International Journal of Foundations of Computer Science

A GENERAL PRAM SIMULATION SCHEME FOR CLUSTERED MACHINES *

CARLO FANTOZZI, ANDREA PIETRACAPRINA, and GEPPINO PUCCI

Department of Information Engineering, University of Padova
Via Gradenigo 6/B, Padova 35131, Italy
e-mail: {bambu,capri,geppo}@dei.unipd.it

Received November 6, 2002
Revised May 12, 2003
Communicated by Albert Zomaya

ABSTRACT

We present a general deterministic scheme to implement a shared memory abstraction on any distributed-memory machine which exhibits a clustered structure. More specifically, we develop a memory distribution strategy and an access protocol for the Decomposable BSP (D-BSP), a generic machine model whose bandwidth/latency parameters can be instantiated to closely reflect the characteristics of machines that admit a hierarchical decomposition into independent clusters. Our scheme achieves provably optimal slowdown for those machines where delays due to latency dominate over those due to bandwidth limitations. For machines where this is not the case, the slowdown is a mere logarithmic factor away from the natural bandwidth-based lower bound.

Keywords: PRAM Simulation, Processor Networks, BSP, D-BSP, Clustered Machines

1. Introduction

Providing a shared address space on a distributed-memory parallel system is a fundamental problem which has been extensively investigated from both a theoretical and a practical perspective over the last two decades. In the theoretical setting, the problem has been regarded as the simulation of the PRAM model of parallel computation [1] over processor networks. More precisely, PRAM simulation requires the development of a *scheme* to represent m shared objects (called *variables*) onto a network of $n \leq m$ processor/memory pairs in such a way that any n-tuple of variables can be read/written efficiently by the processors. The time required by a parallel access to an arbitrary n-tuple of variables is referred to as the *slowdown* of the scheme.

Both randomized and deterministic schemes have been proposed in the litera-

*A preliminary version of this work appeared in the *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, San Francisco, CA April 2001.

ture for a number of specific networks. Randomized schemes typically distribute the variables among the memory modules local to the processors through one (or more) hash functions, chosen from a suitable universal class. The properties of these functions guarantee, with high probability, that the variables are evenly distributed among the modules so that an access to any n-tuple of variables incurs low congestion both at the memory modules and across the network. Randomized schemes based on this strategy have been proposed to simulate a PRAM step, with high probability, in $O(\log\log\log n \log^* n)$ time on the complete network [2], in $O(\log n)$ time on the butterfly [3] and in $O\left(dn^{1/d}\right)$ time on the d-dimensional mesh [4]. Work-optimal randomized schemes are presented in [5], to simulate an $(n \log n)$-processor PRAM on an n-processor hypercube with slowdown $O(\log n)$, and in [6], to simulate an $(n \log\log n)$-processor PRAM on an n-processor Optical Communication Parallel Computer with slowdown $O(\log\log n)$.

In contrast, deterministic schemes require a redundant representation of the address space where every variable is replicated into ρ copies distributed among the memory modules through a map which exhibits suitable expansion properties. The expansion properties are needed to avoid trivial worst cases where all of the copies of some n-tuple of variables are confined within few memory modules, or, more generally, within a low-bandwidth region of the network. The parameter ρ is referred to as the *redundancy* of the scheme. The main idea, originally introduced in [7] and subsequently refined in [8], is that any access (read or write) to a variable is satisfied by reaching only a subset of its copies which is suitably chosen to reduce congestion while ensuring consistency (i.e., a read access must always return the most updated value of the variable).

Based on the above idea, deterministic schemes have been devised in the literature for a number of network topologies. In particular, for m polynomial in n, schemes are known to simulate a PRAM step on a complete network in time $O(\log n)$ [9], and on a mesh of trees with n processors and n^2 switching elements in time $O\left(\log^2 n / \log\log n\right)$ [10]. For arbitrary values of m, schemes achieving polylogarithmic worst-case slowdown have been proposed for an expander-based network [11], and for an augmented mesh of trees [12]. By employing suitable splitting/combining techniques, the deterministic scheme presented in [13] attains $O\left(\sqrt{n \log(m/n)}\right)$ slowdown on the pruned butterfly (a variant of the fat tree), and $O\left(n^{1/d}(\log(m/n))^{1-1/d}\right)$ slowdown on a d-dimensional mesh. In the same work, a general argument is provided to bound from below the worst-case slowdown of any deterministic simulation scheme as a function of certain bandwidth characteristics of the interconnection.

All of the aforementioned deterministic schemes crucially exploit the specific structure of the underlying network topology and, consequently, are not easily applicable to other topologies. Moreover, they require redundancy logarithmic in m and rely on the existence of very powerful expanding graphs of nonconstant degree for which no explicit construction is known.

The first deterministic schemes that are also fully constructive have been devised

in [14] for the complete network. For $m = \Theta\left(n^{3/2}\right)$, $m = \Theta\left(n^2\right)$ and $m = \Theta\left(n^3\right)$, these schemes attain worst-case slowdown of $O\left(n^{1/3}\right)$, $O\left(n^{1/2}\right)$ and $O\left(n^{2/3}\right)$, respectively, while using constant redundancy. In a recent work [15], a deterministic scheme for an n-processor mesh has been presented, which achieves $O\left(\sqrt{n}\log n\right)$ slowdown and features a novel memory organization based on weakly expanding graphs that can be explicitly constructed for a range of memory sizes. Moreover, the memory map exhibits a hierarchical structure tailored to the natural recursive decomposition of the mesh into submeshes and shows promise to be portable to other clustered architectures.

Building on the techniques of [15], in this paper we develop a general scheme to implement shared memory on parallel machines with a clustered structure, achieving a worst-case slowdown which is optimal, or close to optimal, with respect to the limitations imposed by the machine's bandwidth/latency characteristics. To achieve generality, the scheme is designed for the D-BSP, a generic machine model that can be efficiently supported on a wide class of clustered architectures by a suitable setting of the model's parameters.

In the following two subsections we give a formal definition of the machine model and outline the main results of the paper.

