CONSTRUCTIVE, DETERMINISTIC IMPLEMENTATION OF SHARED MEMORY ON MESHES*

ANDREA PIETRACAPRINA[†], GEPPINO PUCCI[†], AND JOP F. SIBEYN[‡]

Abstract. This paper describes a scheme to implement a shared address space of size m on an n-node mesh, with m polynomial in n, where each mesh node hosts a processor and a memory module. At the core of the simulation is a Hierarchical Memory Organization Scheme (HMOS), which governs the distribution of the shared variables, each replicated into multiple copies, among the memory modules, through a cascade of bipartite graphs. Based on the expansion properties of such graphs, we devise a protocol that accesses any n-tuple of shared variables in worst-case time $O\left(n^{1/2+\eta}\right)$, for any constant $\eta > 0$, using $O\left(1/\eta^{1.59}\right)$ copies per variable, or in worst-case time $O\left(n^{1/2}\log n\right)$, using $O\left(\log^{1.59} n\right)$ copies per variable. In both cases the access time is close to the natural $O\left(\sqrt{n}\right)$ lower bound imposed by the network diameter. A key feature of the scheme is that it can be made fully constructive when m is not too large, thus providing in this case the first efficient, constructive, deterministic scheme in the literature for bounded-degree processor networks. For larger memory sizes, the scheme relies solely on a nonconstructive graph of weak expansion. Finally, the scheme can be efficiently ported to other architectures, as long as they exhibit certain structural properties. In the paper we discuss the porting to multi-dimensional meshes and to the pruned butterfly, an area-universal network which is variant of the fat-tree.

Key words. PRAM simulation, parallel computation, shared memory machines, networks of processors, meshes, expander graphs

AMS subject classification. 68Q10

1. Introduction.

A desirable feature of a parallel computer is the provision of a *shared address space* that can be accessed concurrently by all the processors of the machine. Indeed, the manipulation of shared data provides a powerful and uniform mechanism for interprocessor communication, and constitutes a valuable tool for the development of simple and portable parallel software. Unfortunately, when the number of processors exceeds a certain (modest) threshold, any efficient hardware realization of shared memory is either prohibitively expensive or out of reach of current technology. Therefore, a shared address space must be provided *virtually* on hardware platforms consisting of a set of processor/memory module pairs which are connected through a network of point-to-point communication links.

This problem has received considerable attention over the past two decades, and has been the target of a large number of investigations, both theoretical and applied. In the theoretical community, the problem is best known as the *PRAM simulation* problem. An (n, m)-PRAM is an abstraction of a shared-memory machine consisting of n synchronous RAM processors that have direct access to m shared *variables*. In a PRAM step, executed in unit time, any set of n variables can be read or written in parallel by the processors. A solution to the PRAM simulation problem is a scheme to perform *any* computation of an (n, m)-PRAM on a target machine consisting of a

*This research was supported in part, through the Leonardo Fibonacci Institute, by the Istituto Trentino di Cultura. The results in this paper appeared in preliminary conference form in [PPS94] and in [PP95].

[†]Dipartimento di Elettronica e Informatica, Università di Padova, Via Gradenigo 6/a, 35131 Padova, Italy. Email: {andrea,geppo}@artemide.dei.unipd.it. The research of these authors was supported in part by CNR and MURST of Italy.

[‡]Max-Planck Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. Email: jopsi@mpi-sb.mpg.de. The research of this author was supported in part the EC Cooperative Action IC-1000 (project ALTEC: *Algorithms for Future Technologies*).

network of n processor/memory pairs. A typical PRAM simulation scheme distributes the PRAM shared variables among the n modules local to the machine processors, and recasts a parallel access to the shared-memory into the routing of messages from the processors requesting the variables to the processors storing such variables.

Several randomized PRAM simulation schemes have been proposed in the literature. In all these schemes, the shared variables are distributed among the memory modules via one (or more) hash functions randomly drawn from a suitable universal class. Among the most relevant results, we recall that a PRAM step can be simulated, with high probability, in $O(\log\log\log n \log^* n)$ time on the complete network [CMS95], in $O(\log n)$ time on the butterfly [Ran91] and in $O(\sqrt{n})$ time on the mesh [LMRR94].

In contrast, the development of efficient deterministic schemes, that is, schemes that guarantee a fast worst-case simulation time for any PRAM step, appears to be much harder. A simple argument shows that in order to avoid trivial worst-case scenarios, where all the variables requested in the PRAM step are stored in a small region of the network, one has to use several copies for each variable, so that only a subset of "convenient" copies needs to be reached by each operation. The number of copies used for each variable is called the *redundancy* of the scheme.

The idea of replicating each variable into multiple copies dates back to the pioneering work of Mehlhorn and Vishkin [MV84]. In their approach, a read operation need only access one (the most convenient) copy. For $m = O\left(n^R\right)$, the authors obtain a scheme for the complete interconnection which uses R copies per variable and allows any set of n reads to be satisfied in time $O\left(n^{1-1/R}\right)$. However, the execution of n write operations, where all copies of the variables must be accessed, is penalized and requires $O(Rn)$ time in the worst case.

Later, Upfal and Wigderson [UW87] 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 by means of a *Memory Organization Scheme* (MOS). An MOS is a bipartite graph $G = (V, U)$, where V is the set of shared variables, U is the set of memory modules of the underlying machine, and R edges connect each variable to the modules storing its copies. For m polynomial in n and $R = \Theta(\log n)$, the authors show that there exist suitable expanding graphs that guarantee a worst-case $O\left(\log n (\log\log n)^2\right)$ time to access any n variables on the complete interconnection. This bound was later improved to $O(\log n)$ in [AHMP87]. Several authors pursued the ideas in [UW87] to develop simulation schemes for *bounded-degree networks* of various topologies. In particular, schemes have been devised to simulate an arbitrary step of an (n, m)-PRAM, with m polynomial in n, in time $O\left(\log^2 n / \log\log n\right)$ on a Mesh-of-Trees (MoT) with n processors and $\Theta\left(n^2\right)$ switching elements [LPP90], or in time $O(\log n \log m / \log\log n)$ on an n-processor expander-based network [HB94], or in time $O(\log n \log\log n \log\log(m/n))$ on a suitably augmented MoT [Her96].

All of the aforementioned deterministic schemes (except for the one in [MV84] which, however, is not general since write accesses are heavily penalized) suffer from two major limitations.

1. The MOS graphs must exhibit maximum expansion relatively to the m/n ratio. Although the existence of such graphs can be proved through standard counting arguments, no efficient constructions are yet available. In addition, it is unlikely that the (few) constructions known for expanders may be of use when m is much larger than n.

2. The expansion properties of the MOS are exclusively used to curb memory contention. Network congestion issues are either ignored, as in the case of simulations on the complete network, or solved by means of separate mechanisms tailored to the specific network's topology[1].

Recently, constructive deterministic schemes exhibiting nontrivial performance have been developed for the complete interconnection. In [PP97] three schemes are presented for $m = O\left(n^{3/2}\right)$, $m = O\left(n^2\right)$ and $m = O\left(n^3\right)$ variables, which attain $O\left(n^{1/3}\right)$, $O\left(n^{1/2}\right)$ and $O\left(n^{2/3}\right)$ access time, respectively, for any n-tuple of variables using constant redundancy. These schemes rely on MOS graphs that admit efficient explicit constructions but exhibit weak expansion. In this paper we will exploit the same constructions in a more complex framework to achieve efficient implementations of shared data on realistic, low-bandwidth machines. Specifically, we will develop a novel approach where the inefficiencies caused by the weak expansion of the memory map are absorbed into the inherent bandwidth limitations of the interconnection, and where both memory contention and network congestion are controlled through a single mechanism.

1.1. Overview of Results.

This paper presents a deterministic scheme for implementing a shared address space of size m on an n-node square mesh, with m polynomial in n, where each node consists of a processor with direct access to a local memory module. The scheme provides a protocol to access an arbitrary set of n shared variables in nearly-optimal time, for all values of m. The scheme is fully constructive for $m = O\left(n^{3/2}\right)$, whereas for larger values of m it embodies only a nonconstructive component graph of constant degree whose expansion properties, however, are much weaker than those required of the graphs used in previous works. Full constructiveness can also be attained for memory sizes up to $m = O\left(n^{9/2}\right)$, at the expense of a progressive degradation in performance when m gets closer to the upper bound.

The scheme adopts a novel redundant representation of the shared variables and is centered around the *Hierarchical Memory Organization Scheme* (HMOS), which provides a structured distribution of the copies of the variables among the memory modules. The HMOS consists of $k + 1$ levels of logical modules built upon the set of shared variables. The modules of the first level (level 0) store copies of variables, whereas modules of level $i > 0$ store replicas of modules of level $i - 1$. The HMOS is represented by a cascade of bipartite graphs, where the first graph governs the distribution of the copies of the variables to the modules of level 0, and the other graphs govern the distribution of the replicas of modules at higher levels. Each level of the HMOS corresponds to a tessellation of the mesh into submeshes of appropriate size, with each module of that level assigned to a distinct submesh.

We devise an access protocol to satisfy n arbitrary read/write requests issued by the n processors, which takes advantage of the hierarchical structure of the HMOS. As customary in any multi-copy approach, an access to a variable is executed on a selected subset of its copies. A suitable copy selection mechanism is developed to limit the number of copies to be accessed in each submesh, and, ultimately, in each individual module. In this sense, the HMOS provides a single mechanism to cope with both memory contention and network congestion, which represents a novelty with respect to previous works, where the two issues were dealt with separately.

[1]In fact, in [HB94] an MOS with slightly less than maximum expansion is employed in order to reduce the redundancy and, consequently, network congestion, at the expense of an increase in memory contention. However, such an MOS does not embody any specific mechanism to explicitly control network congestion.

TABLE 1
Results in this paper.

