Implementing Shared Memory on Mesh-Connected Computers and on the Fat-Tree*

Kieran T. Herley
Department of Computer Science
University College Cork
Cork, Ireland
k.herley@cs.ucc.ie

Andrea Pietracaprina, Geppino Pucci
Dipartimento di Elettronica e Informatica,
Università di Padova
Via Gradenigo 6/a
35131 Padova, Italy
{andrea,geppo}@artemide.dei.unipd.it

*This research was supported, in part, by the EC ESPRIT Basic Research Project 9072 (project GEPPCOM: *Foundations of GEneral Purpose Parallel COMputing*). The results in this paper appeared in preliminary form in the *Proceedings of the Third Annual European Symposium on Algorithms (ESA'95)*, pages 60–74, 1995.

Running Head:

Implementing Shared Memory on Meshes

Corresponding Author:

Dr Kieran T. Herley

Department of Computer Science

University College Cork

Cork

Ireland

Phone: +353-21-902134

Fax: +353-21-274390

k.herley@cs.ucc.ie

Abstract

We present deterministic upper and lower bounds on the slowdown required to simulate an (n, m)-PRAM on a variety of networks. The upper bounds are based on a novel scheme that exploits the splitting and combining of messages. This scheme can be implemented on an n-node d-dimensional mesh (for constant d) and on an n-leaf pruned butterfly and attains the smallest worst-case slowdown to date for such interconnections, namely, $O\left(n^{1/d}(\log(m/n))^{1-1/d}\right)$ for the d-dimensional mesh (with constant d) and $O(\sqrt{n\log(m/n)})$ for the pruned butterfly. In fact, the simulation on the pruned butterfly is the first PRAM simulation scheme on an area-universal network. Finally, we prove restricted and unrestricted lower bounds on the slowdown of any deterministic PRAM simulation on an arbitrary network, formulated in terms of the bandwidth properties of the interconnection as expressed by its decomposition tree.

List of Symbols Used

1	one	$\lceil X \rceil$	ceiling of X
l	lower-case ell	$a, b, \ldots, z$	ellipsis
0	zero	$\{a, b, c\}$	curly brackets (indicating set notation)
O	upper-case letter oh		
$O(n)$	upper-case oh symbol for big-oh notation	$< a, b, c >$	angle brackets (delimiting triple)
$\sqrt{n}$	square root	$[k]_j$	square brackets subscripted with j
à	accented lower-case a		
$A \times A$	A times A	∞	infinity
$\Omega(n)$	upper-case Greek omega	$[X, Y)$	left square bracket, X, comma, Y, right round bracket
$<, \leq, >, \geq$	standard inequalities		
$\log n$	logarithm		
$\min\{X, Y\}$	minimum	$\pm$	plus or minus
α	lower-case Greek alpha	Φ	upper-case Greek phi
$\subseteq$	subset symbol	Δ	upper-case Greek delta
$\subset$	proper subset symbol	$\overline{m}$	m bar
$\|U\|$	set size	x'	x prime
λ	lower-case Greek lambda	a^b	a to the b-th power (exponentiation)
σ	lower-case Greek sigma		
$\Theta(n)$	upper-case Greek Theta		
Γ	upper-case Greek gamma		
Σ	upper-case Greek sigma (indicating summation)		
$\hat{S}$	S hat		
$3 \cdot 4$	3 dot 4 (indicating multiplication)		
$\lfloor X \rfloor$	floor of X		
$\bigcup$	union symbol		

1 Introduction

The problem of implementing a shared-memory abstraction on various distributed-memory parallel architectures has been intensively studied over the last decade. Generally, this problem has been referred to as the *PRAM simulation problem* and involves representing the m cells of the PRAM shared memory (called *variables*) among the n processor-memory nodes of the simulating machine in such a way that any n-tuple of distinct cells may be read or written efficiently. The time required to simulate one PRAM step is known as the *slowdown* of the simulation. A number of approaches to this problem, both probabilistic and deterministic, have been investigated for a variety of well-known architectures such as the complete interconnection, the mesh of trees, the butterfly, as well as a variety of expander-based architectures, among others.