1.1. Machine Model

The *Decomposable Bulk Synchronous Parallel* (D-BSP) model was introduced in [16] as an extension of Valiant's BSP [17] aimed at capturing locality within clusters; in this paper, we employ the following more regular version of the model (referred to as *recursive D-BSP* in [16]). Let n be a power of two, and let $\boldsymbol{g} = (g_0, g_1, \ldots, g_{\log n})$ and[a] $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{\log n})$. A D-BSP $(n, \boldsymbol{g}, \boldsymbol{\ell})$ is a collection of n processor/memory pairs communicating through a router. For $0 \leq i \leq \log n$, the n processors are partitioned into 2^i fixed, disjoint *i-clusters* $C_0^{(i)}, C_1^{(i)}, \cdots, C_{2^i-1}^{(i)}$ of $n/2^i$ processors each, where the processors of a cluster are able to communicate among themselves independently of the other clusters. The clusters form a hierarchical, binary decomposition tree of the D-BSP machine, specifically, $C_j^{\log n}$ contains only processor P_j, for $0 \leq j < n$ and $C_j^{(i)} = C_{2j}^{(i+1)} \cup C_{2j+1}^{(i+1)}$, for $0 \leq i < \log n$ and $0 \leq j < 2^i$.

A D-BSP computation consists of a sequence of labelled supersteps. In an *i-superstep*, $0 \leq i \leq \log n$, each processor executes internal computation on locally held data and sends messages exclusively to processors within its i-cluster. The superstep is terminated by a barrier, which synchronizes processors within each i-cluster independently. It is assumed that messages sent in one superstep are available at the destinations only at the beginning of the subsequent superstep. If each processor performs at most w local operations, and the messages sent in the superstep form an h-relation (i.e., each processor is source or destination of at most h messages), then the cost of the i-superstep is upper bounded by $w + hg_i + \ell_i$. In other words, an i-cluster in isolation behaves like a BSP of parameters g_i and ℓ_i.

[a] In this paper we use $\log x$ to indicate the base-2 logarithm.

Although all of the D-BSP algorithms developed in this paper are correct independently of $\boldsymbol{g}$ and $\boldsymbol{\ell}$, we will analyze their running time for a class of parameter values of particular significance. Namely, let α and β be two arbitrary constants, with $0 < \alpha, \beta < 1$. We will consider D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ machines with

$$\begin{cases} g_i^{(\alpha)} = \Theta\left((n/2^i)^\alpha\right), \\ \ell_i^{(\beta)} = \Theta\left((n/2^i)^\beta\right), \end{cases} \quad 0 \le i \le \log n. \tag{1}$$

Note that these parameters capture a wide family of machines whose clusters are characterized by moderate bandwidth and moderate to high latency. For instance, the values $\alpha = \beta = 1/d$, for any integer $d \ge 1$, yield a D-BSP machine whose bandwidth/latency distribution reflects that of a d-dimensional array. Indeed, by combining the routing results of [18] with those in [19] it can be shown that any recursive network topology of bounded degree, with flux $\Omega\left(n^{-\alpha}\right)$ and diameter $O\left(n^\beta\right)$, can support a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$with at most polylogarithmic inefficiency[b].

1.2. New Results

The main result of the paper is summarized in the following theorem.

Theorem 1 *For any value m upper bounded by a polynomial in n and for any $k \ge 0$ there exists a scheme to implement a shared memory of size m on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ with redundancy $\Theta\left(3^k\right)$ and slowdown*

$$O\left(2^k n^{\alpha + \frac{\alpha(1-\alpha)}{2^k(2-\alpha)}} + kn^\beta\right).$$

The scheme requires only weakly expanding graphs of constant degree and can be made fully constructive for $m = O\left(n^{3/2}\right)$ and $\alpha \ge 1/2$.

The following corollary is an immediate consequence of the theorem.

Corollary 1 *For any value m upper bounded by a polynomial in n there exists a scheme to implement a shared memory of size m on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ with optimal slowdown $O\left(n^\beta\right)$ and constant redundancy, when $\alpha < \beta$, and slowdown $O\left(n^\alpha \log n\right)$ and redundancy $O\left(\log^{1.59} n\right)$, when $\alpha \ge \beta$. The scheme requires only weakly expanding graphs of constant degree and can be made fully constructive for $m = O\left(n^{3/2}\right)$ and $\alpha \ge 1/2$.*

The importance of the above results is twofold. First, the proposed scheme is general and can be implemented on any machine supporting the D-BSP abstraction. Second, Corollary 1 shows, for the first time in the literature, that optimal worst-case slowdowns for shared memory access are achievable with constant redundancy for machines where latency overheads dominate over those due to bandwidth limitations. On the other hand, the lower bound proved in [13] shows that under reasonable assumptions, the performance of our scheme is not far from optimal for bandwidth-limited machines. Indeed, consider a machine that supports a

[b]Recall that the *flux* of a network $G = (V, E)$ is defined as the minimum among all subsets $U \subset V$, with $|U| \le |V|/2$, of $|C(U, V - U)|/|U|$, where $C(U, V - U)$ denotes the edge-cut induced by U [18].

D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ with $\alpha \geq \beta$, and assume that each subset of processors associated with an i-cluster is connected to the rest of the network by $O\left((n/2^i)^{1-\alpha}\right)$ links (note that such a machine cannot support a D-BSP $(n, \boldsymbol{g}^{(\alpha')}, \boldsymbol{\ell}^{(\beta)})$ with $\alpha' < \alpha$). The result in [13, Theorem 9] implies that the minimal slowdown achievable by any scheme like ours which dispatches an individual message for each copy to be accessed is $\Omega\left(n^{\alpha}(\log(m/n)/\log\log(m/n))^{1-\alpha}\right)$, and that in order to achieve such slowdown redundancy $\Omega\left(\log(m/n)/\log\log(m/n)\right)$ is necessary.

As mentioned before, our scheme builds upon the one presented in [15], whose design exploits the recursive decomposition of the underlying interconnection to provide a hierarchical, redundant representation of the shared memory based on $k+1$ levels of logical modules. More specifically, each variable is replicated into $r = O(1)$ copies, and the copies are assigned to r logical modules of level 0. In turn, the logical modules at the i-th level, $0 \leq i < k$ are replicated into three copies, which are assigned to three modules of level $i+1$. This process eventually creates $r3^k = \Theta\left(3^k\right)$ copies of each variable, and 3^{k-i} replicas of each module at level i. The number (resp., size) of the logical modules decreases (resp., increases) with the level number, and their replicas are assigned for storing to distinct subnetworks of appropriate size.

The key ingredients of the above memory organization are represented by the bipartite graph that governs the distribution of the copies of the variables among the modules of the first level, and those that govern the distribution of the replicas of the modules at the subsequent levels. The former graph is required to exhibit some weak expansion property and its existence can always be proved through combinatorial arguments although, for certain memory sizes, explicit constructions can be given. In contrast, all the other graphs employed in the scheme require expansion properties that can be obtained by suitable modifications of the BIBD graph [20], and can always be explicitly constructed. In fact, the choice of these latter graphs is where the memory organization adopted here mainly differs from the one presented in [15]. In particular, unlike [15], these graphs are not simply subgraphs of the BIBD, but their construction requires some nontrivial rearrangements to make them suitable for the clustered structure of the D-BSP while maintaining good expansion properties.