$m = n^\tau$	Access Time	Constructive	Redundancy
$\tau > 3/2$	$O\left(n^{\frac{1}{2}+\eta}\right)$, $\forall$ constant $\eta > 0$	no	$O\left(1/\eta^{1.59}\right)$
$\tau > 3/2$	$O\left(\sqrt{n}\log n\right)$	no	$O\left(\log^{1.59} n\right)$
$1 \le \tau \le 3/2$	$O\left(n^{\frac{1}{2}+\eta}\right)$, $\forall$ constant $\eta > 0$	yes	$O\left(1/\eta^{1.59}\right)$
$1 \le \tau \le 3/2$	$O\left(\sqrt{n}\log n\right)$	yes	$O\left(\log^{1.59} n\right)$
$\frac{3}{2} < \tau \le \frac{13}{6}$	$O\left(n^{\frac{2\tau+1}{8}}\right)$	yes	$\Theta(1)$
$\frac{13}{6} \le \tau \le \frac{5}{2}$	$O\left(n^{\frac{2}{3}}\right)$	yes	$\Theta(1)$
$\frac{5}{2} \le \tau \le \frac{9}{2}$	$O\left(n^{\frac{2\tau+3}{12}}\right)$	yes	$\Theta(1)$

In order to guarantee low memory contention and network congestion, the HMOS component graphs must exhibit certain expansion properties. Compared to those employed in previous schemes, our graphs have much weaker expansion, attainable using only constant (rather than logarithmic) input degree. This makes the HMOS more amenable to explicit construction. Indeed, all HMOS graphs but the first one are taken as subgraphs of a well-known combinatorial structure, the BIBD, for which an explicit and simple construction is known. As for the first graph, an explicit construction can be provided when m is not too large, thus making the HMOS fully constructive, while for large values of m, the graph can be shown to exist through a standard counting argument. Our results are reported in detail below, and summarized in Table 1.

THEOREM 1. *For any constant $\tau \ge 1$, there exists a scheme to distribute $m = n^\tau$ shared variables among the local memory modules of an n-node mesh with redundancy R so that any n variables can be read/written in time*

$$T = O\left(n^{\frac{1}{2}+\eta}\right)$$

for any constant $\eta > 0$, with $R = O\left(1/\eta^{1.59}\right)$; or in time

$$T = O\left(n^{\frac{1}{2}}\log n\right)$$

with $R = O\left(\log^{1.59} n\right)$.

As mentioned before, for arbitrary values of m the HMOS embodies one non-constructive graph. Full constructiveness can be achieved when $m = O\left(n^{9/2}\right)$, as reported below.

THEOREM 2. *For any constant τ, with $1 \le \tau < 9/2$, there exists a fully constructive scheme to distribute $m = n^\tau$ shared variables among the local memory modules of an n-node mesh, which, for $\tau \le 3/2$, achieves the same performances as those stated in Theorem 1, while for $\tau > 3/2$ achieves the following access times*

$$\begin{aligned} T &= O\left(n^{\frac{2\tau+1}{8}}\right) && \text{for } \tfrac{3}{2} < \tau \le \tfrac{13}{6}, \\ T &= O\left(n^{\frac{2}{3}}\right) && \text{for } \tfrac{13}{6} \le \tau \le \tfrac{5}{2}, \\ T &= O\left(n^{\frac{2\tau+3}{12}}\right) && \text{for } \tfrac{5}{2} \le \tau \le 9/2, \end{aligned}$$

with redundancy $R = \Theta(1)$.

Compared to the natural $\Omega(\sqrt{n})$ lower bound imposed by the network diameter, our fastest access time is only a logarithmic factor away from optimal.

Prior to the present work, no efficient deterministic schemes for implementing shared memory explicitly designed for the mesh topology were known in the literature. However, the schemes designed for the complete interconnection can be implemented on the mesh through sorting and routing. In particular, $O(\sqrt{n}\log n)$ access time can be obtained by implementing the nonconstructive scheme in [AHMP87], and $O\left(n^{1/2+t}\right)$ access time can be obtained by porting the constructive schemes of [PP97] to the mesh, with $t = 1/6$ for $m = O\left(n^{3/2}\right)$, $t = 1/4$ for $m = O\left(n^2\right)$ and $t = 1/3$ for $m = O\left(n^3\right)$.

Our results improve upon previously published ones in the following ways. First, we attain $O(\sqrt{n}\log n)$ access time, as in [AHMP87], with a memory organization which is fully constructive when $m = O\left(n^{3/2}\right)$, while, for all other values of m polynomial in n, it embodies a nonconstructive graph exhibiting much weaker expansion than that required in [AHMP87]. The recent results of [PP97] suggest that our memory map is more amenable to explicit construction. Note also that our constructive results outperform the ones obtainable by the straightforward porting to the mesh of the schemes in [PP97] discussed above.

Finally, it is important to observe that our scheme is not specifically tailored to the mesh topology, but can be ported, with minor adjustments, to other topologies. In particular, the same access times reported in Theorems 1 and 2 can be attained on an n-leaf pruned butterfly, an area-universal network which is a variant of the fat-tree, and Theorem 1 can be generalized to hold for d-dimensional meshes, with constant d, by substituting $n^{1/d}$ for $n^{1/2}$ in the formulas.

The rest of this paper is organized as follows. Section 2 defines the machine model and introduces the routing and sorting primitives used by the access protocol. Section 3 describes the HMOS (§3.1) and its implementation on the mesh (§3.2). A suitable construction for the BIBDs used in the HMOS is given in an appendix to the paper. Section 4 presents the protocol for accessing an arbitrary n-tuple of shared variables. This section is subdivided into two subsections that describe the selection of the copies and the routing protocol, respectively. In §5, suitable values for the design parameters of the HMOS are selected, and Theorems 1 and 2 are proved. Section 6 shows how the scheme can be generalized to other architectures, such as the pruned butterfly and multi-dimensional meshes. Section 7 closes with some final remarks.

2. Machine Model. We present our shared memory implementation on a *mesh*, consisting of an array of $\sqrt{n} \times \sqrt{n}$ processor-memory pairs, connected through a two-dimensional grid of communication links. The machine operates in lock-step, where in each step a processor can perform a constant amount of local computation (including accesses to its local memory) and can exchange a constant number of words with one of its direct neighbors. Our objective is to devise a distributed representation of $m \geq n$ shared variables on the mesh so that any n-tuple of read/write accesses to these variables can be served efficiently. The approach will be generalized to other architectures in §6.

The access protocol will make use of the following primitives, for which optimal algorithms are known in the literature. We call *ℓ-sorting* a sorting instance in which at most ℓ keys are initially assigned to each processor and are to be redistributed so that the ℓ smallest keys will be held by the first processor, the next ℓ smallest ones by the second processor, and so on, with the processors numbered in row major order. We have:

FACT 3 ([Kun93]). *Any ℓ-sorting can be performed on an n-node mesh in $O(\ell\sqrt{n})$ time.*

We call (ℓ_1, ℓ_2)*-routing* a routing problem in which each mesh processor is the source of at most ℓ_1 packets and the destination of at most ℓ_2 packets. We have:

FACT 4 ([SK94]). *Any (ℓ_1, ℓ_2)-routing can be performed on an n-node mesh in $O\left(\sqrt{\ell_1 \ell_2 n}\right)$ time.*

A simple bisection-based argument shows that this result is optimal in the general case. However, for a special class of (ℓ_1, ℓ_2)-routings, a better performance can be achieved as follows. Fix a tessellation of the mesh into n/s submeshes of s nodes each, and consider an (ℓ_1, ℓ_2)-routing where at most δs packets are destined for each submesh. We first use ℓ_1-sorting and (ℓ_1, δ)-routing to spread the packets evenly among the nodes of their destination submeshes, and then complete the routing by running n/s independent instances of (δ, ℓ_2)-routing within each submesh. The overall routing time becomes

$$O\left(\ell_1\sqrt{n} + \sqrt{\ell_1 \delta n} + \sqrt{\delta \ell_2 s}\right).$$

Comparing the $O\left(\sqrt{\ell_1 \ell_2 n}\right)$ complexity of the general (ℓ_1, ℓ_2)-routing algorithm with the above routing time, we see that the new algorithm is profitable when $\delta, \ell_1 = o(\ell_2)$ and $\delta s = o(\ell_1 n)$. This fact will be exploited in our access protocol, where packet routing is used to access selected copies of the variables. In particular, we will employ several nested tessellations of the mesh and provide strong bounds on the congestion within the submeshes of each tessellation, so that the above strategy can be applied. The packets will then be routed gradually to their destinations through a sequence of smaller and smaller submeshes.

3. The Hierarchical Memory Organization Scheme.

This section introduces the *Hierarchical Memory Organization Scheme* (HMOS), a mechanism through which m shared variables are distributed among the n memory modules of a processor network. The section is organized in two subsections: §3.1 presents the logical structure of HMOS, while §3.2 describes its actual implementation on the mesh.

3.1. Logical structure of the HMOS.

The HMOS is structured as $k+1$ levels of logical modules built upon the shared variables, where $k = O(\log\log n)$ is a nonnegative integer function of n, to be specified by the analysis. More specifically, starting from $m = n^\tau$ shared variables, for a fixed constant $\tau > 1$, the HMOS comprises m_i modules at level i, called *i-modules*, for $0 \le i \le k$, where the m_i's are strictly decreasing values that will be specified later. Modules are nested collections of variables, obtained as follows. First, each variable is replicated into $r = \Theta(1)$ copies, 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 $0 < i \le k$. It is easy to see that the above process will eventually create 3^{k-i} replicas of each i-module and $r3^k$ copies per variable. In the rest of this paper, we will reserve the term *copy* to denote the replica of a variable, and *i-block* to denote the replica of an i-module.

The difference between an i-module and one of its i-blocks is akin to the difference between a variable and one of its copies. Namely, an i-module represents an abstract entity, of which several physical replicas, its i-blocks, exist. Since the contents of k-modules are not replicated, the terms k-module and k-block will be used interchangeably. Note that a k-block is made of $(k-1)$-blocks, which in turn are made of $(k-2)$-blocks, and so on until 0-blocks are reached. The latter contain copies of variables.

Let V denote the set of shared variables, and U_i the set of i-modules, for $0 \leq i \leq k$. The HMOS is represented as a leveled *direct acyclic graph* (dag) $\mathcal{H}$, which is defined by a cascade of $k+1$ bipartite graphs, namely (V, U_0) and (U_{i-1}, U_i), $0 < i \leq k$, whose edges are directed left-to-right. In (V, U_0), each variable $v \in V$ is adjacent to r 0-modules, denoted by $\gamma_0(v,j)$, $1 \leq j \leq r$. For $0 < i \leq k$, in (U_{i-1}, U_i), each $(i-1)$-module u is adjacent to three i-modules denoted by $\gamma_i(u,j)$, $1 \leq j \leq 3$. For notational convenience, we will number the levels in the HMOS starting from -1, the level of the variables. As a consequence, nodes at level i, $0 \leq i \leq k$, are i-modules.

