}, \ldots, \Phi_{i_\ell}$, with $|\Phi_{i_j} \cap V_3| = \Theta(n)$, $1 \leq j \leq \ell$. Since each variable in $\Phi_{i_j} \cap V_3$ has μ updated copies, then one of these copies must belong to a module in S_{i_j}. Therefore, the i_j-th write step requires

$$\Omega\left(\frac{n}{|S_{i_j}|}\right) = \Omega\left(\min\left\{\frac{n}{r}, \frac{n}{p}\left(\frac{m}{n}\right)^{\frac{1}{\mu}}\right\}\right)$$

simulation time, for $1 \leq j \leq \ell$. Therefore, the overall time needed to simulate the $T = \Theta(m/n)$ write steps is

$$\Omega\left(T \cdot \frac{n}{p}\left(\mu + \min\left\{\frac{p}{r}, \left(\frac{m}{n}\right)^{\frac{1}{\mu}}\right\}\right)\right). \tag{4}$$

The theorem follows by choosing

$$\lambda = \max\left\{1, (\mu-1)\,\alpha\frac{\log\log(m/n)}{\log(m/n)}, \frac{\mu-1}{\sqrt{p}}\right\},$$

for some fixed constant $\alpha > 1$, and computing the minimum among (2), (3) and (4). □

Theorem 1 implies that for any nonconstant p and any $n = o(m)$ it is not possible to devise a deterministic algorithm that simulates an (n, m)-PRAM step on a p-DMM work-efficiently in $O(n/p)$ time, which is the case, instead, for randomized simulations [3]. However, the theorem has been proved under the somewhat restrictive assumptions listed at the beginning of this section, so it is not inconceivable that deterministic work-efficient simulations might be attainable in a more general setting. In fact, the following subsection shows that this is not the case.

2.1 Work-Efficient Deterministic Simulations are Impossible

A general lower bound on deterministic simulations can be obtained by adapting the proof in [17] to hold for any value of $p \leq n$ and to satisfy the very weak assumptions formulated in [4]. Namely,

1. The number and location of the copies of each variable may vary with time;
2. The simulation of an instruction may start only after the simulation of all previous read operations has been completed. (This implies that the execution of consecutive writes is not constrained to follow the order in which they appear in the PRAM program.)

For a variable $v \in V$, let r_v^t denote the number of DMM processors storing updated copies of v at time t of the simulation. Define $r^t = \sum_{v \in V} r_v^t/m$ as the *average redundancy* at time t. Thus, at time t at least $m/2$ variables have less than $2r^t$ updated copies each, and Lemma 1 can be applied to find a set of n variables requiring

$$g(r) = \Omega\left(\frac{n}{p}\min\left\{\frac{p}{r^t}, \left(\frac{m}{2n}\right)^{\frac{1}{2r^t}}\right\}\right)$$

to be read. Also, an average redundancy r^t implies that a total of mr^t copies have been written, which takes $\Omega\left((m/n)(n/p)r^t\right)$ time.

Consider a (n, m)-PRAM program that first initializes all the variables and then lets an adversary choose a set of $\Theta(m/n)$ hard reads. If at some point during the simulation the average redundancy is r, then the writes must have taken $\Omega\left((m/n)(n/p)r\right)$ time. Otherwise, the adversary can make each hard read require $g(r)$ time. Therefore, the total simulation time is

$$\Omega\left(\frac{m}{n}\frac{n}{p}\left(r + \min\left\{\frac{p}{r}, \left(\frac{m}{2n}\right)^{\frac{1}{2r}}\right\}\right)\right).$$

By minimizing the above formula over all possible values of r we obtain:

Theorem 2 *For any algorithm which simulates an (n, m)-PRAM on a p-DMM, with $p \leq n$ and $m > 2n$, there exists a sequence of $T = \Theta(m/n)$ PRAM steps requiring simulation time*

$$\Omega\left(T \cdot \frac{n}{p}\min\left\{\sqrt{p}, \frac{\log(m/n)}{\log\log(m/n)}\right\}\right).$$

Observe that the minimum in the above formula is $O(1)$ only if $m = O(n)$ or $p = O(1)$.