For an n-tuple of variables to be read/written, the selection of the copies to be accessed and the subsequent execution of the accesses of the selected copies are performed through suitable protocols, similar to the ones in [15], which can be implemented through a combination of prefix, sorting and routing primitives. We employ efficient D-BSP implementations for these primitives, including a novel optimal implementation of (k_1, k_2)-routing, to obtain the results stated above.

The rest of the paper is organized as follows. Section 2 presents the D-BSP primitives employed in our scheme. Section 3 describes the memory organization and the graphs involved. Section 4 describes the copy selection (Subsection 4.1) and the subsequent access to the selected copies (Subsection 4.2). Section 5 briefly discusses issues related to the constructivity of the scheme. Finally, Section 6 offers

some concluding remarks.

2. D-BSP Primitives

The scheme described in the following sections makes use of prefix, broadcast, sorting and routing primitives. The following propositions state the complexities of these primitives on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$. For ease of reference, the results are summarized in Table 1.

Table 1. Execution times on D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ of some common primitives.

Problem	**Execution Time**
Broadcast	$O\left(n^\alpha + n^\beta\right)$
Prefix computation	$O\left(n^\alpha + n^\beta\right)$
k-sorting	$O\left(kn^\alpha + n^\beta\right)$ if $k = \text{poly}(n)$
(k_1, k_2)-routing	$O\left(k_{\min}^\alpha k_{\max}^{1-\alpha} n^\alpha + n^\beta\right)$, where $k_{\min} = \min\{k_1, k_2\}$ and $k_{\max} = \max\{k_1, k_2\}$

Proposition 1 ([16]) *Any instance of single-item broadcast or parallel prefix of n items can be accomplished in time $O\left(n^\alpha + n^\beta\right)$ on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$.*

Let *k-sorting* denote a sorting problem in which k keys are initially assigned to each one of the n D-BSP processors and are to be redistributed so that the k smallest keys will be held by processor P_0, the next k smallest ones by processor P_1, and so on. We have:

Proposition 2 *Any instance of k-sorting, with k upper bounded by a polynomial in n, can be executed in time $T_{\text{sort}}(k, n) = O\left(kn^\alpha + n^\beta\right)$ on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$.*

Proof. Sort the k keys inside each processor sequentially, then simulate bitonic sorting on a hypercube [21] using the *merge-split* rather than the *compare-swap* operator, and mapping processors adjacent along the i-th dimension of the cube to D-BSP processors within the same $(\log n - i - 1)$-cluster, for $0 \leq i < \log n$. This yields an overall running time for k-sorting of

$$O\left(k \log k + \sum_{i=0}^{\log n - 1} (i+1)\left(kg_i + l_i\right)\right),$$

which simplifies to $O\left(kn^\alpha + n^\beta\right)$ if k is upper bounded by a polynomial in n. □

We call *(k_1, k_2)-routing* a routing problem where each processor is the source of at most k_1 packets and the destination of at most k_2 packets. Observe that any (k_1, k_2)-routing is a $\max\{k_1, k_2\}$-relation, hence the "greedy" routing strategy where all packets are delivered within the same superstep requires, in the worst case, $\max\{k_1, k_2\} \cdot n^\alpha + n^\beta$ time on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$. In the following proposition we show that a careful exploitation of the submachine locality exhibited by the model yields a better algorithm for (k_1, k_2)-routing.

Proposition 3 *Any instance of (k_1, k_2)-routing can be executed on a D-*

BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ *in time*

$$T_{\text{rout}}(k_1, k_2, n) = O\left(k_{\min}^{\alpha} k_{\max}^{1-\alpha} n^{\alpha} + n^{\beta}\right),$$

where $k_{\min} = \min\{k_1, k_2\}$ *and* $k_{\max} = \max\{k_1, k_2\}$.

Proof. We accomplish (k_1, k_2)-routing on a D-BSP in two phases as follows.

1. For $i = \log n - 1$ down to 0, in parallel within each $C_j^{(i)}$, $0 \leq j < 2^i$: evenly redistribute messages with origins in $C_j^{(i)}$ among the processors of the cluster.

2. For $i = 0$ to $\log n - 1$, in parallel within each $C_j^{(i)}$, $0 \leq j < 2^i$: send the messages destined to $C_{2j}^{(i+1)}$ to such cluster, so that they are evenly distributed among the processors of the cluster. Do the same for messages destined to $C_{2j+1}^{(i+1)}$.

Note that the above algorithm does not require that the values of k_1 and k_2 be known *a priori*. It is easy to see that at the end of iteration i of the first phase each processor holds at most $a_i = min\{k_1, k_2 2^i\}$ messages, while at the end of iteration i of the second phase, each message is in its destination $(i+1)$-cluster, and each processor holds at most $d_i = \min\{k_2, k_1 2^{i+1}\}$ messages. Note also that iteration i of the first (resp., second) phase can be implemented through a constant number of prefix operations and one routing of an a_i-relation (resp., d_i-relation) within i-clusters. Putting it all together, the running time of the above algorithm on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ is

$$O\left(\sum_{i=0}^{\log n - 1} \left(\max\{a_i, d_i\}\left(\frac{n}{2^i}\right)^{\alpha} + \left(\frac{n}{2^i}\right)^{\beta}\right)\right).$$

The theorem follows by plugging in the above formula the bounds for a_i and d_i derived before. □

It must be remarked that all the aforementioned primitives yield optimal algorithms when prominent interconnections (such as arrays) simulate a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ machine with matching bandwidth/latency parameters.

3. Memory Organization

The Hierarchical Memory Organization Scheme (HMOS) that governs the distribution of the copies of the m shared variables among the local memories of the n processors is a cascade of bipartite graphs obtained as a modification of the one introduced in [15]. A sample HMOS is depicted in Figure 1. In what follows we briefly recall how the HMOS is structured and point out the differences with the version in [15].