In the HMOS, each variable v uniquely identifies a single-source subdag $\mathcal{H}_v$ induced by all nodes reachable from $v \in V$. A straightforward property of $\mathcal{H}_v$ is that it contains $r3^k$ distinct source-sink paths which are in one-to-one correspondence with the $r3^k$ copies of v. Each source-sink path in $\mathcal{H}_v$ (hence, each copy of v) is uniquely identified by the string of nodes traversed by the path. Moreover, for $0 \leq i \leq k$, the suffix of any such string starting with a node u at level i of $\mathcal{H}_v$, identifies a specific i-block storing a copy of v. Note that several source-sink paths in $\mathcal{H}_v$ may correspond to strings with a common suffix starting from level i. In this case, the i-block corresponding to the common suffix will store several distinct copies of v. A small HMOS for 8 shared variables is shown in Figure 1.

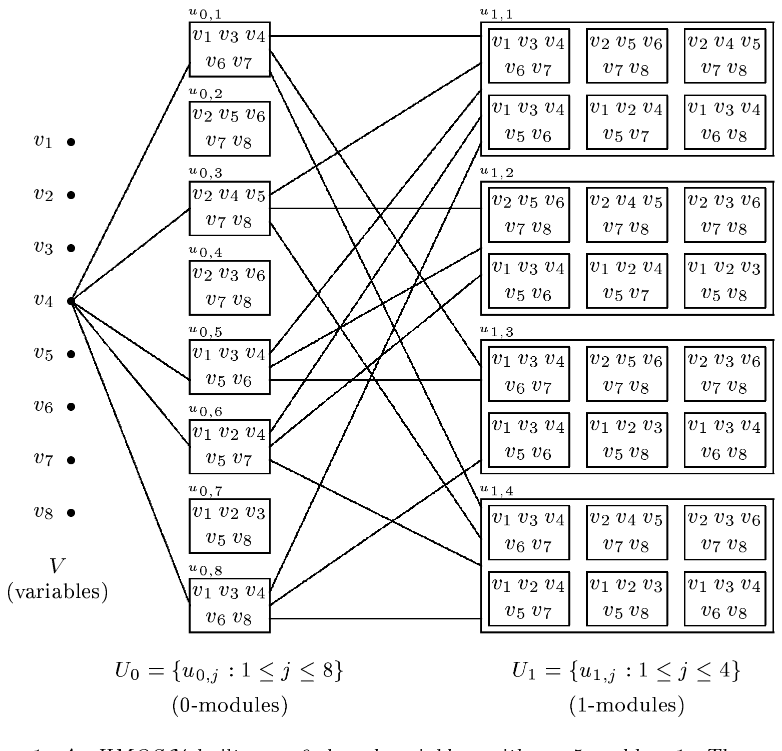
$u_{0,1}$ $v_1\ v_3\ v_4\ v_6\ v_7$
$u_{0,2}$ $v_2\ v_5\ v_6\ v_7\ v_8$
$u_{0,3}$ $v_2\ v_4\ v_5\ v_7\ v_8$
$u_{0,4}$ $v_2\ v_3\ v_6\ v_7\ v_8$
$u_{0,5}$ $v_1\ v_3\ v_4\ v_5\ v_6$
$u_{0,6}$ $v_1\ v_2\ v_4\ v_5\ v_7$
$u_{0,7}$ $v_1\ v_2\ v_3\ v_5\ v_8$
$u_{0,8}$ $v_1\ v_3\ v_4\ v_6\ v_8$
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8
V (variables)
$u_{1,1}$ $u_{1,2}$ $u_{1,3}$ $u_{1,4}$
$U_0 = \{u_{0,j} : 1 \leq j \leq 8\}$ (0-modules)
$U_1 = \{u_{1,j} : 1 \leq j \leq 4\}$ (1-modules)

FIG. 1. *An HMOS $\mathcal{H}$ built upon 8 shared variables, with $r=5$ and $k=1$. There are 8 0-modules and 4 1-modules/1-blocks. Each 1-block contains 6 0-blocks (physical copies of 0-modules), each of which in turn contains 5 copies of the variables. The edges shown in the figure are those of the subdag $\mathcal{H}_{v_4}$. Note that there are 15 edge-disjoint, source-sink paths in $\mathcal{H}_{v_4}$, each path corresponding to one the 15 copies of v_4.*

The component bipartite graphs of the HMOS must be carefully chosen in order to guarantee a good distribution of the copies of the variables, once the HMOS is mapped onto the processors' memory modules. More specifically, we require that the graphs exhibit good *expansion*, according to the following definition.

DEFINITION 5. *Let $G = (X, Y)$ be a bipartite graph, where each input node in X has degree d. For $0 < \alpha \leq 1$, $0 < \epsilon < 1$ and $1 \leq \mu \leq d$, G is said to have* (α, ϵ, μ)-expansion *if for any subset $S \subseteq X$, $|S| \leq \alpha|X|$, 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 let $|U_0| = n^\rho$, where $0 < \rho < 3/2$ is a parameter to be fixed by the analysis. We require that (V, U_0) has input degree r, for a fixed odd constant $r > 0$, output degree $|V|r/|U_0|$, and exhibits (α, ϵ, μ)-expansion, where $\alpha = n/m$, ϵ is a positive constant less than 1, and $\mu = (r + 1)/2$. Clearly, a necessary condition for the existence of such a graph is $(\alpha|V|)^{1-\epsilon} = n^{1-\epsilon} \leq n^\rho$, which implies $\rho + \epsilon - 1 \geq 0$. The analysis will determine suitable values for r, ϵ and ρ that guarantee the existence of (V, U_0). Moreover, an explicit construction for such graph will be available when $m = O\left(n^{9/2}\right)$.

The graphs (U_{i-1}, U_i), for $0 < i \leq k$, are derived from instances of varying size of the same combinatorial structure, the *Balanced Incomplete Block Design*, defined below.

DEFINITION 6 ([Hal86]). *A* Balanced Incomplete Block Design *with parameters w and q, or* (w, q)-BIBD, *is a bipartite graph (X, Y) such that*

- *$|Y| = w$;*
- *The degree of each node in X is q;*
- *For any two nodes $y_1, y_2 \in Y$ there is exactly one node $x \in X$ adjacent to both.*

From the definition, it immediately follows that $|X| = w(w - 1)/(q(q - 1))$ and that the degree of each node in Y is $(w - 1)/(q - 1)$. One important property of the BIBD, which we will heavily exploit, is stated in the following lemma.

LEMMA 7. *Let $G = (X, Y)$ be a (w, q)-BIBD. Consider a node $y \in Y$, and a subset $S \subseteq X$ such that any node in S is adjacent to y. Let E be a set of $\mu|S|$ edges, with $\mu \leq q$, containing μ outgoing edges from each x in S, and let $\Gamma^E(S)$ denote the set of nodes of U reached by the chosen edges. Then*

$$|\Gamma^E(S)| \geq (\mu - 1)|S| + 1.$$

Proof. The definition of BIBD implies that no two nodes in S share a neighbor other than y. If $y \notin \Gamma^E(S)$, then $|\Gamma^E(S)| = \mu|S|$. Otherwise, at most $|S|$ of the selected edges reach y, the other $(\mu - 1)|S|$ reach distinct nodes. □

COROLLARY 8. *A (w, q)-BIBD has $(1, 1/2, \mu)$-expansion, for every $2 \leq \mu \leq q$.*

Proof. Let $G = (X, Y)$ be a (w, q)-BIBD, and let S be an arbitrary subset of input nodes. We now show that $|\Gamma^E(S)| > \sqrt{(\mu - 1)\mu|S|}$, for an arbitrary set E containing μ outgoing edges from each node in S. Assume that $|\Gamma^E(S)| \leq \sqrt{(\mu - 1)\mu|S|}$. Then, there must then be an output node in $\Gamma^E(S)$ that is adjacent to at least $\sqrt{|S|\mu/(\mu - 1)}$ nodes in S. According to Lemma 7 this implies that $|\Gamma^E(S)| \geq (\mu - 1)\sqrt{|S|\mu/(\mu - 1)} + 1 > \sqrt{(\mu - 1)\mu|S|}$, which contradicts our assumption. □

For convenience, we assume that both n and $3^k n^\rho$ are even powers of three. Consider the sequence of integers $d_0, d_1, \ldots, d_k$ defined as

$$\begin{cases} d_0 &= \log_3 n^\rho, \\ d_i &= 2\left\lceil \frac{1}{2}\left(\frac{d_{i-1}}{2} + 1 + k - i\right)\right\rceil - k + i, \quad \text{for } 1 \le i \le k. \end{cases}$$

For $0 \le i \le k$, set the number of i-modules to be $m_i = 3^{d_i}$. The following two properties are easily established.

(i) $d_i + k - i$ is even, for $0 \le i \le k$.

(ii) $3\sqrt{m_{i-1}} \le m_i \le 16\sqrt{m_{i-1}}$, which implies that $m_{i-1} \le m_i(m_i - 1)/6$, for $1 \le i \le k$, and $m_i = \Theta\left(n^{\rho/2^i}\right)$, for $0 \le i \le k$.

We choose (U_{i-1}, U_i) as a subgraph of a $(m_i, 3)$-BIBD, where $m_i(m_i - 1)/6 - m_{i-1}$ inputs are removed along with their incident edges. The inputs to be removed are chosen in such a way that the remaining edges are evenly distributed among the outputs, so that each node of U_i becomes adjacent to

$$n_i = \frac{3m_{i-1}}{m_i}$$

nodes of U_{i-1}. An efficient construction of such subgraphs is described in the appendix.

3.2. Mapping the HMOS onto the Mesh. The HMOS is physically mapped onto the mesh by storing each i-block in a distinct submesh of appropriate size. For some values of the parameter ρ, the number of 0-blocks exceed the mesh nodes, hence a single mesh node must store more than one 0-block. There are $3^{k-i}m_i = \Theta\left(3^{k-i}n^{\rho/2^i}\right)$ i-blocks, $0 \le i \le k$, and each i-block contains exactly n_i $(i-1)$-blocks, $1 \le i \le k$. We define k nested tessellations of the mesh into submeshes as follows. The outermost tessellation is a subdivision of the mesh into m_k submeshes (*k-submeshes*), each storing a distinct k-block. Each k-submesh is in turn tessellated into n_k *$(k-1)$-submeshes* storing the component $(k-1)$-blocks of the k-block assigned to the k-submesh. In general, for $2 \le i \le k$, each *i-submesh*, storing a given i-block, is tessellated into n_i *$(i-1)$-submeshes* storing its component $(i-1)$-blocks. Thus, for $1 \le i \le k$, we have a total of

$$m_k n_k n_{k-1} \cdots n_{i+1} = 3^{k-i}m_i = \Theta\left(3^{k-i}n^{\rho/2^i}\right)$$

i-submeshes, each of size

$$t_i = \frac{n}{3^{k-i}m_i} = \frac{n}{3^{d_i+k-i}} = \Theta\left(3^{i-k}n^{1-\rho/2^i}\right).$$

Note that the assumption $k = O(\log\log n)$ ensures $t_i \ge 1$, for $i \ge 1$ and n sufficiently large. Moreover, since both $\log_3 n$ and $d_i + k - i$ are even, we have that t_i is an even power of three, hence $\sqrt{t_i}$ is integral and $\sqrt{t_{i-1}}$ divides $\sqrt{t_i}$. As a consequence, the $(i-1)$-submeshes storing the n_i component $(i-1)$-blocks of an i-block are all contained within the i-submesh storing the i-block. Finally, the organization of the $3^k n^\rho$ 0-blocks depends on the parameter ρ. When $3^k n^\rho < n$, we assign each 0-block to a submesh of $t_0 = n^{1-\rho}/3^k$ nodes and evenly partition the contents of the block among these nodes. Otherwise, when $3^k n^\rho > n$ there are more 0-blocks than processors, so we assign $3^k n^{\rho-1}$ 0-blocks to each processor. In either case, each processor stores $r3^k m/n$ copies of variables.