We will not attempt to summarize the extensive literature on this problem here but only quote those results that relate directly to our work, and refer the interested reader to [PPS94] for a recent and comprehensive summary of further work on this topic. Building on earlier work of Upfal and Wigderson [UW87], Alt *et al.* [AHMP87] presented a deterministic scheme to simulate a PRAM with n processors and m variables (called an (n, m)-PRAM) on an n-node *Module Parallel Computer* (MPC), an architecture in which each node includes both a processor and a private memory module accessible only to that processor, and in which the nodes are connected by a crossbar that allows each node to transmit or receive one message per step. Their scheme employs the following copy-based method for the representation of the PRAM variables, which most of the deterministic simulation algorithms, including this present work, adopt. Specifically, each variable is represented by a set of copies, whose size $2c - 1$ is logarithmically related to n and m, and each copy consists of a value and a time-stamp. The copies are distributed carefully among the memory modules of the simulating machine. To write a variable, at least c of its copies are overwritten to reflect the intended value and the time of writing. To read a value, at least c copies are inspected. This set of c copies read contains at least one of the copies most recently written, which is readily identifiable by virtue of its time-stamp. Alt *et al.* show that for a suitable distribution of the copies among the nodes of the machine, any n-tuple of variables may be accessed (read or written) in $O(\log m)$ time.

The above scheme can be ported to an arbitrary network by simulating each MPC step using standard techniques such as routing and sorting. In particular, this approach yields a simulation with slowdown $O\left(n^{1/d} \log m\right)$ on an n-node d-dimensional mesh ($d = O(1)$)

and a simulation with slowdown $O(\sqrt{n}\log m)$ on an n-leaf pruned butterfly, which are the interconnections that we consider in this paper. In [AHMP87], it was also observed that a simple PRAM simulation for the two-dimensional mesh with an optimal slowdown of $O(\sqrt{n})$ is indeed possible. Unfortunately, this simulation requires up to n copies per variable, resulting in an unacceptable memory blow-up. Moreover, the method does not extend to higher-dimensional meshes and other interconnections.

Most of the deterministic simulations that appear in the literature, including those of this paper, rely on memory distributions that are built upon certain expander-based graphs whose existence can be proved, but for which no efficient construction is known. Recently, Pietracaprina *et al.* [PPS94, PP95] have studied deterministic simulations based entirely on explicitly constructible structures. By resorting to a complex hierarchical arrangement of constructible, mildly-expanding graphs, they achieve $O(\sqrt{n}\log n)$ slowdown on an n-node mesh for memories of $O(n^{1.5})$ size, using $O\left(\log^{1.59} n\right)$ copies per variable. In this paper, our focus is on slowdown rather than constructiveness. By employing more powerful expander-based structures, we achieve a better slowdown than that of [PP95] at a lower level of redundancy (copies per variable).

It might appear, at least at first glance, that updating c copies apiece for n variables must involve the physical movement of cn distinct packets across the entire network, which on the $\sqrt{n} \times \sqrt{n}$ mesh would require $\Omega(c\sqrt{n})$ time. In this paper, we devise a novel *splitting/combining* technique to circumvent this difficulty, based on the following idea. If a processor p wishes to send the same packet to nodes x and y that are "distant" from p but "close" to one another, then rather than dispatch a separate packet for each, it may be more efficient to dispatch a single message to some "intermediate" node z close to both x and y. At node z, the original packet is then made into two replicas which are forwarded to x and y separately. A careful implementation of this idea leads to the following result.

Theorem 1 *For any $m \leq 2^{\Theta\left(n^{1/(d+1)}\right)}$ there exists a scheme to simulate an (n, m)-PRAM on an n-node d-dimensional mesh (d constant) with worst-case slowdown $O\left(n^{1/d}(\log(m/n))^{1-1/d}\right)$, using $O(\log(m/n))$ copies per variable and $O\left((m/n)\log^5(m/n)\right)$ storage per node.*

In order to implement the splitting/combining strategy outlined above, the scheme relies on a recursive decomposition of the mesh and on efficient algorithms for k-sorting, where each processor initially holds k packets, and for k-relation routing, where each processor sends and receives at most k packets. Theorem 1 implies that our simulation scheme incurs a slowdown which is a factor $O\left((\log(m/n))^{1/d}\right)$ smaller than the one obtainable by porting the

MPC algorithm of [AHMP87] on an n node d-dimensional mesh. We want to remark that the (exponential) upper bound on the memory size m in Theorem 1 is placed to avoid that the cost of sequential bookkeeping operations such as local sorting or counting dominate the overall running time of the simulation algorithm. Such a bound on m is not needed if a cost model which accounts for interprocessor communication only was adopted, as customary for network algorithms [Kun93].