Let V denote the set of variables, and let U_i, $0 \leq i \leq k$, denote a set of nodes referred to as *i-modules*, where $k = O(\log \log n)$ is a suitable nonnegative integer that will be specified in the analysis. The U_i's can be regarded as nested collections

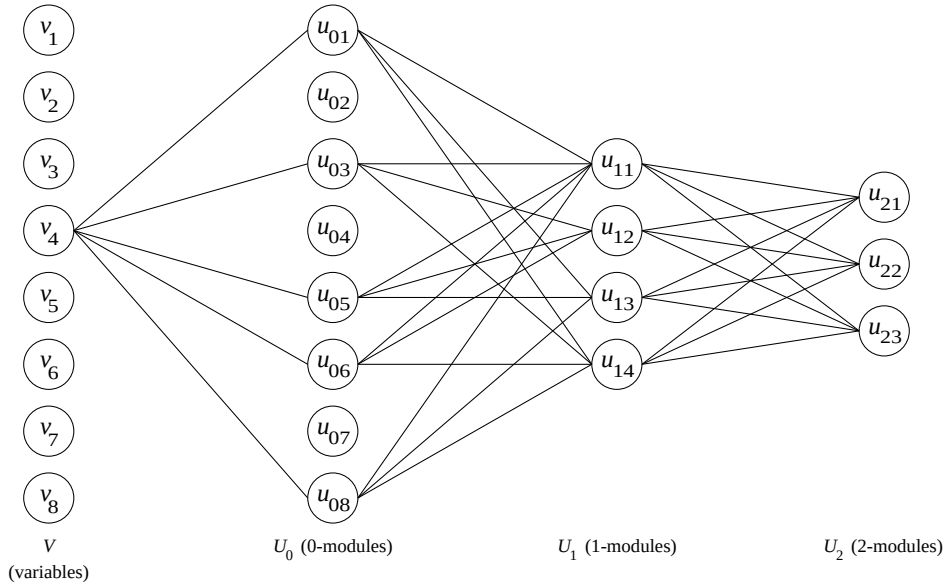

Fig. 1. A sample HMOS built upon $|V| = m = 8$ variables, with $k = 2$ levels of i-modules, $|U_0| = 8$ 0-modules, $|U_1| = 4$ 1-modules, $|U_2| = 3$ 2-modules and $r = 5$. To reduce the size of the HMOS, thus making it easier to read, the figure does not respect all the constraints on the parameters that are listed in Section 3. For the same reason, only the edges associated with the copies of variable v_4 are shown.

of variables, and are obtained as follows. First, each variable is replicated into $r = O(1)$ copies, r odd, which are assigned to distinct 0-modules. The contents of each 0-module, viewed as an indivisible unit, are in turn replicated into 3 copies, which are assigned to distinct 1-modules. In general, the contents of each $(i-1)$-module, viewed as an indivisible unit, are replicated into 3 copies, which are assigned to distinct i-modules, for $0 < i \leq k$. The above process will eventually create 3^{k-i} replicas of each i-module and $r3^k$ copies per variable. An example of the content of i-modules and of the whole memory is given by Figures 2 and 3. We will reserve the term *copy* to denote the replica of a variable, and *i-block* to denote the replica of an i-module. (Note that with this terminology k-modules and k-blocks coincide.) A k-module is composed of $(k-1)$-blocks, which in turn are composed of $(k-2)$-blocks, and so on. Finally, 0-blocks contain copies of variables.

The mapping between variables and 0-modules is represented by a bipartite graph[c] (V, U_0), where each variable $v \in V$ is adjacent to the r 0-modules whose 0-blocks hold the copies of v. Similarly, the mapping between $(i-1)$-modules and i-modules is represented by a bipartite graph (U_{i-1}, U_i), $1 \leq i \leq k$, where each $(i-1)$-module u is adjacent to the 3 i-modules whose i-blocks contain the $(i-1)$-blocks of u. We assume that $m = n^\tau$, for some constant $\tau > 1$. Let us fix

[c]For ease of presentation, we identify a bipartite graph by the sets of input and output nodes only.

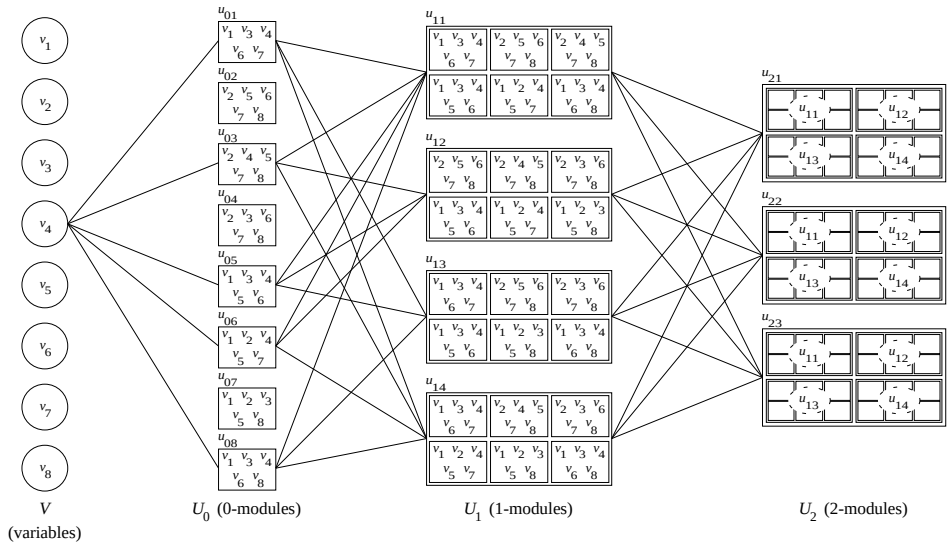

Fig. 2. Contents of i-modules, $0 \leq i \leq 2$, according to the HMOS of Figure 1. Variable v_4 is first replicated into $r = 5$ copies, which are assigned to 0-modules, then 3 copies of each such module are assigned to suitable 1-modules. Finally, each 1-module is replicated into 3 copies. As a whole, the process creates $5 \cdot 3^2 = 45$ copies of v_4.

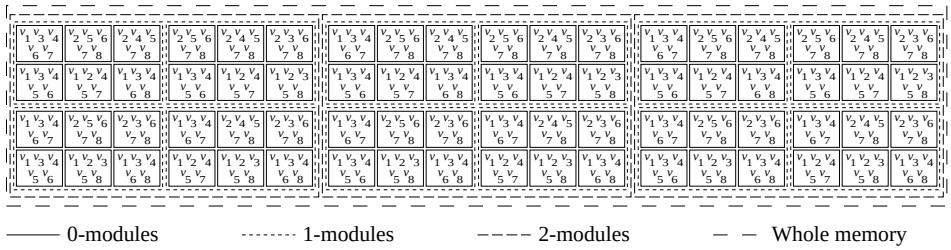

Fig. 3. Contents of the aggregate memory of the D-BSP according to the HMOS of Figure 1. The mapping process ensures that each i-block, $0 < i \leq k$, is recursively assigned to a distinct D-BSP cluster.