4. The Access Protocol.

Suppose that m shared variables are distributed among the n mesh nodes according to the HMOS. In this section, we present the protocol that realizes a parallel access to any n-tuple of variables, where each processor issues a read/write request for a distinct variable. (The case of concurrent accesses can be reduced to the case of exclusive accesses in time $O(\sqrt{n})$ by means of standard sorting-based techniques for leader election and data distribution [Lei92].)

Let S denote the set of variables to be accessed. The access protocol consists of a *copy selection phase* followed by a *routing phase*. In the first phase, a suitable set of copies for the variables in S is chosen, so that accessing these copies will enforce data consistency and generate low memory contention and network congestion. In the subsequent phase, the selected copies are effectively accessed through an appropriate routing strategy. In case of read operations, the accessed data are returned to the requesting processors along the reverse routing paths.

4.1. Copy Selection Phase.

Copy selection achieves the double objective of controlling both memory contention and network congestion by means of a single mechanism. The hierarchical structure of the HMOS provides a geographical distribution of the copies into nested regions of the network. By carefully limiting the number of copies that have to be accessed in any block at any level, we reduce the number of packets that will ever be routed to any such region, which allows us to adopt the efficient routing strategy illustrated in §2.

4.1.1. A New Consistency Rule.

Recall from §3.1 that the $r3^k$ copies of a variable v are associated with the source-sink paths of $\mathcal{H}_v$, the subdag of $\mathcal{H}$ induced by v and by all of its descendants. Suppose that we want to read/write v. In order to guarantee consistency, the copies of v to be accessed are selected according to a new rule, which extends the majority protocol of [UW87] to fit the structure of the HMOS. Specifically, we require that the selected copies form a *target set*, which is defined as follows. Let C_v be a set of copies of v and let $\mathcal{N}(C_v)$ be the set of nodes of $\mathcal{H}_v$ belonging to the source-sink paths associated with these copies. Recall that r is odd, and let $\mu = (r+1)/2$.

DEFINITION 9. *C_v is a* target set *for v if $|C_v| = \mu 2^k$ and the following condition holds: a majority (μ) of the nodes at level 0 of $\mathcal{H}_v$ belong to $\mathcal{N}(C_v)$, and, for each node at level i belonging to $\mathcal{N}(C_v)$, a majority (two) of its successors at level $i+1$ belong to $\mathcal{N}(C_v)$, for $0 \leq i < k$.*

Figure 2 depicts the source-sink paths corresponding to a target set C_{v_4} for variable v_4 in the HMOS of Figure 1.

An easy inductive argument shows that any two target sets for the same variable have nonempty intersection. Based on such a property we can guarantee consistency, that is, ensure that a read always returns the most updated value, as follows. As customary in any multi-copy approach, we equip each copy with a time-stamp, which is set to the current step whenever the copy is written. A read or write operation on a variable v is simulated by accessing a target set of its copies. By the intersection property of target sets, the copies accessed for reading a variable v must include at least one of the most recently written copies for v, which can be identified by looking for the most recent time-stamp. It should be noted that a target set contains only $\mu 2^k$ copies out of the $r3^k$ total copies of a variable. Therefore, unlike previous protocols, we maintain consistency by accessing much less than a majority of the copies.

4.1.2. The Selection Procedure.

The copy selection phase determines a target set C_v for each $v \in S$. This is accomplished in $k+1$ iterations, numbered from 0

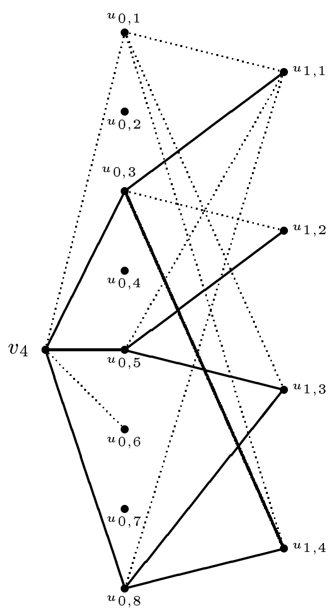

FIG. 2. *The source-sink paths (solid lines) in $\mathcal{H}_{v_4}$ corresponding to a target set C_{v_4} of 6 copies for variable v_4 in the HMOS $\mathcal{H}$ of Figure 1.*

to k, during which the nodes of the $\mathcal{H}_v$'s are marked in a top-down fashion from the sources to the sinks. More specifically, for every $\mathcal{H}_v$, with $v \in S$, Iteration 0 marks the source v and μ of its successors (0-modules); Iteration i, $0 < i \leq k$, marks two successors (i-modules) of each marked node at level $i-1$. In this fashion, at the end of Iteration i, $0 \leq i \leq k$, the marked nodes in each $\mathcal{H}_v$ form exactly $\mu 2^i$ distinct paths from the source to nodes at level i (in what follows we refer to such paths as *marked paths*). We choose C_v as the set of copies of v corresponding to the $\mu 2^k$ source-sink marked paths in $\mathcal{H}_v$ at the end of the last iteration.

It is important to notice that a node of $\mathcal{H}$, say an i-module u, may belong to several subdags $\mathcal{H}_v$ corresponding to variables in S. During the copy selection procedure, we keep track of u independently in each such subdag, hence, u may result marked in some of the subdags and unmarked in the others. Suppose that at the end of Iteration i u is marked in some subdags, and that a total of h marked paths in these subdags reach u. This implies that at the end of copy selection there will be $h2^{k-i}$ marked paths that pass through u, that is, $h2^{k-i}$ copies in $\bigcup_{v \in S} C_v$ stored in i-blocks of u. In Iteration $i+1$ the two successors of u to be marked are chosen to be the same for all subdags in which u is marked; hence, for each chosen successor node, u will contribute h paths to the total number of marked paths that will pass through that node at the end of the iteration. The main idea behind the copy selection phase is to control congestion in $(i+1)$-blocks by choosing the nodes to be marked in Iteration $i+1$ in such a way to keep the number of marked paths passing through each such node under some reasonable bound.

The following notations will be needed to describe the copy selection procedure:

DEFINITION 10. *For $0 \leq i \leq k$, $A_i \subseteq U_i$ denotes the set of i-modules that are marked in some $\mathcal{H}_v$, with $v \in S$, during Iteration i. The modules in A_i are called* active modules.

DEFINITION 11. *The* weight $w(u)$ *of an active module* $u \in A_i$ *is the sum, over all variables* $v \in S$*, of the number of marked paths from* v *to* u *in* $\mathcal{H}_v$.

From the previous discussion we conclude that exactly $w(u)$ copies in $\bigcup_{v\in S} C_v$ will reside in *each* of the selected 2^{k-i} i-blocks of u, while the other i-blocks of u will not contain any copy in $\bigcup_{v\in S} C_v$. Using the expansion properties of the HMOS, we will be able to guarantee suitably low values for the $w(u)$'s.

The actions performed by the copy selection procedure are reported below. Since Iteration 0 is different from the others, it is described separately. In order to understand the parameters used in Iteration 0, recall that we chose (V, U_0) to have $(n/m, \epsilon, \mu)$-expansion, where $0 < \epsilon < 1$ and $\mu = (r+1)/2$. In other words, the graph guarantees that for any set S of at most $|S| \leq n$ variables and any choice of μ 0-modules adjacent to each variable, the overall number of chosen 0-modules is at least $\beta|S|^{1-\epsilon}$, for some constant $\beta > 0$. Finally, recall that each processor is in charge of a distinct variable.

Iteration 0.

1. For $v \in S$, let p_v denote the processor in charge of v. Each p_v creates r *copy-packets* denoted by the tuples $[p_v, v, u_j = \gamma_0(v,j), h_{v,u_j} = 1]$, for $1 \leq j \leq r$. Upon creation, all copy-packets are regarded as *unmarked.*

2. Let $\mathcal{CP}_0$ denote an initially empty set. The following three substeps are executed until μ copy-packets for each variable are put in $\mathcal{CP}_0$.

(i) *Sorting*: Sort all unmarked copy-packets by their third component.

(ii) *Selection*: For each $u \in U_0$, let c_u be the number of copy-packets with third component u in the sorted sequence. If $c_u \leq (2r/\beta)n^{\epsilon}$, then all such packets are marked. Otherwise, none of them is marked. Subsequently, all the packets are sent back to their originating processors.

(iii) *Counting*: For each $v \in S$, p_v counts the total number of its copy-packets marked so far. If these are at least μ, then exactly μ of them are put in $\mathcal{CP}_0$ while the remaining $r - \mu$ copy-packets are discarded. Otherwise, the marked copy-packets are locally buffered.

3. For each $v \in S$, p_v marks the source of $\mathcal{H}_v$ and μ of its successors corresponding to the copy-packets for v included in $\mathcal{CP}_0$.

The analysis will show that the set $\mathcal{CP}_0$ is determined in at most $\log n + 1$ iterations of Step 2.

Iteration i $(1 \leq i \leq k)$*.* At the beginning of Iteration i, the mesh processors store a set $\mathcal{CP}_{i-1}$ of copy-packets. Specifically, each processor p_v stores copy-packets of type $[p_v, v, u, h_{v,u}]$, where $u \in A_{i-1}$ and $\sum_u h_{v,u} = \mu 2^{i-1}$. The value $h_{v,u}$ reflects the *multiplicity* of u with respect to v, that is, the number of distinct marked paths in $\mathcal{H}_v$ from v to u. Iteration i consists of the following steps.