3 Upper Bound

Suppose we want to simulate an (n, m)-PRAM on a p-DMM, with m polynomial in n and $p \leq n$. In this section, we provide a protocol to simulate a PRAM step where the processors read/write n *distinct* variables. The case of concurrent accesses to the same variable can be handled by appending a preprocessing phase to the protocol, where a representative is elected for each subset of processors requiring access to the same variable. The representatives will then perform the simulation algorithm and, in case of read operations, distribute the accessed data to the appropriate processors. Both preprocessing and data distribution phases can be accomplished through standard sorting and prefix operations without increasing the overall slowdown by more than a constant factor.

In Subsection 3.1, we describe the access protocol for the case $p = n$. In Subsection 3.2, we modify the protocol to run on a p-DMM, for any $p \leq n$, also showing how it can be implemented on a specific p-processor bounded-degree network with no loss in efficiency.

3.1 The Case $p = n$

Our simulation scheme follows the ideas pioneered in [17], where each variable is replicated into r copies, distributed among the DMM modules according to a suitable Memory Organization Scheme. The MOS is modeled by a bipartite graph $G = (V, U)$, where V denotes the set of m PRAM variables and U the set of n DMM processors. The graph is *r-regular*, meaning that each node in V has degree r, each node in U has degree mr/n, and the r edges adjacent to a variable v reach the processors storing its copies, with exactly mr/n copies assigned to each module. We assume that given a variable v, a processor can determine the location of any copy of v in $O(1)$ time.

Every copy is stored together with a time-stamp, which is updated every time the copy is written. In order to read or write a variable, at least a majority $\mu = \lfloor r/2 \rfloor + 1$ of its copies have to be accessed (for read operations, the value stored in the copy with the most recent time-stamp is returned). Note that this is sufficient to guarantee consistency, since a read

will always access at least one of the most recently written copies of a variable. Therefore, we say that a variable is *accessed* if at least μ of its copies are accessed.

Accesses to the copies of the variables requested in a PRAM step are governed by a simple protocol, which prescribes that the processors continuously send requests to the modules until enough copies per variable are reached. The efficiency of the scheme relies on the existence of a suitable MOS which makes the access protocol converge rapidly. The protocol is patterned after the one in [17], but performs a reduced number of bookkeeping operations, which is crucial to obtain optimality when $r = O(1)$.

When $p = n$, each DMM processor simulates a distinct PRAM processor and is therefore in charge of a single variable. We partition the DMM processors into n/r *clusters* of r processors each. Let P_j^i denote the jth processor of cluster i, where $1 \leq j \leq r$ and $1 \leq i \leq n/r$. Let also v_j^i denote the variable requested by P_j^i and $v_j^i(1), v_j^i(2), \ldots, v_j^i(r)$ its copies.

The protocol consists of $r+1$ *stages*, numbered from 1 to $r+1$. In Stage s, for $1 \leq s \leq r$, the clusters try to access variables $v_s^1, v_s^2, \ldots, v_s^{n/r}$ in parallel, with Cluster i in charge of v_s^i. More specifically, at the beginning of the stage, each P_s^i, $1 \leq i \leq n/r$ sends the name of its variable v_s^i and the operation to be performed to the other processors in the cluster. P_j^i is now in charge of the copy $v_s^i(j)$ and determines its location. Then, a number of iterations are executed in which the processors repeatedly try to access the copies they are assigned until they succeed or the stage terminates. These iterations are grouped into $\log r$ batches, where the kth batch consists of Φ_k iterations, $1 \leq k \leq \log r$. At the end of each batch, Cluster i counts the number of copies of v_s^i accessed so far and, if these are at least μ, it declares the variable successfully accessed and stops further accesses to its copies. The Φ_k's will be specified in the analysis to guarantee that at the end of the stage at most n/r^2 variables (among $v_s^1, v_s^2, \ldots, v_s^{n/r}$) remain unaccessed. Therefore, at the end of Stage r, there will be a total of $\ell \leq n/r$ unaccessed variables. In Stage $r+1$, these variables are distributed among the first ℓ clusters, one per cluster. Then, $\log_2(n/r)$ batches of iterations are performed. The kth batch consists of Φ_k iterations, $1 \leq k \leq \log_2(n/r)$, followed by counting as in the previous stages. The Φ_k's are chosen to guarantee that this stage complete all accesses (note that the first $\log r$ values of the Φ_k's are the same as those used in Stages 1 to r).