$m_0 = |U_0| = n$ and $m_i = |U_i| = \Theta\left(n^{1/2^i}\right)$, for $1 \leq i \leq k$. We assume that the quantity $d_i = \lceil 3m_{i-1}/m_i \rceil$ is a power of 2, and that every $u \in U_i$ has degree at most d_i in (U_{i-1}, U_i), for $1 \leq i \leq k$. Later, we will show how the various graphs can be chosen to satisfy all of the above constraints on the parameters, while exhibiting suitable expansion properties, which are needed to ensure low parallel access time.

The HMOS is mapped onto the D-BSP by assigning each i-block to a cluster of appropriate size, in the following recursive fashion. Each of the m_k k-blocks is assigned to a distinct cluster with $t_k = n/2^{\lceil \log m_k \rceil}$ processors. The (at most) d_k $(k-1)$-blocks contained in a k-block, as prescribed by (U_{k-1}, U_k), are assigned to distinct subclusters of size $t_{k-1} = t_k/d_k$ of the cluster assigned to the k-block. In general, for $2 \leq i \leq k$, the (at most) d_i $(i-1)$-blocks contained in an i-block are assigned to distinct subclusters of size $t_{i-1} = t_i/d_i$ of the cluster assigned to the i-block. Consequently, an i-block, $1 \leq i \leq k$, is mapped to a cluster with

$$t_i = \frac{n}{2^{\lceil \log_2 m_k \rceil} \prod_{j=i+1}^{k} d_j}$$

processors. Tedious but simple calculations show that

$$t_i = \Theta\left(3^{i-k} n^{1-1/2^i}\right), \qquad 1 \leq i \leq k. \tag{2}$$

Moreover, t_i is a power of 2 and, since $k = O(\log \log n)$, we have that $t_i > 1$ if n is large enough. Finally, the (at most) $d_1 = \Theta(\sqrt{n})$ 0-blocks contained in a 1-block are evenly distributed among the $t_1 = \Theta\left(\sqrt{n}/3^k\right)$ processors belonging to the cluster assigned to the 1-block, and each such processor stores the copies of the variables contained in every 0-block it receives. In this way, each processor stores a total of $O\left(r3^k m/n\right)$ copies of variables, which ensures a balanced distribution of the copies among the processors.

As mentioned before, suitable expansion properties are required of the component graphs of the HMOS in order to ensure that, for any set of variables to be accessed, their copies be well spread among the processors' memories, thus yielding low access time.

Definition 1 *Let $G = (X, Y)$ be a bipartite graph where each node in X has degree d. For $0 < \sigma \leq 1$, $0 < \epsilon < 1$ and $1 \leq \mu \leq d$, G has (σ, ϵ, μ)-expansion if for any subset $S \subseteq X$, $S \leq \sigma|X|$, and for any set E of $\mu|S|$ edges, μ outgoing edges for each node in S, the set $\Gamma^E(S) \subseteq Y$ reached by the chosen edges has size $|\Gamma^E(S)| = \Omega\left(|S|^{1-\epsilon}\right)$.*

We will choose (V, U_0) to have odd input degree r and output degree mr/n, and to exhibit $(n/m, \epsilon, (r+1)/2)$-expansion, where $\epsilon < 1$ is a suitable constant that will be chosen by the analysis. The existence of such a graph is proved in [15, Lemma 5.1] through the probabilistic method; however, explicit constructions are currently known for certain ranges of the parameters (see Section 5).

The main difference between the HMOS presented here and the one in [15] concerns the structure of the graphs (U_{i-1}, U_i), $1 \leq i \leq k$. Specifically, we need these graphs to exhibit similar expansion and constructivity properties as in [15],

but we are now forced to comply with different and stricter constraints on the parameters, which require nontrivial modifications to the construction presented in that paper. The following lemma provides the new crucial technical step through which the graphs are obtained.

Lemma 1 *For every positive integer w a bipartite graph $G = (X, Y)$ with $|X| = w$ can be explicitly constructed such that*

(i) $\sqrt{6w} < |Y| < 6 + 6\sqrt{6w}$*;*
(ii) Every node of X has degree 3;
(iii) Every node of Y has degree at most $d = \lceil 3w/|Y| \rceil$, and d is a power of 2;
(iv) For every subset $S \subset X$ whose nodes are all adjacent to some $y \in Y$ and for every selection of $\nu|S|$ edges, ν incident on every $s \in S$, the nodes of Y reached by the selected edges are at least $(\nu - 1)|S|/4 + 1$.

Proof. The base of our construction is the one presented in [15] for a $(z, 3)$-BIBD (Balanced Incomplete Block Design) [20], where z is any power of three. A $(z, 3)$-BIBD is a bipartite graph $G_{\text{BIBD}} = (A, B)$ with $z = |B|$ a power of 3, $|A| = z(z-1)/6$, and such that for any two nodes $b_1, b_2 \in B$ there is exactly one node $a \in A$ adjacent to both. Such graph has the following property: for every subset $S \subseteq A$ of nodes all adjacent to some node $y \in B$ and any selection of $\nu|S|$ edges, ν incident on every $s \in S$, the nodes of B reached by the selected edges are at least $(\nu-1)|S|+1$. We fix z to be the smallest power of 3 such that $|A| \geq w$. By employing the techniques presented in [15] we can extract a subgraph $G' = (X, B)$ of G_{BIBD} with $X \subseteq A$, $|X| = w$, and such that each node of X has degree 3 and each node of B has degree either $\lfloor 3w/z \rfloor$ or $\lceil 3w/z \rceil$.

Let d be the largest power of 2 not exceeding $\lfloor 3w/z \rfloor$, and let $B = \{b_i : 0 \leq i < z\}$. We index the edges of G' from 1 to $3w$ so that all edges incident on the same node b_i have consecutive indices, and, for $j < i$, edges incident on b_j have indices smaller than those incident on b_i. Observe that by the BIBD property, for every $0 < i < z$ there is at most one node $x \in X$ adjacent to both b_{i-1} and b_i. This allows us to rearrange the indices of the edges incident on b_i, for every i, so that if there are two edges (x, b_{i-1}) and (x, b_i), for some $x \in X$, the indices of these two edges differ by at least d. This is obtained by sequentially examining all pairs of nodes $\langle b_{i-1}, b_i \rangle$ starting from $\langle b_0, b_1 \rangle$: if there exists a pair of edges (x, b_{i-1}) and (x, b_i), for some $x \in X$, then the indexing is changed so that (x, b_{i-1}) is given the lowest index among the edges incident on b_{i-1}, and (x, b_i) is given the lowest index among the edges incident on b_i. Note that Step i of this procedure does not disrupt the work done in the previous steps: if for some $x' \in X$ there exists a pair of edges (x', b_{i-2}) and (x', b_{i-1}), which has been dealt with in Step $i-1$, then the distance between the indices of these edges can only increase as a consequence of Step i.