1. The copy-packets in $\mathcal{CP}_{i-1}$ are sorted by their third component;

2. For each group of packets with the same third component $u \in A_{i-1}$, a *leader* processor p_u is elected. Each p_u computes $w(u)$ as the sum of the multiplicities carried by the packets in its group, and creates the three *module-packets* $[p_u, u, \gamma(u,j), w(u)]$, for $1 \leq j \leq 3$;

3. The $3|A_{i-1}|$ module-packets are sorted lexicographically by their third and fourth components;

4. For each $x \in U_i$, a maximal subset $\mathcal{P}_x$ of module-packets with third component x is chosen, such that

$$\sum_{[p_u,u,x,w(u)]\in\mathcal{P}_x} w(u) \leq c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}} + n^{1-\frac{1-\epsilon}{2^{i-1}}}\right);$$

5. The chosen packets are sent back to their originating leader processors;

6. Assume that a leader processor p_u receives $h \leq 3$ module-packets back. If $h < 2$, then p_u selects $2 - h$ extra module-packets from the $3 - h$ that were not chosen in the previous step. Otherwise, p_u selects two module-packets among those received;

7. Let $[p_u, u, \gamma(u, j_1), w(u)]$ and $[p_u, u, \gamma(u, j_2), w(u)]$ be the two module-packets selected by p_u. Processor p_u sends the names $\gamma(u, j_1)$ and $\gamma(u, j_2)$ to the processors storing the copy-packets in its group;

8. Each copy-packet $[p_v, v, u, h_{v,u}] \in \mathcal{CP}_{i-1}$ is augmented with two extra components containing $\gamma(u, j_1)$ and $\gamma(u, j_2)$;

9. The augmented copy-packets are routed back to the processors in charge of their respective variables;

10. For any augmented copy packet $[p_v, v, u, h_{v,u}, \gamma(u, j_1), \gamma(u, j_2)]$ received at processor p_v, the nodes $\gamma(u, j_1)$ and $\gamma(u, j_2)$ at level i of $\mathcal{H}_v$ are marked (note that the same node may be redundantly marked more than once). Moreover for each newly marked node u' at level i, a copy packet $[p_v, v, u', h_{v,u'}]$ is created, where $h_{v,u'}$ is obtained by summing up the multiplicities of the received (augmented) copy-packets carrying u' in one of the two extra components. These new copy-packets form the set $\mathcal{CP}_i$, while all other packets are discarded.

When Iteration k terminates, for each $v \in S$, p_v determines the set C_v of copies to be accessed as those corresponding to the $\mu 2^k$ source-sink marked paths in $\mathcal{H}_v$. It is easily seen that for each $v \in S$, the set C_v computed by the copy selection procedure is indeed a target set for v.

4.1.3. Analysis of the Selection Procedure. We now determine the running time of the selection procedure described above. Let us first consider Iteration 0. By Fact 3 and since $r = O(1)$, Steps 1 and 3 require altogether $O(\sqrt{n})$ time. The sorting, selection and counting substeps of Step 2 can be implemented in terms of sorting and prefix operations in $O(\sqrt{n})$ time. It will be shown in Lemma 12 that $\log n + 1$ executions of such substeps are sufficient. Therefore, Iteration 0 requires $O(\sqrt{n}\log n)$ time altogether. For $i \geq 1$, Iteration i can be implemented in terms of a constant number of sorting and prefix operations on a set of $O\left(\mu 2^{i-1} n\right)$ packets, yielding a running time of $O\left(\sqrt{n} 2^{i-1}\right)$. Therefore, copy selection is completed in time

$$O\left(\sqrt{n}\log n + \sqrt{n}\sum_{i=1}^{k} 2^{i-1}\right) = O\left(\sqrt{n}\left(\log n + 2^k\right)\right). \tag{1}$$

LEMMA 12. *After* $\log n + 1$ *executions of Step 2 in Iteration 0, the set* $\mathcal{CP}_0$ *contains exactly* μn *copy-packets. Moreover, for each active 0-module* $u \in A_0$, $w(u) \leq (2r/\beta)n^{\epsilon}$.

Proof. Let S_j be the number of variables for which fewer than μ copy-packets have been selected by the end of the j-th execution of Step 2. For the sake of convenience, set $S_0 = |S| = n$. We now show by induction that

$$S_j \leq \frac{n}{2^j},$$

for any $j \geq 0$, and this will imply that for $T = \log n + 1$, $S_T = 0$. The inequality for S_0 is immediate, establishing the basis. Assume that the inequality holds for $j - 1$, and suppose, for a contradiction, that $S_j > n/2^j$. By using the expansion properties of (V, U_0), it is easy to see that in the j-th execution of the selection substep, at least $\beta S_j^{1-\epsilon}$ 0-modules are addressed by unmarked copy-packets. All such 0-modules must have been congested in the previous iteration (i.e., addressed by more than $(2r/\beta)n^\epsilon$ copy-packets), which accounts for a total of at least

$$\frac{2r}{\beta}n^\epsilon \beta S_j^{1-\epsilon} > rS_{j-1}$$

unmarked copy-packets involved in that iteration. However, this is impossible since S_{j-1} variables account for at most rS_{j-1} copy packets.

The bound on $w(u)$ is easily established by observing that for each 0-module u, all copy-packets with third component u that are added to $\mathcal{CP}_0$ are marked during the same iteration of Step 2. Therefore

$$w(u) \leq \frac{2r}{\beta}n^\epsilon. \qquad \square$$

It must be remarked that the copy selection phase can be improved in a number of ways to obtain a faster running time at the expense of a more complex implementation. However, to avoid further complications to the presentation, we chose to describe a simpler yet slightly less efficient implementation, since, as shown in §5, its complexity does not influence the overall running time of the access protocol.

To complete the analysis, it remains to establish the bound on the weight $w(u)$ of any $u \in A_i$, at the end of Iteration i. Recall that the sum of the multiplicities of the copy-packets in $\mathcal{CP}_i$ with third component u yields $w(u)$. Therefore, for $0 \leq i \leq k$,

$$\sum_{u \in A_i} w(u) = \mu 2^i n. \tag{2}$$

LEMMA 13. *There is a suitable constant $c \geq 3$ such that, at the end of Iteration i, for each $u \in A_i$, $0 \leq i \leq k$,*

$$w(u) \leq c\mu 2^i n^{1-\frac{1-\epsilon}{2^i}}.$$

Proof. The proof proceeds by induction on i. The basis ($i = 0$) is established by Lemma 12. Suppose that the inequality holds for $i-1$ and let x be an i-module. The weight of x is determined by Steps 4 and 6 of Iteration i. More precisely, recall that $\mathcal{P}_x$ is the set of module-packets of kind $[p_u, u, x, w(u)]$ selected in Step 4, and let $\mathcal{P}'_x$ be the set of additional module-packets (still with third component x) not in $\mathcal{P}_x$, selected in Step 6. It is easy to see that

$$w(x) \leq \sum_{[p_u,u,x,w(u)] \in \mathcal{P}_x \cup \mathcal{P}'_x} w(u).$$

Because of the way the module-packets are selected in Step 4, we already know that

$$\sum_{[p_u,u,x,w(u)] \in \mathcal{P}_x} w(u) \leq c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}} + n^{1-\frac{1-\epsilon}{2^{i-1}}}\right).$$

We must only show that the contribution of $\mathcal{P}'_x$ to $w(x)$ is not too large. In order to derive a contradiction, we suppose that $w(x) > c\mu 2^i n^{1-\frac{1-\epsilon}{2^i}}$. This implies

$$\sum_{[p_u,u,x,w(u)] \in \mathcal{P}'_x} w(u) > c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}} - n^{1-\frac{1-\epsilon}{2^{i-1}}}\right).$$

Define $S_x = \{u \in A_{i-1} : [p_u, u, x, w(u)] \in \mathcal{P}'_x\}$, so that

$$\sum_{[p_u,u,x,w(u)]\in\mathcal{P}'_x} w(u) = \sum_{u\in S_x} w(u).$$

By the inductive hypothesis, the weight of each $u \in S_x$ is at most $c\mu 2^{i-1} n^{1-\frac{1-\epsilon}{2^{i-1}}}$, therefore

$$|S_x| \geq \frac{\sum_{u\in S_x} w(u)}{c\mu 2^{i-1} n^{1-\frac{1-\epsilon}{2^{i-1}}}} > n^{\frac{1-\epsilon}{2^i}} - 1.$$

Note that for each $u \in S_x$, at least two module-packets, including $[p_u, u, x, w(u)]$, have not been selected in Step 4. Let

$$\Gamma^*(S_x) = \{y \in U_i : \exists\, u \in S_x \text{ s.t. } [p_u, u, y, w(u)] \text{ has not been selected in Step 4}\}.$$

Note that in the graph (U_{i-1}, U_i) each $u \in S_x$ is adjacent to either two of three nodes in $\Gamma^*(S_x)$, one of which is x (see Figure 3).

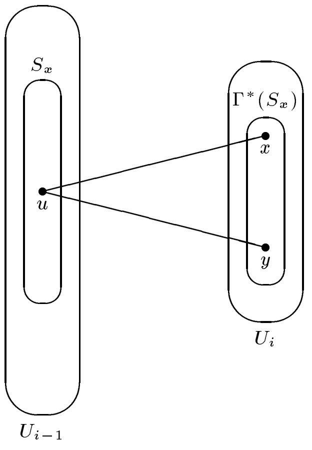

FIG. 3. *Critical modules in the proof of Lemma 13.*

Since (U_{i-1}, U_i) is a BIBD, we can apply Lemma 7 and conclude that $|\Gamma^*(S_x)| \geq |S_x| + 1 > n^{\frac{1-\epsilon}{2^i}}$. We now show that the global weight assigned in Step 4 to all the nodes in $\Gamma^*(S_x)$ exceeds the total weight carried by all the module packets, thereby leading to a contradiction. Let $y \in \Gamma^*(S_x)$ and let $[p_{u'}, u', y, w(u')]$ be a module packet which has not been selected in Step 4, with $u' \in S_x$. Then, we must have

$$w(u') + \sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u) > c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}} + n^{1-\frac{1-\epsilon}{2^{i-1}}}\right),$$

that is,

$$\sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u) > c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}} + n^{1-\frac{1-\epsilon}{2^{i-1}}}\right) - c\mu 2^{i-1} n^{1-\frac{1-\epsilon}{2^{i-1}}} = c\mu 2^{i-1} n^{1-\frac{1-\epsilon}{2^i}}.$$

Adding up the contributions of all nodes in $\Gamma^*(S_x)$ we get

$$\begin{aligned}\sum_{y\in\Gamma^*(S_x)} \sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u) &> |\Gamma^*(S_x)| c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}}\right) \\ &> n^{\frac{1-\epsilon}{2^i}} c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}}\right) = c\mu 2^{i-1} n.\end{aligned}$$

Since $c \geq 3$, the above inequality leads to a contradiction because

$$\sum_{y\in\Gamma^*(S_x)} \sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u) \leq \sum_{y\in U_i} \sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u) \leq 3 \sum_{u\in A_{i-1}} w(u) = 3\mu 2^{i-1}n,$$

where the last equality follows from Equation 2. □

4.2. Routing Phase. After copy selection is completed, the copies in $\bigcup_{v\in S} C_v$ have to be accessed. Each request is encapsulated in a distinct packet, routed from the requesting processor (*origin*) to the processor storing the copy (*destination*), and back to the origin. The idea is to route the packets in stages so that they are moved gradually closer to their destinations through smaller and smaller submeshes, in accordance with the tessellations defined on the mesh. As argued in §2, when the number of packets destined for any submesh is not too large, such a strategy yields more profitable results than sending the packets directly to their destinations.