The protocol's pseudocode is reported in Figure 1. The core of a stage is represented by

```
begin MAIN
  for s := 1 to r do { Stage s }
      ACCESS(s, r);
  { Stage r + 1 }
  let ℓ ≤ n/r be the number of unaccessed variables;
  rank the ℓ unaccessed variables and send the ith such variable
    to P_1^i, 1 ≤ i ≤ ℓ, specifying the operation to be performed;
  ACCESS(1, n/r);
end MAIN


procedure ACCESS(s, x)
  for all i ∈ {1, ..., n/r} do in parallel
    P_s^i broadcasts v_s^i to all processors in its cluster,
      specifying the operation to be performed;
  for k := 1 to log_2 x do { Batch k }
    for all i ∈ {1, ..., n/r} do in parallel
      if v_s^i is unaccessed then
        for all j ∈ {1, ..., r} do in parallel
          for h := 1 to Φ_k do
            if v_s^i(j) is not accessed yet then
              P_j^i sends a request for v_s^i(j) to the appropriate module;
            {each module serves one incoming request, if any}
      the processors in Cluster i count the number a_i of accessed copies of v_s^i;
      if a_i ≥ μ then v_s^i is marked accessed
end ACCESS
```

Figure 1: Access protocol for $p = n$

procedure ACCESS. Namely, ACCESS(s, x) deals with n/r variables held by P_s^i, $1 \leq i \leq n/r$ and successfully completes the accesses for all but at most a fraction x of such variables. The correctness of this procedure, as well as that of the entire protocol, depends on the choice of the Φ_k's, which are determined in terms of the expansion capabilities of the MOS.

Definition 1 *Let $G = (V, U)$ be an r-regular, bipartite graph, and let $\mu = \lfloor r/2 \rfloor + 1$. For $\alpha > 0$ and $0 < \delta < 1$, G is said to have* (α, δ)-expansion *if for any subset $S \subset V$, $|S| \leq |U|/r$, and any choice of μ outgoing edges for each node in S, the set $\Gamma^{\mu}(S) \subseteq U$ reached by the chosen edges has size*

$$|\Gamma^{\mu}(S)| > \alpha r |S|^{1-\delta} \ .$$

We have:

Lemma 3 *If the MOS graph G has (α, δ)-expansion, then for $1 \leq k \leq \log_2(n/r)$, choosing*

$$\Phi_k = \frac{2}{\alpha}\left(\frac{n}{r2^k}\right)^{\delta} \tag{5}$$

is sufficient to make the protocol work correctly.

Proof: Let R be the set of n/r variables relative to the call ACCESS(s, x), with $1 \leq x \leq n/r$. It suffices to show that the above choice of the Φ_k's guarantees that ACCESS(s, x) successfully completes the accesses for all but at most a fraction x of these variables. Let $v \in R$. During the execution of the procedure, we say that a copy of v is *alive* if it has not yet been accessed and that v itself is *alive* if at least a majority μ of its copies are alive.

We prove by induction that at the end of Batch k, $k \geq 0$, at most $n/(r2^k)$ variables in R are still alive (for convenience, we consider the beginning of Batch 1 as the end of Batch 0). The basis of the induction is trivial. Let the property be true for some $k \geq 0$, and, for the sake of contradiction, suppose that at the end of Batch $k+1$ more than $n/(r2^{k+1})$ variables are alive. Then, the expansion property of the MOS guarantees that each iteration in the batch accesses at least $\alpha r \left(n/(r2^{k+1})\right)^{1-\delta}$ copies, for a total of

$$\Phi_k \alpha r \left(\frac{n}{r2^{k+1}}\right)^{1-\delta} > \frac{n}{2^k}$$

copies accessed by the end of the batch. Such copies belong only to the variables alive at the end of Batch k, which, by the inductive hypothesis, are at most $n/(r2^k)$ and therefore account for no more than $n/2^k$ copies, yielding the desired contradiction. □

It remains to prove that MOS's with high expansion properties do exist.