We construct $G = (X, Y)$ by grouping the edges of G' into $\lceil 3w/d \rceil$ bundles of d consecutively indexed edges each (the last bundle may have less than d edges), and by creating a distinct node of Y for each bundle, which becomes the new right endpoint for all edges in the bundle.

Note that Properties (ii) and (iii) stated in the lemma follow easily from the above construction. To show that Property (i) holds, observe that the choice of z ensures that $w \leq |A| < z^2/6$, whence $|Y| \geq |B| = z > \sqrt{6w}$. Similarly, the upper bound on $|Y|$ follows from the inequalities $w > (z/3-1)^2/6$ and $|Y| \leq 2|B|$. Finally, the construction of the bundles guarantees that the edges incident on every node $y \in Y$ were previously adjacent to at most 2 nodes of B, therefore for every pair of nodes in Y there are no more than four nodes in X adjacent to both. By applying this observation to the nodes in S (that are all adjacent to a single node $y \in Y$) it is straightforward to show that for each $y' \in Y$, $y' \neq y$, there are at most 4 nodes of S adjacent to y'. Let now $\Gamma \subseteq Y$ be the set of nodes reached by the selected edges: from the previous observations, Γ has cardinality at least $(\nu - 1)|S|/4 + 1$. □

Given U_0 we apply the above lemma to construct the graph (U_0, U_1), thus fixing the size of U_1. Once the size of U_1 is known, we apply the lemma again to construct the graph (U_1, U_2), thus fixing the size of U_2. By iterating this process we can construct every graph (U_{i-1}, U_i), $1 \leq i \leq k$. By Property (i) of Lemma 1 such a process yields $m_i = |U_i| = \Theta\left(n^{1/2^i}\right)$, for $1 \leq i \leq k$, as required.

4. Access Protocol

Suppose that m shared variables are distributed among the n D-BSP processors according to the HMOS described above. In this section, we show how any n-tuple of variables can be efficiently accessed when every processor requires read or write access to a distinct variable. Note that the case of concurrent accesses to the same variable can be reduced to the case of exclusive accesses by means of straightforward, sorting-based techniques which do not affect the overall running time. The access protocol has the same overall structure as the one presented in [15], but it differs in the implementation which fully exploits the explicit hierarchical nature of the machine model.

Let S denote the set of variables to be accessed, with every processor in charge of a distinct variable of S. As customary in redundant shared memory implementation schemes, a suitable set of copies for the variables in S must be chosen, so that accessing these copies will enforce data consistency and generate low memory contention and network congestion. We first recall how copy selection is accomplished and then show how the selected copies can be efficiently reached.

4.1. Copy Selection

The hierarchical structure of the HMOS provides a geographical distribution of the copies into the D-BSP clusters. Copy selection essentially aims at limiting the number of copies that have to be accessed in any block at any level of the HMOS, in order to reduce the traffic into/from the cluster storing the block. Consider a directed version $\mathcal{H}$ of the HMOS, where edges in every constituent bipartite graph are directed from the left to the right node set. Note that $\mathcal{H}$ is a dag and that the $r3^k$ copies of a variable v are in one-to-one correspondence with the source-sink

paths of the subdag $\mathcal{H}_v \subset \mathcal{H}$ induced by v and by all of its descendants. In order to guarantee consistency, we require that the selected copies for every variable $v \in S$ form a *target set*, defined as follows. A set of copies C_v of v is a target set for v if, by coloring the nodes in $\mathcal{H}_v$ along the paths corresponding to the copies in C_v, it turns out that each colored node has a majority of its children colored. Specifically, $\mu = (r+1)/2$ children of the root v and two children of every colored node $x \neq v$ must be colored. It is easy to see that any two target sets for v share at least one copy. Consistency is enforced since a read access will always reach at least one most updated copy, which can be identified by means of timestamps [8].

Copy selection for the set of variables S is accomplished in $k+1$ iterations, numbered from 0 to k, during which in every $\mathcal{H}_v$, $v \in S$, the nodes along the paths corresponding to the copies being selected are colored level by level from the source to the sinks. Since, in general, a node of the HMOS belongs to several $\mathcal{H}_v$'s, we use a distinct color γ_v for each variable $v \in S$, and allow a node to be assigned a set of colors. Initially, the color set of every node is empty. Iteration 0 assigns the set $\{\gamma_v\}$ to the source v and adds γ_v to the color sets of μ of its adjacent 0-modules, for every $v \in S$. Iteration i, $0 < i \leq k$, selects two adjacent i-modules for every colored $(i-1)$-module u, and adds to their color sets the one assigned to u. (Clearly, if an i-module is selected for two or more colored $(i-1)$-modules, it receives the union of the color sets of these $(i-1)$-modules.) In this fashion, for every $v \in S$, at the end of Iteration i the nodes whose color sets contain γ_v form exactly $\mu 2^i$ distinct paths from the source to nodes at level i in $\mathcal{H}_v$. We call such paths γ_v*-colored paths*. We choose C_v as the set of copies of v corresponding to the $\mu 2^k$ γ_v-colored paths in $\mathcal{H}_v$ at the end of the last iteration. In order to identify the selected copies of each $v \in S$, the processor issuing the access request for v keeps track of the coloring being performed on $\mathcal{H}_v$.

We define the *weight* $w(u)$ of an i-module $u \in U_i$, $0 \leq i \leq k$, as the sum, over all $v \in S$, of the number of γ_v-colored paths containing u. The above coloring ensures that for every $u \in U_i$ with nonempty color set, only 2^{k-i} i-blocks of u contain copies in $\bigcup_{v \in S} C_v$, with exactly $w(u)$ copies per block. Using the expansion properties of the HMOS, we are able to establish suitably low bounds for the $w(u)$'s. The following lemmas provide such bounds and evaluate the running times of the iterations on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$.

Lemma 2 *If (V, U_0) has $(n/m, \epsilon, \mu)$-expansion, Iteration 0 can be executed in time $O\left(n^\alpha + n^\beta\right)$, in such a way that, for every $u \in U_0$*

$$w(u) = O\left(n^\epsilon\right).$$

Proof. At the beginning of the iteration, every processor P_v in charge of a variable $v \in S$ creates r packets of type $[P_v, v, u]$, where u is the name of a distinct 0-module adjacent to v in $\mathcal{H}_v$. Upon creation, all packets are regarded as *unmarked*. Then, the following steps are executed until μ packets for each variable are marked and all other packets are destroyed.