The origin-destination part of a packet's journey consists of $k+2$ routing stages, numbered from $k+1$ down to 0. Stage i, with $k+1 \geq i \geq 1$, is executed in parallel and independently in every i-submesh (here, the whole mesh is viewed as a $(k+1)$-submesh). In this stage the packets are routed to arbitrary positions in the $(i-1)$-submeshes hosting their destination $(i-1)$-blocks, in such a way that the processors of each submesh receive approximately the same number of packets. This can be achieved by first sorting the packets according to their destination submeshes, and then ranking the packets destined to the same submesh. Observe that when $3^k n^\rho < n$, a 0-block is assigned to a 0-submesh of $t_0 = n^{1-\rho}/3^k$ nodes. By the end of Stage 1, each packet reaches a processor within its destination 0-submesh, and in Stage 0 is sent to its final destination. Instead, when $3^k n^\rho \geq n$, there are $n^{\rho-1}3^k$ 0-blocks stored within a single processor, hence each packet is at its final destination by the end of Stage 1, and Stage 0 is not needed. In either case, once the packet reaches its final destination, the request it carries is satisfied.

In order to estimate the time complexity of the above protocol, we need to determine the maximum number of packets sent and received by any processor in each stage. More formally, let δ_i, for $k+1 \geq i \geq 0$, denote the maximum number of packets held by any processor at the beginning of Stage i. Let also δ_{-1} be the maximum number of packets received by a processor at the end of Stage 0, when such stage is needed (i.e., when $3^k n^\rho < n$). We have:

LEMMA 14. *Let $k \geq 0$. Then*

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

When $3^k n^\rho < n$, we also have $\delta_{-1} = O(\mu n^\epsilon)$.

Proof. The statement is immediately evident for δ_{k+1}, since every target set contains $\mu 2^k$ copies. By Lemma 13, an i-block is addressed by at most $c\mu 2^i n^{1-(1-\epsilon)/2^i}$ packets, for $k \geq i \geq 1$. Since there are $t_i = \Theta\left(3^{i-k} n^{1-\rho/2^i}\right)$ processors storing an i-block, we have

$$\delta_i \leq \frac{c\mu 2^i n^{1-\frac{1-\epsilon}{2^i}}}{t_i} = O\left(\mu 2^i 3^{k-i} n^{\frac{\rho+\epsilon-1}{2^i}}\right).$$

In order to establish the bound for δ_0, we distinguish between two cases. If $3^k n^\rho < n$, each 0-block is assigned to a submesh of $t_0 = n^{1-\rho}/3^k$ nodes and, by Lemma 13, is addressed by at most $c\mu n^\epsilon$ packets, whence $\delta_0 = c\mu n^\epsilon / t_0 = O\left(\mu 3^k n^{\rho+\epsilon-1}\right)$. In this case, Stage 0 is needed to bring the packets to their final destinations. At the end of this stage, each processor receives at most $\delta_{-1} \leq c\mu n^\epsilon$ packets. Otherwise, if $3^k n^\rho \geq n$, there are $3^k n^{\rho-1}$ 0-blocks stored within a single processor, whence $\delta_0 = c\mu n^\epsilon 3^k n^{\rho-1} = O\left(\mu 3^k n^{\rho+\epsilon-1}\right)$ as before, but the routing terminates with Stage 1. □

Set $t_{k+1} = n$, the size of the entire mesh, and let T_i be the time complexity of Stage i, for $k+1 \geq i \geq 0$.

LEMMA 15. *We have:*

$$\begin{aligned} T_{k+1} &= O\left(2^k n^{\frac{1}{2}+\frac{\rho+\epsilon-1}{2^{k+1}}}\right), \\ T_i &= O\left(2^i 3^{\frac{k-i}{2}} n^{\frac{1}{2}+\frac{2\rho+3\epsilon-3}{2^{i+1}}}\right), \qquad \text{for } k \geq i \geq 1. \end{aligned}$$

When $3^k n^\rho < n$, we also have $T_0 = O\left(n^\epsilon\right)$.

Proof. Recall that for $k+1 \geq i \geq 1$, Stage i is executed in parallel and independently in each i-submesh. The initial sorting and ranking are accomplished in $O\left(\delta_i \sqrt{t_i}\right)$ time. By Fact 4, the subsequent (δ_i, δ_{i-1})-routing requires $O\left(\sqrt{\delta_i \delta_{i-1} t_i}\right)$ time. Since $\delta_i \leq \delta_{i-1}$, we get $T_i = O\left(\sqrt{\delta_i \delta_{i-1} t_i}\right)$. When $3^k n^\rho < n$, Stage 0 consists of a (δ_0, δ_{-1})-routing in each submesh of size t_0, requiring $O\left(\sqrt{\delta_0 \delta_{-1} t_0}\right)$ time. The lemma follows by plugging in the values for the δ_i's and the t_i's, and by recalling that μ is a constant. □

After reaching their destinations, the packets relative to read operations must return to their origins carrying the accessed data. This second part of the routing can be accomplished by running the above protocol backwards, thus maintaining the same time complexity.

THEOREM 16. *The access protocol requires overall time*

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

Proof. The running time of the access protocol is obtained by adding the contributions of the copy selection and routing phases together. The complexity of copy selection is given by Equation (1), while the routing time is obtained by summing of the T_i's given in Lemma 15. Note that $\rho+\epsilon-1 \geq 0$, therefore the term $n^{1/2} 2^k n^{(\rho+\epsilon-1)/2^{k+1}}$ dominates the term $n^{1/2} 2^k$ coming from copy selection. Note also that the term n^ϵ, which accounts for the complexity of Stage 0, does not dominate when $3^k n^\rho \geq n$. □

5. Tuning of the Parameters.

The complexity of the access protocol established by Theorem 16 is a function of the following design parameters:

(i) k: there are $k+1$ levels in the HMOS;
(ii) ρ: there are n^ρ 0-modules in U_0;
(iii) ϵ: the first graph of the HMOS (V, U_0) has $(n/m, \epsilon, \mu)$-expansion;
(iv) r: the (odd) input degree of (V, U_0).

Furthermore, recall that $m = |V| = n^\tau$, for some constant $\tau > 1$ and that $\mu = (r+1)/2$.

The goal of this section is to determine suitable values of the above parameters that guarantee the existence of graph (V, U_0) and yield a good performance of the access protocol. Such performance is closely related to the redundancy of the HMOS, that is, the number of copies $(r3^k)$ used per variable. On the one hand, using many copies per variable yields better access times, while, on the other, lower redundancy yields simpler and space-efficient schemes. We will consider two scenarios: in the first scenario, we optimize the parameters under the assumption that the number of copies for each variable $r3^k$ can grow arbitrarily large. In the second scenario, we optimize under the restriction that the scheme uses no more than a constant number of copies for each variable.

We need the following technical result, which is a straightforward adaptation of Lemma 4 in [PP97a]:

LEMMA 17. *Let $m = n^\tau$, with constant $\tau > 1$. There is a suitable constant $c > 0$ such that for any odd constant $r \geq c\tau \log \tau$, a random bipartite graph $G = (V, U_0)$ with $|V| = m$, $|U_0| = n$, input degree r and output degree mr/n has $(n/m, \epsilon, \mu)$-expansion with $\mu = (r+1)/2$ and $\epsilon = (\tau - 1)/\mu$, with high probability.*

We are now ready to prove one of the main results of this paper, which was stated in §1.1.

Proof of Theorem 1. We fix $\rho = 1$ and choose r to be the smallest odd integer greater than $\max\{c\tau \log \tau, 6(\tau - 1)\}$. For such values, Lemma 17 ensures the existence of (V, U_0) with $(n/m, \epsilon, (r+1)/2)$-expansion, where $\epsilon \leq 1/3$. Since $2\rho + 3\epsilon - 3 \leq 0$, the complexity of the access protocol given in Theorem 16 becomes

$$T = O\left(2^k n^{\frac{1}{2} + \frac{\epsilon}{2^{k+1}}}\right). \tag{3}$$

By fixing $k = \max\{0, \lfloor \log_2(\epsilon/\eta) \rfloor\}$, we have $2^{k+1} \geq \epsilon/\eta$, whence $T = O\left(n^{1/2+\eta}\right)$ and $R = r3^k = O\left(1/\eta^{\log_3 2}\right) = O\left(1/\eta^{1.59}\right)$. By instead fixing $k = \log_2 \log_2 n + O(1)$, so that $2^{k+1} \geq \epsilon \log_2 n$, we have $T = O\left(n^{\frac{1}{2}} \log n\right)$ and $R = O\left(\log^{1.59} n\right)$. □

As already noted before, the HMOS underlying the above result is fully constructive, except for the first graph (V, U_0), for which Lemma 17 only guarantees existence. In practice, one can resort to a random graph for (V, U_0), which, as the lemma shows, will exhibit the required expansion property with high probability. Although no explicit construction for (V, U_0) is known in the general case, this graph needs only weak expansion, which makes it more amenable to explicit constructions than the graphs employed in previous schemes (e.g., [UW87, AHMP87]).

In fact, an explicit construction for (V, U_0) can be obtained when the shared memory size m is within certain ranges. For example, [PP97] shows how to construct a bipartite graph with $m = \Theta\left(n^{3/2}\right)$ inputs, n outputs and input degree $r = 3$, which has $(n/m, 1/3, 2)$-expansion. This graph can be efficiently represented using constant storage per node. Thus, using this graph as (V, U_0) when $m = \Theta\left(n^{3/2}\right)$, the result of Theorem 1 still holds and the HMOS becomes fully constructive.

A larger range of values for m for which the HMOS can be made fully constructive, still yielding nontrivial performance, can be obtained by employing other graphs for (V, U_0). This is shown below, thus proving Theorem 2, which was stated in the introduction.