Lemma 4 *Let $m = n^{1+\epsilon}$, with arbitrary $\epsilon > 0$. There is a constant c such that for any r, with $c(1+\epsilon)\log(1+\epsilon) \leq r \leq n$, a random r-regular bipartite graph $G = (V, U)$ with $|V| = m$ and $|U| = n$ has (α, δ)-expansion with $\alpha = \Theta(1)$ and $\delta = \epsilon/\mu$, where $\mu = \lfloor r/2 \rfloor + 1$, with high probability.*

Proof: $G = (V, U)$ does not have (α, δ)-expansion if there is a set $S \subset V$ of at most n/r nodes such that some choice of μ outgoing edges for each node in S yields $|\Gamma^{\mu}(S)| \leq \alpha r |S|^{1-\delta}$. As customary, we use the probabilistic method showing that there is a constant α, $0 < \alpha < 1$, such that the fraction of r-regular bipartite graphs G that do not have $(\alpha, \epsilon/\mu)$-expansion is very small. Let us associate the r-regular bipartite graphs with the permutations of the integers $1, 2, \ldots, mr$. The fraction of bad graphs is at most $\sum_{s=1}^{n/r} f(s)$, where

$$f(s) = \binom{m}{s} \binom{n}{\alpha r s^{1-\delta}} \binom{r}{\mu}^{s} \frac{[\alpha r s^{1-\delta}(mr/n)]_{\mu s}}{[mr]_{\mu s}},$$

with $[K]_t = K \cdot (K-1) \cdot \ldots \cdot (K-t+1)$. Note that

$$\frac{[\alpha r s^{1-\delta}(mr/n)]_{\mu s}}{[mr]_{\mu s}} \leq \left(\frac{\alpha r s^{1-\delta}}{n} \right)^{\mu s}.$$

Using Stirling's bounds on binomial coefficients and tedious but simple arithmetic manipulations, we obtain

$$f(s) < \left(\left(\frac{s}{n} \right)^{\mu - (1+\epsilon) - \alpha r} (\alpha r)^{\mu - \alpha r} e^{2r} \right)^{s}.$$

By suitably choosing constants c and α, for $r \geq c(1+\epsilon)\log(1+\epsilon)$, the term within brackets in the right hand side of the above relation is an increasing function of s and, for $s \leq n/r$,

$$\left(\frac{s}{n} \right)^{\mu - (1+\epsilon) - \alpha r} (\alpha r)^{\mu - \alpha r} e^{2r} \leq r^{1+\epsilon} \alpha^{\mu - \alpha r} e^{2r} \leq 2^{-r}.$$

If $r \geq \log_2 n$, we have

$$\sum_{s=1}^{n/r} f(s) \leq \sum_{s=1}^{n/r} 2^{-rs} \leq \frac{2}{n}.$$

If instead $r \leq \log_2 n$, we have

$$\sum_{s=1}^{n/r} f(s) \leq \sum_{s=1}^{\log_2 n - 1} f(s) + \sum_{s=\log_2 n}^{n/r} 2^{-rs}.$$

Again, a suitable choice of the constants c and α yields

$$\sum_{s=1}^{\log_2 n-1} f(s) \leq \sum_{s=1}^{\log_2 n-1} \left[\left(\frac{n}{\log_2 n} \right)^{1+\epsilon} \left(\frac{\alpha r \log_2 n}{n} \right)^{\mu-\alpha r} e^{2r} \right]^s < \sum_{s=1}^{\log_2 n-1} \frac{1}{n^s}.$$

Therefore,

$$\sum_{s=1}^{n/r} f(s) \leq \sum_{s=1}^{\log_2 n-1} \frac{1}{n^s} + \sum_{s=\log_2 n}^{n/r} 2^{-rs} \leq \frac{3}{n},$$

and the thesis follows. □

We are ready to prove the main result of this section.

Theorem 3 *Let $m = n^{1+\epsilon}$, with arbitrary $\epsilon > 0$. For a fixed constant $c > 0$ and any value $r \geq c(1+\epsilon)\log(1+\epsilon)$, there is a scheme to simulate an (n, m)-PRAM on an n-DMM with redundancy r and slowdown*

$$O\left(r \log^2 r + \log n \log r + r \left(\frac{n}{r} \right)^{\frac{\epsilon}{\mu}} \left(\log r + \frac{1}{\epsilon} \right) \right),$$

where $\mu = \lfloor r/2 \rfloor + 1$.