1. (*Sort*) Sort all unmarked packets by their third component (i.e., the associated

0-module).

2. (*Select*) Let a be a suitable constant. For each $u \in U_0$, if there are at most arn^ϵ packets with third component u in the sorted sequence, then all such packets are marked. Otherwise, none of them is marked. Subsequently, all the packets are sent back to their origins.

3. (*Count*) For each $v \in S$, the total number of marked packets relative to v are counted. If these are at least μ, then exactly μ of them are kept and the remaining $\mu - 1$ (either marked or unmarked) are destroyed.

Finally, for every $v \in S$ the nodes in $\mathcal{H}_v$ corresponding to the μ marked packets picked for v receive color γ_v. Based on the expansion of (V, U_0) it can be shown [15, Lemma 4.4] that at the end of the whole procedure the bound on $w(u)$ holds for every $u \in U_0$; moreover, after the ith execution of the Sort, Select and Count steps, packets relative to at most $n/2^i$ variables remain unmarked. Hence, the ith execution of these three steps can be implemented through r-sorting and prefix within an $(i-1)$-cluster, which yields the desired running time by applying the results of Propositions 1 and 2. □

Lemma 3 *For $1 \leq i \leq k$, Iteration i can be executed in time $O\left(2^i n^\alpha + n^\beta\right)$ in such a way that, for every $u \in U_i$*

$$w(u) = O\left(2^i n^{1-(1-\epsilon)/2^i}\right).$$

Proof. We can adopt the same implementation of Iteration i as in [15], which essentially requires a constant number of $O\left(2^i\right)$-sorting and prefix operations. The running time follows from Propositions 1 and 2, while the bound on $w(u)$ is obtained by repeating the argument in [15] and tuning the constants to reflect the different expansion property of the graphs defined in Lemma 1 with respect to the ones adopted in that paper. □

The following theorem is an immediate consequence of the above lemmas and the preceding discussion.

Theorem 2 *Copy selection requires time $O\left(2^k n^\alpha + kn^\beta\right)$ and ensures that every i-block, $0 \leq i \leq k$, contains at most $O\left(2^i n^{1-(1-\epsilon)/2^i}\right)$ selected copies.*

4.2. Access to the Selected Copies

After copy selection is completed, for every $v \in S$ the processor in charge of v creates $\mu 2^k$ distinct messages to access the copies in C_v. Each message is routed to its destination (i.e., the processor whose memory stores the requested copy) where the read/write access is performed[d]. Messages are delivered to the destinations in $k+1$ stages through smaller and smaller clusters, thus taking advantage of the good distribution of the copies among the i-blocks, for every i.

[d] For the case of read accesses, the return of the accessed data to the requesting processors is dealt with in a symmetric fashion, hence we omit its description.

For $1 \leq i \leq k$, let $\tau_i = \log n - \log t_i$ and recall that every i-block is assigned to a distinct cluster with t_i processors, that is, a τ_i-cluster. Let also $\tau_{k+1} = 0$. The stages are numbered from $k+1$ down to 1. For $k+1 \geq i \geq 2$, Stage i is executed in parallel and independently in every τ_i-cluster and sends all messages residing in the cluster to arbitrary positions in the internal τ_{i-1}-clusters associated with their destination $(i-1)$-blocks, in such a way that the processors in the same τ_{i-1}-cluster receive approximately the same number of messages. The intermediate destination of each message is easily established via sorting and ranking. Finally, in Stage 1 every message is sent to its final destination, in parallel and independently within every τ_1-cluster.

Let δ_i be the maximum number of messages held by any processor at the beginning of Stage i, $k+1 \geq i \geq 1$. By virtue of the message balancing described above, δ_i is upper bounded by the maximum number of selected copies within a single i-block divided by the size of a τ_i-cluster. Moreover, at the end of Stage 1, $O(n^\epsilon)$ messages per 0-block reach the processor storing the block in its local memory. Since the number of 0-blocks assigned to a single processor is $\Theta\left(3^k\right)$, the maximum number of messages delivered to a processor at the end of the access protocol is $\delta_0 = O\left(3^k n^\epsilon\right)$. By plugging in the bounds derived in Theorem 2 and Equation 2 we obtain:

$$\delta_i = \begin{cases} \mu 2^k & \text{for } i = k+1, \\ O\left(2^i 3^{k-i} n^{\epsilon/2^i}\right) & \text{for } k \geq i \geq 0. \end{cases}$$

Stage i can be easily implemented by means of a constant number of δ_i-sorting and prefix operations and one instance of (δ_i, δ_{i-1})-routing within τ_i-clusters. From Propositions 1, 2 and 3 it then follows that the time required to access the selected copies is

$$O\left(\left(2^k n^{\frac{(1-\alpha)\epsilon}{2^k}} + \sum_{i=1}^{k} 2^i 3^{(1-\alpha)(k-i)} n^{\frac{(2-\alpha)\epsilon-\alpha}{2^i}}\right) n^\alpha + k n^\beta\right). \tag{3}$$

Note that the above time always dominates over the time for copy selection. Let us now choose the expansion parameter ϵ of (V, U_0) to be strictly less than $\alpha/(2-\alpha)$. Simple manipulations of Equation 3 then suffice to prove the following theorem on the overall slowdown of our simulation scheme.

Theorem 3 *Any n-tuple of variables can be accessed by the D-BSP processors in time*

$$O\left(2^k n^{\alpha + \frac{\alpha(1-\alpha)}{2^k(2-\alpha)}} + k n^\beta\right).$$

In order to achieve the best possible tradeoff between performance and redundancy on a specific D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$, we need to choose a suitable value for k as a function of α and β. More specifically, when $\alpha < \beta$, it suffices to choose $k = O(1)$ large enough to obtain optimal $O(n^\beta)$ slowdown with constant redundancy. This result demonstrates that optimal worst-case slowdown is achievable on machines where delays due to latency dominate over those due to bandwidth. Instead, when

$\alpha \geq \beta$, by choosing $k = \log\log n$, we obtain a minimal slowdown of $O\left(n^{\alpha} \log n\right)$ with redundancy $\Theta\left(\log^{\log_2 3} n\right) = O\left(\log^{1.59} n\right)$.