Proof of Theorem 2. Let us consider first the case $\tau \leq 3/2$. We assume $m = x(x-1)/6$, where x is an even power of three. The argument for different values of m requires only trivial modifications. Fix $n^\rho = x = \Theta\left(n^{\tau/2}\right)$ and choose (V, U_0) as an $(n^\rho, 3)$-BIBD. By Corollary 8, such graph has $(1, \epsilon, 2)$-expansion, with $\epsilon = 1/2$.

Since $2\rho + 3\epsilon - 3 = O(1/\log n)$, the complexity of the access protocol is still given by Equation (3), and the same argument used to prove Theorem 1 carries through. Consider now the range $3/2 < \tau \leq 13/6$ and choose n^ρ and (V, U_0) as before. By plugging $\epsilon = 1/2$ and $n^\rho = \Theta\left(n^{\tau/2}\right)$ in the complexity formula given in Theorem 16 and choosing $k = O(1)$ large enough and even, the complexity of the access protocol becomes

$$T = O\left(n^{\frac{1}{2}+\frac{2\tau-3}{8}}\right) = O\left(n^{\frac{2\tau+1}{8}}\right).$$

For $\tau \geq 13/6$, it is convenient to choose (V, U_0) as a $3-(n^\rho, 5, 3)$ *design*, a graph with the following properties: $|U_0| = n^\rho$, each node of V has degree 5, and for every three distinct nodes u_1, u_2, u_3 of U_0 there are exactly 3 nodes of V adjacent to all three of them. This implies that $m = |V| = \Theta\left(n^{3\rho}\right)$. (See [Hal86] for for a formal definition of the graph). In [PP97], an explicit construction for the graph is provided and it is shown that it has $(1, 2/3, 3)$-expansion. With this choice for (V, U_0) we can plug $\epsilon = 2/3$ and $n^\rho = \Theta\left(n^{\tau/3}\right)$ in the formula of Theorem 16, and by choosing $k = O(1)$ large enough, we get access time

$$T = O\left(n^{\frac{2}{3}} + n^{\frac{1}{2}+\frac{2\tau/3-1}{4}}\right) = O\left(n^{\frac{2}{3}} + n^{\frac{2\tau+3}{12}}\right).$$

This yields $T = O\left(n^{2/3}\right)$, for $13/6 \leq \tau \leq 5/2$, and $T = O\left(n^{(2\tau+3)/12}\right)$, for $5/2 \leq \tau \leq 9/2$, which completes the proof. $\square$

Note that the access time of the constructive scheme tends to $O(n)$ as m approaches $n^{9/2}$, a performance that can be obtained through a straightforward scheme.

6. Extension to Other Architectures.

A closer look at the access protocol developed in the previous sections for the mesh reveals that it solely relies upon a recursive decomposition of the network into subnetworks of the same type, and upon ℓ-sorting and (ℓ_1, ℓ_2)-routing primitives. As a consequence, our scheme can be ported to any network topology that exhibits a suitable decomposition into subnetworks, and for which an efficient implementation of the above primitives is available. In this section we briefly discuss the porting of the scheme to the pruned butterfly and to multi-dimensional meshes.

An n-leaf *pruned butterfly*, introduced in [BB95], is a variant of Leiserson's fat-tree [Lei85]. Its coarse structure may be interpreted as a n-leaf complete binary tree where the leaves represent the processor-memory nodes of the machine, the internal nodes represent clusters of routing switches, and where the edges represent channels whose bandwidth doubles every other level from the leaves to the root. More precisely, each subtree of n' leaves is connected to its parent through a channel of capacity $\Theta\left(\sqrt{n'}\right)$. The pruned butterfly is an important interconnection since it is *area-universal* in the sense that it can route any set of messages almost as efficiently as any circuit of similar area.

It follows from the definition that an n-leaf pruned butterfly can be decomposed into 4^i $(n/4^i)$-leaf pruned butterflies connected through channels of capacity $\sqrt{n/4^i}$, a decomposition similar to the one of the mesh employed in our scheme. Moreover, it is shown in [HPP95] that ℓ-sorting and (ℓ_1, ℓ_2)-routing can be performed on the pruned butterfly in the same running time as on the mesh. This immediately implies that both Theorem 1 and 2 also hold for the pruned butterfly

We now consider the extension of the scheme to d-dimensional meshes, with d constant. For $d \geq 3$, a decomposition of an n-node d-dimensional mesh into submeshes is obtained as an immediate generalization of the two-dimensional case. As for the primitives, ℓ-sorting and (ℓ_1, ℓ_2)-routing, with $\ell_1 < \ell_2$, require time $\Theta\left(\ell n^{1/d}\right)$ and $\Theta\left(\ell_2^{1-1/d}(n\ell_1)^{1/d}\right)$, respectively [SK94]. Then, the same argument presented in §4.2 shows that the access protocol can be executed on a d-dimensional mesh in time

$$T = O\left(n^{\epsilon} + n^{\frac{1}{d}}\left(\log n + 2^k n^{\frac{(d-1)(\rho+\epsilon-1)}{d2^k}} + \sum_{i=1}^{k} 2^i 3^{\frac{(d-1)(k-i)}{d}} n^{\frac{2(d-1)(\rho+\epsilon-1)+\epsilon-1}{d2^i}}\right)\right).$$

Let us fix $\rho = 1$ and $\epsilon < 1/(2d-1)$, which, based on Lemma 17, requires $r = \Omega\left(d\tau\right)$ in order to guarantee the existence of the first graph (V, U_0) of the HMOS. Straightforward calculations show that the above formula becomes:

$$T = O\left(2^k n^{\frac{1}{d}+\frac{(d-1)\epsilon}{d2^k}}\right).$$

Arguing as in the proof of Theorem 1 we can prove the following result:

THEOREM 18. *For any constant $\tau \geq 1$, there exists a scheme to distribute $m = n^{\tau}$ shared variables among the local memory modules of an n-node d-dimensional mesh (d constant) with redundancy R so that any n variables can be read/written in time*

$$T = O\left(n^{\frac{1}{d}+\eta}\right)$$

for any constant $\eta > 0$, with $R = O\left(1/\eta^{1.59}\right)$; or in time

$$T = O\left(n^{\frac{1}{d}} \log n\right)$$

with $R = O\left(\log^{1.59} n\right)$.

It has to be remarked that the bandwidth of a d-dimensional mesh increases with d, hence, in order to achieve access time close to the natural $\Omega\left(n^{1/d}\right)$ lower bound, the expansion required of (V, U_0) must also increase with d. For this reason, the graphs for which an explicit construction is currently available do not exhibit sufficient expansion to grant a generalization of Theorem 2; however they can still be used to yield fully constructive schemes with nontrivial $O\left(n^{1/d+\xi_d}\right)$ access time, for suitable constants $\xi_d < (d-1)/d$. The details follow from tedious yet trivial arithmetic manipulations, which are omitted for the sake of brevity.

7. Conclusions.

In this paper, we devised a scheme for implementing a shared address space on a mesh of processor/memory pairs. The scheme enables the processors to read/write any n-tuple of shared variables concurrently and yields a quasi-optimal access time in the worst case. One of the most relevant novelties of our implementation is represented by the hierarchical memory organization scheme, the HMOS, which provides a structured distribution of copies of the shared variables among the memory modules. In particular, the HMOS succeeds in the following objectives, which were not attained by the memory organizations known in the literature: (i) it provides a single mechanism to cope with both memory contention and network congestion. In this fashion, copy selection can be employed to reduce both; (ii) it yields fast access time by using a cascade of bipartite graphs with weak expansion, rather than using one graph of maximum expansion, which greatly simplifies the implementation. Indeed, the HMOS is fully constructive and yields quasi-optimal performance for any memory size $m = O\left(n^{3/2}\right)$, which is sufficient, for example, to run any NC algorithm.

For large memory sizes, the HMOS embodies only one nonconstructive graph of weak expansion.

The design of the HMOS is not specifically cast for the mesh topology. We showed that it can be implemented on the pruned butterfly and on d-dimensional meshes yielding good performance. More generally, our scheme is efficiently portable to any low-bandwidth interconnection where routing takes advantage of partitions of the processors into subnetworks, in the sense that it achieves higher performance by moving messages gradually closer to their destinations through smaller and smaller subnetworks, rather than by sending them directly to their destinations.

A challenging and long-standing open problem remains the construction of bipartite graphs that exhibit good expansion. The availability of explicit constructions and concise representations for such graphs is crucial for attaining simple and efficient deterministic shared memory implementations for all memory sizes. Recent developments in this area [PP97] seem to indicate that the construction of graphs with a linear number of edges and moderate expansion, such as those required in our scheme, be easier than the construction of the highly expanding graphs used in previous schemes. If this is true, our scheme could become a general and constructive tool for the implementation of shared memory on distributed memory machines based on low-bandwidth interconnections.

Finally, we wish to point out that in a recent paper [HPP95], which appeared after the results in the present paper were first presented [PPS94, PP95], a shared memory implementation scheme for the mesh is devised that, through a novel and complex protocol, achieves $O\left(\sqrt{n \log n}\right)$ access time. However, this scheme relies on a nonconstructive graph of maximum expansion, hence it suffers from the same limitations affecting other schemes in the literature, as discussed in the introduction. The paper also proves an $\Omega\left(\sqrt{n \log(m/n^2)/\log\log(m/n^2)}\right)$ lower bound on the access time of any deterministic scheme for implementing $m = \Omega\left(n^2\right)$ shared variables. The lower bound assumes that variables are accessed through a point-to-point protocol, which requires that a processor dispatch a separate message for each copy it wants to update. The assumption is satisfied by the scheme presented in this paper, which implies that our access time is only a sublogarithmic factor away from optimal.

Appendix. In this appendix, we show how to construct a bipartite graph $G = (X, Y)$ which is a subgraph of a (q^d, q)-BIBD with the same number of output nodes, i.e., $|Y| = q^d$, fewer input nodes, say $|X| = m$, $1 \le m < q^{d-1}(q^d - 1)/(q - 1)$, and such that each input $x \in X$ has degree q, as in the original BIBD, and each output $y \in Y$ has degree ρ, with

$$\left\lfloor \frac{qm}{q^d} \right\rfloor \le \rho \le \left\lceil \frac{qm}{q^d} \right\rceil .$$

As explained in §3.1, these subgraphs (with $q = 3$ and m a power of three) govern the assignment of replicas of $(i-1)$ modules to i-modules in the HMOS, for $1 \le i \le k$. The construction is obtained by modifying the one for a (q^d, q)-BIBD given in [PP93].