Proof: Consider the access protocol of Figure 1, and assume that the distribution of the variables among the DMM processors is governed by an r-regular MOS with $(\Theta(1), \epsilon/\mu)$-expansion. By Lemma 4, a random r-regular graph will exhibit such expansion with high probability. Let us evaluate the running time of the protocol. Since broadcast and counting within a cluster take time $O(\log r)$, an execution of ACCESS(s, x) is completed in time

$$O\left(\sum_{k=1}^{\log_2 x} (\Phi_k + \log r) \right).$$

The redistribution of the unaccessed variables in Stage $r+1$ can be accomplished via a prefix operation in $O(\log n)$ time. Therefore, the overall running time of the protocol is

$$\begin{aligned} & O\left(r \sum_{k=1}^{\log_2 r} (\Phi_k + \log r) + \log n + \sum_{k=1}^{\log_2 n/r} (\Phi_k + \log r) \right) \\ = \; & O\left(r \left(\left(\frac{n}{r} \right)^{\frac{\epsilon}{\mu}} \log r + \log^2 r \right) + \log n + \left(\frac{n}{r} \right)^{\frac{\epsilon}{\mu}} \frac{r}{\epsilon} + \log n \log r \right), \end{aligned}$$

where the Φ_k's are as in Lemma 3. The theorem follows. □

Note that the protocol's running time differs from the lower bound proved in Theorem 1 by a factor which is a superlinear function of the redundancy, and is therefore suboptimal for nonconstant values of r. Note also that our general scheme attains the same slowdown as the one in [17] for $r = \Theta(\log(m/n))$. Indeed, a slightly smaller slowdown can be achieved by choosing $r = \Theta(\log(m/n)/\log\log\log(m/n))$, which yields a slowdown of $O\left(\log(m/n)(\log\log(m/n))^2/\log\log\log(m/n)\right)$. Finally, our scheme is optimal for constant values of the redundancy and m polynomial in n. In this case, it can be easily shown that $r > 3(\epsilon+1)$ is sufficient to guarantee the existence of MOS graphs with the required expansion. Thus,

Corollary 1 *For any $m = n^{1+\epsilon}$, with constant $\epsilon > 0$, and any constant $r \geq 3(\epsilon+1)$ there is a scheme to simulate an (n,m)-PRAM on an n-DMM with redundancy r and optimal slowdown $\Theta\left(n^{\epsilon/\mu}\right)$, where $\mu = \lfloor r/2 \rfloor + 1$.*

3.2 The General Case

When the number of DMM processors is $p < n$, we would expect the simulation slowdown to increase by a factor n/p. Indeed, an optimal deterministic simulation of an n-DMM on a p-DMM is known when either $p = O(1)$ or $p = \Omega\left(n^{\beta}\right)$, with constant $\beta < 1$, hence, in these cases, the simulation slowdown given in Theorem 3 goes up by a factor $O(n/p)$ as expected. For all other values of p, however, no optimal deterministic simulation of an n-DMM on a p-DMM is known, therefore such scaling is not easy to obtain. In what follows, we present a modified access protocol that does scale for any $p \leq n$ and for most values of r. Moreover, this new protocol can be ported to a suitable Bounded-Degree Network (BDN) with no loss of efficiency. The lower bound proved in Theorem 1 shows that the protocol is optimal for both DMMs and BDNs for any $p \leq n$ and parameters m, n and r varying in the same ranges as those specified in Corollary 1.

Each processor of the p-DMM simulates n/p PRAM processors and is in charge of their variables. Each of the p memory modules is divided into n/p *sectors*, and the $m = n^{1+\epsilon}$ variables are distributed among the n sectors according to an r-regular MOS $G = (V,U)$

with $(\alpha, \epsilon/\mu)$-expansion, where $\mu = \lfloor r/2 \rfloor + 1$. (Note that a DMM module may now store several copies of the same variable.)