The n-leaf pruned butterfly [BB95] (described later) is an area-universal network that is a variant of Leiserson's fat-tree [Lei85]. Although quite different from the two-dimensional mesh in terms of the details of its structure, it is sufficiently similar in its bandwidth characteristics to support the key operations upon which our simulations rely with comparable efficiency. By providing novel sorting and routing primitives for this network, and by using its natural decomposition into subtrees, we are able to implement the above simulation scheme with the same slowdown achieved for the two-dimensional mesh, thereby obtaining the first PRAM simulation on an area-universal network. The result is stated in the following theorem.

Theorem 2 *For any $m \leq 2^{\Theta\left(n^{1/3}\right)}$ there exists a scheme to simulate an (n, m)-PRAM on an n-leaf pruned butterfly with worst-case slowdown $O\left(\sqrt{n \log(m/n)}\right)$, using $O\left(\log(m/n)\right)$ copies per variable and $O\left((m/n)\log^5(m/n)\right)$ storage per node.*

Lower bounds on the slowdown of PRAM simulations on bounded-degree networks have been presented in a number of studies [AHMP87, KU88, HB94]. All such bounds, however, apply to the entire class of such networks, and cannot be specialized to the characteristics of a given topology. For example, in [HB94] the authors show an $\Omega\left(\log^2(m/n)/\log\log(m/n)\right)$ lower bound on the slowdown required to simulate a PRAM step on *any* bounded degree network, which is too weak for our purposes, since a trivial $\Omega\left(n^{1/d}\right)$ lower bound may easily be obtained for d-dimensional meshes based on diameter limitations. An $\Omega\left(\sqrt{n}\right)$ bound holds for the pruned butterfly based on straightforward bandwidth considerations. In this paper, we present the first lower bound argument that takes into account the characteristics of the individual network. To capture the properties of the network topology, the bound exploits the notion of decomposition tree [BL84, Lei85], which provides a partition of the network into disjoint regions of limited bandwidth.

As in all previous works, the lower bound is proved under the *point-to-point* assumption, which requires that a processor updating a number of copies of a variable dispatch a separate message for each copy. When specialized to d-dimensional meshes and to the pruned butterfly, our lower bound technique yields the following results.

Theorem 3 *Let $m \geq 16n$. For every $T \geq 2m/n$, there exists a T-step (n,m)-PRAM program, whose point-to-point, on-line simulation requires time*

$$\Omega\left(Tn^{\frac{1}{d}} \min\left\{\left(\frac{\log(m/n)}{\log\log(m/n)}\right)^{1-\frac{1}{d}}, n^{\frac{d^2-2d+1}{2d^2-d}}\right\}\right)$$

on an n-node d-dimensional mesh (with d constant).

Theorem 4 *Let $m \geq 16n$. For every $T \geq 2m/n$, there exists a T-step (n,m)-PRAM program, whose point-to-point, on-line simulation requires time*

$$\Omega\left(T \min\left\{\sqrt{n\frac{\log(m/n)}{\log\log(m/n)}}, n^{\frac{2}{3}}\right\}\right)$$

on an n-leaf pruned butterfly.

Unfortunately, the point-to-point assumption upon which Theorems 3 and 4 and the other works in the literature rely, precludes the splitting and combining of messages. As a consequence, the above lower bounds do not apply to our simulations directly. However, we are able to prove similar bounds in an unrestricted model by limiting the total level of redundancy used to represent the variables. Such bounds show that our simulations use an amount of redundancy which is only a doubly logarithmic factor higher than the minimum redundancy needed to achieve the same slowdown. Specifically, we have the following result.

Theorem 5 *Let $m \geq 16n$. For every $T \geq 2m/n$ and every constant $\alpha \geq 1$, there exists a T-step (n,m)-PRAM program whose on-line simulation on an n-node d-dimensional mesh requires time*

$$\Omega\left(Tn^{\frac{1}{d}} \min\left\{\left(\frac{\log(m/n)}{\log\log(m/n)}\right)^{\alpha\left(1-\frac{1}{d}\right)}, n^{\frac{\alpha}{\alpha+d/(d-1)}\left(1-\frac{1}{d}\right)}\right\}\right)$$

if the total number of copies used to represent the m PRAM variables in the local memory modules is mr, with

$$r \leq \frac{1}{8\alpha} \min\left\{\frac{\log(m/n)}{\log\log(m/n)}, n^{\frac{1}{\alpha+d/(d-1)}}\right\}.$$

The bound with $d = 2$ also applies to the pruned butterfly.