Let q be a prime power and let $\mathbb{F}_q$ be the finite field with q elements, with its elements represented by the integers $0, 1, \ldots, q-1$. The q^d output nodes of the BIBD are associated with the set of d-dimensional vectors over $\mathbb{F}_q$, and the inputs with the $q^{d-1}(q^d - 1)/(q - 1)$ pairs of vectors of kind

$$\begin{array}{llllllll} (a_{d-2}, & \ldots, & a_h, & 0, & a_{h-1}, & \ldots, & a_1, & a_0) \\ (0, & \ldots, & 0, & 1, & b_{h-1}, & \ldots, & b_1, & b_0), \end{array}$$

where the a_i's and b_i's are elements of the field, and h ranges between 0 and $d-1$. For convenience, each such pair will be denoted by $\chi(h,A,B)$ where A is the integer in $[0,q^{d-1})$ whose representation in base q is $(a_{d-2}\dots a_h a_{h-1}\dots a_1 a_0)$, and B is the integer in $[0,q^h)$ whose representation in base q is $(b_{h-1}\dots b_1 b_0)$. The subgraph G is obtained from this BIBD by taking the same output set and selecting a subset of m inputs as follows. Let $\ell < d$ be the index such that

$$q^{d-1}\frac{q^\ell - 1}{q-1} \le m < q^{d-1}\frac{q^{\ell+1}-1}{q-1},$$

so that

$$m = q^{d-1}\left(\frac{q^\ell-1}{q-1} + w\right) + z, \tag{4}$$

for some w, $0 \le w < q^\ell$ and z, $0 \le z < q^{d-1}$. The m pairs $\chi(h,A,B)$ that we select to represent the nodes of X consist of the union of the three sets X_1, X_2 and X_3 defined below:

$$\begin{aligned}
X_1 &= \left\{\chi(h,A,B) : 0 \le h < \ell,\ 0 \le A < q^{d-1},\ 0 \le B < q^h\right\};\\
X_2 &= \left\{\chi(h,A,B) : h = \ell,\ 0 \le A < q^{d-1},\ 0 \le B < w\right\};\\
X_3 &= \{\chi(h,A,B) : h = \ell,\ 0 \le A < z,\ B = w\}.
\end{aligned}$$

It is easy to verify that $|X_1| + |X_2| + |X_3| = m$.

The edges are defined as follows: the input node

$$\begin{matrix}
(a_{d-2}, & \dots, & a_h, & 0, & a_{h-1}, & \dots, & a_1, & a_0)\\
(0, & \dots, & 0, & 1, & b_{h-1}, & \dots, & b_1, & b_0)
\end{matrix}$$

is adjacent to the q outputs

$$(a_{d-2},\dots,a_h,x,a_{h-1}+x\cdot b_{h-1},\dots,a_1+x\cdot b_1,a_0+x\cdot b_0),$$

for every $x \in \mathbb{F}_q$, where $+$ and $\cdot$ denote the field operations. We now show that the edges in G are evenly distributed among the outputs.

THEOREM 19. *Any node $u \in Y$ is connected to ρ nodes of X, where*

$$\left\lfloor \frac{qm}{q^d} \right\rfloor \le \rho \le \left\lceil \frac{qm}{q^d} \right\rceil.$$

Proof. Let u be associated with the vector $(a_{d-1},\dots,a_0)$. We determine the value of ρ by separately counting the contributions of the nodes in the three subsets X_1, X_2 and X_3. Consider X_1 and fix $h < \ell$. Using the properties of field operations, one can easily show that for any B, $0 \le B < q^h$, there exists exactly one value A such that the node $\chi(h,A,B)$ is connected to u. Therefore, there are exactly $\sum_{h=0}^{\ell-1} q^h = (q^\ell-1)/(q-1)$ nodes of X_1 connected to u. A similar argument shows that exactly w nodes of X_2 are connected to u. Finally, it can be seen that the z nodes of X_3 are connected to qz distinct output nodes, therefore, according to whether u is one of such nodes or not, we know that either $\rho = (q^\ell-1)/(q-1)+w$ or $\rho = (q^\ell-1)/(q-1)+w+1$. By (4) we conclude that

$$\left\lfloor \frac{qm}{q^d} \right\rfloor \le \rho \le \left\lceil \frac{qm}{q^d} \right\rceil.$$

Note that when m is a power of q, it must be $z = 0$ and therefore $\rho = qm/q^d$ for every output node. $\square$

Let $X = U_{i-1}$ be the set of $(i-1)$-modules, and $Y = U_i$ the set of i-modules. Thus, $q = 3$, $d = d_i$ and $m = 3^{d_{i-1}}$. Each $(i-1)$-module is adjacent to the 3 i-modules that contain its $(i-1)$-blocks, and, accordingly, each i-module u is adjacent to the ρ $(i-1)$-modules, each of which has one of its $(i-1)$-blocks stored in u. For each $(i-1)$-module, we must be able to efficiently determine the i-modules that store its $(i-1)$-blocks and the location of each block within the module.

It is easy to establish a bijection between the $(i-1)$-modules and the pairs $\chi(h, A, B)$ in X so that given and index s, $1 \leq s \leq m$, the pair associated with the s-th module is determined in $O(d)$ time. Similarly, a bijection between the i-modules and the d-dimensional vectors over $\mathbb{F}_3$ is easily established. Consider the $(i-1)$-module associated with the pair

$$\chi(h, A, B) = \begin{matrix} (a_{d-2}, & \ldots, & a_h, & 0, & a_{h-1}, & \ldots, & a_1, & a_0) \\ (0, & \ldots, & 0, & 1, & b_{h-1}, & \ldots, & b_1, & b_0). \end{matrix}$$

We adopt the convention that, for $0 \leq j < 3$, the j-th $(i-1)$-block of this module is the ℓ-th item stored in the i-module u, where

$$u = (a_{d-2}, \ldots, a_h, j, a_{h-1} + jb_{h-1}, \ldots, a_1 + jb_1, a_0 + jb_0).$$

and

$$\ell = \frac{3^h - 1}{2} + B.$$

In [PP93] it is proved that the above rule is correct, i.e., no two $(i-1)$-blocks of $(i-1)$-modules are assigned the same location within the same i-module. Moreover, it is not difficult to show that $0 \leq \ell < \rho$.

Observe that the structure of any (U_{i-1}, U_i) is completely determined by the parameter d_i. Since each d_i can be derived from n, we conclude that, in order to represent (U_{i-1}, U_i), a processor needs only know n. From this parameter, the processor can determine the exact location of any copy of any $(i-1)$-module performing $O(\log n)$ operations (arithmetic or in $\mathbb{F}_3$).

Acknowledgments. This paper benefited from discussions with Matteo Frigo, Tim Harris and Franco Preparata. The authors wish to thank the anonymous referees for their valuable comments that helped improve both the presentation and the quality of the paper.

REFERENCES

[AHMP87] H. Alt, T. Hagerup, K. Mehlhorn, and F.P. Preparata, *Deterministic simulation of idealized parallel computers on more realistic ones*, SIAM J. Comput., 16 (1987), pp. 808–835.

[BB95] P. Bay and G. Bilardi, *Deterministic on-line routing on area-universal networks*, J. Assoc. Comput. Mach., 42(1995), pp. 614–640.

[CMS95] 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, Corfu, Greece, 1995, pp. 46–59.

[Hal86] M. Hall Jr., *Combinatorial Theory*, John Wiley & Sons, New York NY, 1986.

[Her96] K.T. Herley, *Representing shared data on distributed-memory parallel computers*, Math. Systems Theory, 29 (1996), pp. 111–156.

[HB94] K.T. Herley and G. Bilardi, *Deterministic simulations of PRAMs on bounded-degree networks*, SIAM J. Comput., 23 (1994), pp. 276–292.

[HPP95] K.T. Herley, A. Pietracaprina, and G. Pucci, *Implementing shared memory on multi-dimensional meshes and on the fat-tree*, in Proc. of the 3rd European Symposium on Algorithms, Corfu, Greece, 1995, pp. 60–74.

[Kun93] M. Kunde, *Block gossiping on grids and tori: deterministic sorting and routing match the bisection bound*, in Proc. of the 1st European Symposium on Algorithms, Bad Hönnef, Germany, 1993, pp. 272–283.

[Lei92] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.

[LMRR94] F.T. Leighton, B. Maggs, A. Ranade, and S. Rao, *Randomized routing and sorting on fixed-connection networks*, J. Algorithms, 17 (1994), pp. 157–205.

[Lei85] C.E. Leiserson, *Fat-trees: universal networks for hardware-efficient supercomputing*, IEEE Trans. Comput., c-34 (1985), pp. 892–901.

[LPP90] F. Luccio, A. Pietracaprina, and G. Pucci, *A new scheme for the deterministic simulation of PRAMs in VLSI*, Algorithmica, 5 (1990), pp. 529–544.

[MV84] K. Mehlhorn and U. Vishkin, *Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories*, Acta Inform., 21 (1984), pp. 339–374.

[PP93] A. Pietracaprina and F.P. Preparata, *An $O(\sqrt{n})$-worst-case-time solution to the granularity problem*, in Proc. of the 10th Symposium on Theoretical Aspects of Computer Science, Würzburg, Germany, 1993, pp. 110–119.

[PP97] ———, *Practical constructive schemes for deterministic shared-memory access*, Theory Comput. Systems, 30 (1997), pp. 3–37.

[PP95] A. Pietracaprina and G. Pucci, *Improved deterministic PRAM simulation on the mesh*, in Proc. of the 22nd International Colloquium on Automata, Languages and Programming, Szeged, Hungary, 1995, pp. 372–383.

[PP97a] ———, *The complexity of deterministic PRAM simulation on distributed memory machines*, Theory Comput. Systems, 30 (1997), pp. 231–247.

[PPS94] A. Pietracaprina and G. Pucci and J.F. Sibeyn, *Constructive deterministic PRAM simulation on a mesh-connected computer*, in Proc. of the 6th Annual Symposium on Parallel Algorithms and Architectures, Cape May, NJ, 1994, pp. 248–256.

[Ran91] A.G. Ranade, *How to emulate shared memory*, J. Comput. System Sci., 42 (1991), pp. 307–326.

[SK94] J.F. Sibeyn and M. Kaufmann, *Deterministic 1-k routing on meshes with application to hot-potato worm-hole routing*, in Proc. of the 11th Symposium on Theoretical Aspects of Computer Science, Caen, France, 1994, pp. 237–248.

[UW87] E. Upfal and A. Widgerson, *How to share memory in a distributed system*, J. Assoc. Comput. Mach., 34 (1987), pp. 116–127.