The high level structure of the modified access protocol is identical to the one presented for the case $p = n$ (see Fig. 1), the only differences arising in the implementation of procedure ACCESS. The p processors are still organized in p/r clusters of r processors each. Accesses to the modules are organized in a sequence of batches, each serving a suitable number of requests per sector. However, since n/p sectors are now stored into a single DMM module, accesses directed to sectors within the same memory module must be scheduled in such a way that no collisions arise. A word description of the steps performed in ACCESS(s, x) is given below.

1. Processor P_s^i, $1 \leq i \leq p/r$, broadcasts its n/p variables to the other processors of the cluster, so that P_j^i will request the j-th copy of each of such variables. This takes time $O((n/p) + \log r)$.
2. For k batches, $1 \leq k \leq \log x$:
   (a) All the requests are sorted by sectors and ranked, and Φ_k copy requests per sector are selected. This step takes $O((n/p) \log n)$ time (recall that sorting of p keys can be done in time $O(\log p)$ on a $O(p)$-node BDN [8] and therefore on a p-DMM. The extension to $n > p$ keys is obtained by invoking the techniques of Baudet and Stevenson [2]);
   (b) The selected requests are compacted in contiguous processors, n/p per processor. Then, requests destined to the same module are ranked. The rank of each request is referred to as its *access time* t_a, $1 \leq t_a \leq (n/p)\Phi_k$. This step takes $O((n/p) + \log n)$ time;
   (c) Note that requests destined to the same module u are now stored in n_u consecutive processors, $0 \leq n_u \leq \Phi_k$. Using the access times computed in the previous steps, it is easy to perform the actual accesses to the modules in time $O((n/p)\Phi_k)$;
   (d) By reversing the previous steps the accessed data return to their originating clusters, within the same time bounds. At the clusters, counting is performed in time $O((n/p) + \log r)$.

Overall, ACCESS(s, x) takes time

$$O\left(\frac{n}{p} \sum_{k=1}^{\log_2 x} (\Phi_k + \log n)\right).$$

The running time of the entire protocol can be derived by observing that the variable redistribution at the beginning of Stage $r+1$ can be performed in $O((n/p)+\log n)$ time, and by plugging in the values of the Φ_k's given in Lemma 3. By truncating above the running time at n (which can be achieved by a trivial sequential protocol) we obtain:

Theorem 4 *Let $m = n^{1+\epsilon}$, with arbitrary $\epsilon > 0$, and let $p \leq n$. For a fixed constants $c > 0$ and any value r such that $r \geq c(1+\epsilon)\log(1+\epsilon)$, there is a scheme to simulate an (n,m)-PRAM on an p-DMM with redundancy r and slowdown*

$$O\left(\min\left\{n, \frac{n}{p}\left(r\log r\log n + \log^2 n + r\left(\frac{n}{r}\right)^{\frac{\epsilon}{\mu}}\left(\log r + \frac{1}{\epsilon}\right)\right)\right\}\right),$$

where $\mu = \lfloor r/2 \rfloor + 1$.

The theorem implies that the protocol's running time is a factor n/p higher than the protocol of Figure 1 for any $r = O(\log(m/n)/\log\log n)$. Moreover, the protocol is optimal for any $p \leq n$, m polynomial in n and constant r.

Corollary 2 *For any $m = n^{1+\epsilon}$, with constant $\epsilon > 0$, any $p \leq n$ and any constant $r \geq 3(\epsilon+1)$, there is a scheme to simulate an (n,m)-PRAM on an p-DMM with redundancy r and optimal slowdown $\Theta\left(n^{1+\epsilon/\mu}/p\right)$, where $\mu = \lfloor r/2 \rfloor + 1$.*

It is important to remark that the above protocol can be implemented on a BDN within the same time bound. Specifically, Peleg and Upfal [11] present an algorithm for a suitable p-processor BDN that performs (n, k_1, k_2)-routing, where each processor sends (resp., receives) k_1 (resp., k_2) packets and the total number of packets is n, in time $O((n/p)\log p + k_1 + k_2)$. By augmenting their network with the sorting network of [8], we can perform all the tasks prescribed in the implementation of ACCESS within the stipulated time bound. This implies that both Theorem 4 and Corollary 2 also hold for a BDN.