The rest of the paper is organized as follows. Section 2 discusses the distribution of the copies among the memory modules and the properties required of the graph representing the

memory map. In Section 3, the simulation algorithm for the two-dimensional mesh is presented. The algorithm consists of two phases, copy-selection and routing, which are described in Subsections 3.1 and 3.2, respectively. This scheme is extended to higher-dimensional meshes in Section 4 and to the pruned butterfly in Section 5. Section 6 shows how the space bounds quoted in Theorems 1 and 2 may be achieved. Section 7 presents the lower bound results discussed above. Section 8 concludes the paper with some final remarks and indicates future research directions.

2 Memory Organization

Consider the simulation of an (n, m)-PRAM on an n-node machine and suppose that each variable is replicated into $2c - 1$ copies, for a suitable integer c. It is convenient to model the distribution of copies among the nodes of the machine by means of a bipartite graph $G = (U, V; E)$, where U represents the set of variables, V the set of processor-memory nodes of the simulating machine, and $2c-1$ edges connect each variable to the distinct nodes storing its copies. In the following we will denote by $E(S)$ the set of edges in E incident on a set $S \subseteq U$. Note that there is a one-to-one correspondence between $E(S)$ and the set of all copies of variables in S.

Let $S \subseteq U$ and $F \subseteq E(S)$. When F contains exactly k edges incident on each $s \in S$ we call F a *k-bundle* for S. Also, we denote by $\Gamma_F(S)$ the subset of V reached by edges in F. A vertex $v \in V$ is said to be *q-congested with respect to F* if more than q edges in F are incident on v. Finally, the *congestion* of F is the maximum value q such that there is a vertex in V that is q-congested with respect to F.

Recall that our simulations adopt the majority protocol, which requires that at least c copies be accessed in order to complete a read or a write. Equivalently, in graph-theoretic terms, if we wish to access a set S of variables, then we must select a c-bundle for S. Since the congestion of a c-bundle models the maximum number of physical copies that have to be accessed sequentially by some individual node in the underlying machine, it is desirable to access a c-bundle with low congestion. The existence of c-bundles of low congestion is intimately related to the expansion properties of the graph G. This motivates the following definition [HB94] that characterizes a class of graphs called *generalized expanders* that make good memory organizations.

Definition 1 *A bipartite graph $G = (U, V; E)$ with $|U| = m$ and $|V| = n$, and with each*

node in U having degree d is a (λ, d, c, σ)-generalized expander if, for every $S \subseteq U$ such that $|S| \leq \sigma n$ and for every c-bundle F of S, $|\Gamma_F(S)| \geq \lambda c|S|$.

We say that a generalized expander is *smooth* if the maximum degree of any node in V is $\Theta(|E|/|V|)$. Herley and Bilardi [HB94] have established the existence of certain generalized expanders using counting techniques similar in spirit to those found in the seminal work of Upfal and Widgerson [UW87]. A minor variation of this result (guaranteeing smoothness) is quoted below.

Theorem 6 *For every n and m, with $m \geq n$, there exists a smooth $(\lambda, 2c-1, c, 1/(2c-1))$-generalized expander $G = (U, V; E)$ with $|U| = m$, $|V| = n$, $\lambda = \Theta(1)$ and $c = \Theta(\log(m/n))$.*

The graph of Theorem 6 will govern the memory organization of our simulations. We shall see that such a graph has the desirable property that every set $S \subseteq U$ of size at most n has a c-bundle of low congestion. Moreover, this c-bundle can be constructed efficiently.

Let $S \subseteq U$ be the set of variables to be accessed. The simulation algorithms described in the next sections construct a c-bundle for S starting from $E(S)$ and applying a sequence of *whittling steps*. Each whittling "prunes" the set of edges by selecting c edges apiece for some of the variables in S, and discarding the remaining $c-1$. At the beginning of a whittling step, a variable is said to be *alive* if the c edges for the variable have not been selected yet, and *dead* otherwise. The sequence terminates when all variables are dead, at which point we are left with the desired c-bundle. The variables to whittle at each step are chosen to ensure that the degree of the final c-bundle will not exceed a fixed congestion q, whose value will be specified later.

For $i \geq 1$, let $S_i \subseteq S$ denote the set of live variables and $F_i \subseteq E(S)$ the residual set of edges at the beginning of the i-th whittling step. Initially, $S_1 = S$ and $F_1 = E(S)$. Conceptually, the i-th whittling step identifies a set W_i of "congested" nodes and selects c edges apiece for as many live variables as possible without touching nodes in W_i. More formally, we say that $x \in S_i$ is *confined to W_i under F_i* if x has c or more copies in F_i stored in nodes of W_i. In the i-th whittling step, for each $x \in S_i$ which is not confined to W_i under F_i, we select c edges not incident on W_i and remove the remaining $c-1$ from F_i. This operation will be referred to as *whittling of S_i with respect to W_i*.