5. Constructivity Issues

It must be remarked that the performance of the access protocol analyzed in the previous section relies on the $(n/m, \epsilon, \mu)$-expansion of (V, U_0). Although such level of expansion is considerably milder than those required by most schemes in the literature (e.g., $\epsilon = O\left(1/\log n\right)$ in [8, 9, 13]), in general we can only show the existence of (V, U_0) through the probabilistic method. However, unlike the aforementioned schemes, ours can take advantage of the few explicit constructions known in the literature. For example, in [14] an efficient construction is provided for a bipartite graph with m inputs, $n' = \Theta\left(m^{2/3}\right)$ outputs, input degree $r = 3$, and output degree $\Theta\left(m^{1/3}\right)$, which exhibits $(1, 1/3, 2)$-expansion. This graph can be employed to implement (V, U_0) for every value $m = O\left(n^{3/2}\right)$. Note that for $m = o\left(n^{3/2}\right)$, the number n' of output nodes in the graph is $o(n)$. In this case, in order to obtain $|U_0| = n$, we simply replace each output node by n/n' new outputs, subdividing the incoming edges evenly among these new nodes. Clearly, the expansion property is not affected by this modification. Also it is easy to see that by plugging $\epsilon = 1/3$ and $\alpha \geq 1/2$ in Equation 3, the overall running time of the access protocol reduces to the one stated in Theorem 3, thus yielding the constructivity result of Theorem 1.

6. Conclusions

We have presented a deterministic PRAM simulation scheme for the Decomposable BSP (D-BSP), a machine abstraction belonging to the class of bandwidth/latency models. As argued in [19], D-BSP is an effective model for a wide class of distributed-memory machines, in that its parameters are a succinct yet sufficiently descriptive summary of the bandwidth/latency characteristics of a typical networked architecture, and affords the exploitation of submachine locality without giving up generality. Indeed, D-BSP can be efficiently supported on several prominent architectures, including multidimensional arrays and the hypercube [16, 19].

The scheme presented in this paper generalizes the one in [15], specifically tailored to the mesh topology, by making nontrivial modifications to the memory organization and by expressing the access protocol in terms of a few general primitives, so that it can adapt to the hierarchical structure of different architectures. More specifically, its novel features mainly include new expanding component graphs of the HMOS, required to fit the clustered structure of the D-BSP model, and a general routing algorithm for unbalanced communication which does not exploit the fine details of the interconnection. Our scheme can be immediately ported to any architecture supporting D-BSP.

For the sake of concreteness, in the paper we have analyzed the slowdown for a spectrum of machine parameters, showing that close to optimal performance can be

obtained for machines characterized by moderate bandwidth (e.g., multidimensional arrays). As a corollary, the instantiation of the scheme on the mesh yields the results in [15]. Moreover, we have shown that optimality can be achieved on machines where delays due to latency dominate over those due to bandwidth limitations. In fact, none of the schemes previously developed in the literature for specific networks had been able to uncover this fact.

Common to all previous works on deterministic PRAM simulation, a challenging open problem is the explicit construction of expanding bipartite graphs that could make the scheme fully constructive for any number m of shared variables. As argued in the paper, explicit constructions are known only to deal with the case $m = O(n^{3/2})$, which however is sufficient to simulate any NC algorithm [1].

Acknowledgements

The authors wish to thank the anonymous referees for their comments that helped improve the presentation of the paper. This research was supported in part by MIUR of Italy under project *ALINWEB: Algorithmics for the Internet and the Web*.

References

1. J. JáJá, *An Introduction to Parallel Algorithms* (Addison Wesley, Reading, MA, 1992).
2. A. Czumaj, F. Meyer auf der Heide and V. Stemann, "Contention resolution in hashing based shared memory simulations," *SIAM J. Comput.* **29(5)** (2000) 1703–1739.
3. A. G. Ranade, "How to emulate shared memory," *J. Comput. System Sci.* **42** (1991) 307–326.
4. F. T. Leighton, B. M. Maggs, A. G. Ranade and S. Rao, "Randomized routing and sorting on fixed-connection networks," *J. Algorithms* **17** (1994) 157–205.
5. L. G. Valiant, "General purpose parallel architectures," *Handbook Computer Sci.*, ed. J. van Leeuwen (North Holland, 1990) pp. 943–972.
6. L. A. Goldberg, Y. Matias and S. Rao, "An optical simulation of shared memory," *SIAM J. Comput.* **28(5)** (1999) 1829–1847.
7. K. Mehlhorn and U. Vishkin, "Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories," *Acta Informatica* **21** (1984) 339–374.
8. E. Upfal and A. Widgerson, "How to share memory in a distributed system," *J. ACM* **34** (1987) 116–127.
9. H. Alt, T. Hagerup, K. Mehlhorn and F. P. Preparata, "Deterministic simulation of idealized parallel computers on more realistic ones," *SIAM J. Comput.* **16(5)** (1987) 808–835.
10. F. Luccio, A. Pietracaprina and G. Pucci, "A new scheme for the deterministic simulation of PRAM in VLSI," *Algorithmica* **5** (1990) 529–544.
11. K. T. Herley and G. Bilardi, "Deterministic simulations of PRAMs on bounded-degree networks," *SIAM J. Comput.* **23(2)** (1994) 276–292.

12. K. T. Herley, "Representing shared data in distributed-memory parallel computers," *Math. Syst. Theory* **29(2)** (1996) 111–156.
13. K. T. Herley, A. Pietracaprina and G. Pucci, "Implementing shared memory on mesh-connected computers and on the fat-tree," *Information and Computation* **165(2)** (2001) 123–143.
14. A. Pietracaprina and F. P. Preparata, "Practical Constructive Schemes for Deterministic Shared-Memory Access," *Theory Comp. Systems*, **33** (1997) 3–37.
15. A. Pietracaprina, G. Pucci and J. Sibeyn, "Constructive, deterministic implementation of shared memory on meshes," *SIAM J. Comput.* **30(2)** (2000) 625–648.
16. P. De la Torre and C. P. Kruskal, "Submachine locality in the bulk synchronous setting," *Proc. Euro-Par 1996*, Lyon, France, Aug. 1996, LNCS 1124, pp. 352–358.
17. L. G. Valiant, "A bridging model for parallel computation," *Comm. ACM* **33** (1990) 103–111.
18. F. T. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms," *J. ACM* **46** (1999) 787–832.
19. G. Bilardi, A. Pietracaprina and G. Pucci, "A quantitative measure of portability with application to bandwidth-latency models for parallel computing," *Proc. Euro-Par 1999*, Toulouse, France, Aug.–Sep. 1999, LNCS 1685, pp. 543–551.
20. M. Hall Jr., *Combinatorial Theory, Second Edition* (John Wiley & Sons, New York, NY, 1986).
21. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes* (Morgan Kaufmann, San Mateo, CA, 1992).
22. F. T. Leighton, B. M. Maggs and A. W. Richa, "Fast algorithms for finding O(congestion + dilation) packet routing schedules," *Combinatorica* **19** (1999) 375–401.