4 Conclusions

The efficient simulation of shared memory in distributed systems is a central problem for parallel computation. Our paper improves the understanding of the deterministic complexity

of this problem, by tightening both lower and upper bounds on the simulation slowdown. For shared memories of size polynomial in the number of processors and constant values of redundancy, we show that optimal slowdowns can be attained. However, there still is a gap between lower and upper bounds for nonconstant values of the redundancy. This gap is largely caused by bookkeeping operations that appear to be needed in the simulation protocol and are difficult to account for in the lower bound. An interesting open question is whether this gap can be reduced or even closed.

Regarding the upper bound, it must be remarked that the schemes presented in Section 3 rely on MOS graphs for which no efficient construction is known, although the existential argument shows that a random graph would exhibit the required expansion property with high probability. Unfortunately, the explicit construction of highly expanding MOS graphs represents a long-standing open problem, and the limitations suffered by our scheme are typical of most simulation schemes in the literature. For specific values of memory size and redundancy, explicit memory organizations have been proposed in [12, 13]. It remains a challenging open problem to extend these results to an entire range of values of the parameters.

References

[1] H. Alt, T. Hagerup, K. Mehlhorn, and F.P. Preparata. Deterministic simulation of idealized parallel computers on more realistic ones. *SIAM Journal on Computing*, 16(5):808–835, 1987.

[2] G. Baudet and D. Stevenson. Optimal sorting algorithms for parallel computers. *IEEE Trans. on Computers*, C-27(1):84–87, January 1978.

[3] A. Czumaj, F. Meyer auf der Heide, and V. Stemann. Shared memory simulations with triple-logarithmic delay. In *Proc. of the 3rd European Symposium on Algorithms*, pages 46–59, 1995.

[4] K. Herley and G. Bilardi. Deterministic simulations of PRAMs on bounded-degree networks. *SIAM Journal on Computing*, 23(2):276–292, April 1994.

[5] K. Herley, A. Pietracaprina, and G. Pucci. Implementing shared memory on multidimensional meshes and on the fat-tree. In *Proc. of the 3rd European Symposium on Algorithms*, pages 60–74, 1995.

[6] K.T. Herley. Representing shared data on distributed-memory parallel computers. *Mathematical Systems Theory*, 29:111–156, 1996.

[7] A.R. Karlin and E. Upfal. Parallel hashing: An efficient implementation of shared memory. *Journal of the ACM*, 35(4):876–892, October 1988.

[8] F.T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Trans. on Computers*, C-34(4):344–354, Apr. 1985.

[9] F. Luccio, A. Pietracaprina, and G. Pucci. A new scheme for the deterministic simulation of PRAMs in VLSI. *Algorithmica*, 5(4):529–544, 1990.

[10] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, 21:339–374, 1984.

[11] D. Peleg and E. Upfal. The token distribution problem. *SIAM Journal on Computing*, 18(2):229–243, April 1989.

[12] A. Pietracaprina and F.P. Preparata. An $O(\sqrt{n})$-worst-case-time solution to the granularity problem. In K.W. Wagner P. Enjalbert, A. Finkel, editor, *Proc. 10th Symp. on Theoretical Aspects of Computer Science*, LNCS 665, pages 110–119, Würzburg, Germany, February 1993. Springer-Verlag.

[13] A. Pietracaprina and F.P. Preparata. A practical constructive scheme for deterministic shared-memory access. In *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, pages 100–109, Velen, Germany, July 1993.

[14] A. Pietracaprina and G. Pucci. Tight bounds on deterministic PRAM emulations with constant redundancy. In J.V. Leeuwen, editor, *Proc. 2nd European Symposium on Algorithms*, LNCS 855, pages 391–400, Utrecht, NL, September 1994. Springer-Verlag.

[15] A. Pietracaprina and G. Pucci. Improved deterministic PRAM simulation on the mesh. In *Proc. 22nd Int. Colloquium on Automata, Languages and Programming*, LNCS, pages 372–383, Szeged, H, July 1995. Springer-Verlag.

[16] A. Pietracaprina, G. Pucci, and J. Sibeyn. Constructive deterministic PRAM simulation on a mesh-connected computer. In *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures*, pages 248–256, Cape May, NJ, June 1994.

[17] E. Upfal and A. Widgerson. How to share memory in a distributed system. *Journal of the ACM*, 34(1):116–127, 1987.