Definition 2 *Let $S \subseteq U$, $|S| \leq n$. A* q-whittling sequence *of length t for S is a sequence (S_1, F_1, W_1), (S_2, F_2, W_2), ..., (S_t, F_t, W_t) such that*

- *$S_1 = S$, $F_1 = E(S)$, and $W_1 \subseteq V$ is a set of at most $(2c-1)n/q$ nodes including all nodes that are q-congested with respect to $E(S)$;*
- *For $i > 1$, $S_i \subset S_{i-1}$ and $F_i \subset F_{i-1}$ are, respectively, the set of live variables and the set of residual edges left after whittling S_{i-1} with respect to W_{i-1} and W_i is the set of q-congested nodes with respect to F_i;*
- *$S_t = W_t = \emptyset$ and F_t is a c-bundle for S.*

Note that in the above definition each W_i, with $i > 1$, contains only nodes that are q-congested with respect to F_i, while W_1 may include an additional (small) number of uncongested nodes. The rationale behind this asymmetry will become clear later in the paper. It is also easy to see that F_t, the final c-bundle for S, has congestion at most q. The following lemma characterizes the rate at which the variables "die" during a whittling sequence.

Lemma 1 *Let S be a set of at most n variables, and $q \geq 4c/\lambda = \Theta(\log(m/n))$. Then, for any q-whittling sequence of length t, (S_1, F_1, W_1), (S_2, F_2, W_2), $\ldots$, (S_t, F_t, W_t), we have $|S_i| \leq n/(2c-1)^{i-1}$ for $i \geq 1$. Therefore $t = O(\log n / \log\log(m/n))$.*

Proof: The proof proceeds by induction on i. The basis $i = 1$ is trivial. For $i = 2$, consider the set S_2 and assume that $|S_2| > n/(2c-1)$. Since all variables in S_2 are confined to W_1, we can choose an arbitrary subset $S' \subseteq S_2$ of exactly $n/(2c-1)$ variables, and a c-bundle $F' \subseteq F_1$ for S', with $\Gamma_{F'}(S') \subseteq W_1$. Then, by the expansion properties of our memory organization, we obtain

$$|W_1| \geq \left|\Gamma_{F'}(S')\right| \geq \lambda c|S'| = \frac{\lambda cn}{(2c-1)} > \frac{(2c-1)n}{q},$$

a contradiction. Finally, suppose that $i \geq 3$, and note that all the edges in F_{i-1} relative to the set $S - S_{i-1}$ of dead variables at the beginning of Step $i-1$ cannot be incident on nodes of W_{i-1}, since we never pick edges for c-bundles out of those incident on q-congested nodes. Therefore, the congestion of nodes in W_{i-1} is entirely caused by copies of variables in S_{i-1}, whence

$$|W_{i-1}| \leq \frac{(2c-1)|S_{i-1}|}{q} \leq \frac{\lambda}{2}|S_{i-1}|. \tag{1}$$

On the other hand, being confined to W_{i-1}, all variables in S_i have a c-bundle in W_{i-1}, and $|S_i| \leq |S_2| \leq n/(2c-1)$, hence, by the expansion properties of the memory map,

$$|W_{i-1}| \geq \lambda c|S_i|. \tag{2}$$

The bound for S_i follows by combining Inequalities (1) and (2) and applying the inductive hypothesis. □

3 Simulation on the Mesh

We now consider the simulation of an (n, m)-PRAM on a $\sqrt{n} \times \sqrt{n}$ mesh. In the subsequent section we will sketch the (relatively minor) modifications required to extend the simulation to meshes of higher dimensions. For notational convenience we will assume that n is a power of four. Each mesh node simulates the activities of a distinct PRAM processor. Specifically, a node comprises a processor, capable of executing a standard repertoire of logical and arithmetic operations, and a local memory directly accessible to that processor alone. Each processor-memory node is connected to its immediate neighbours in the mesh by means of bidirectional wires capable of transmitting a single $O(\log m)$-bit quantity per step.

Recall that the distribution of the PRAM variables (set U) among the memory modules of the mesh (set V) is governed by a $(\lambda, 2c-1, c, 1/(2c-1))$ generalized expander $G = (U, V; E)$ with $|U| = m$ and $|V| = n$, $\lambda < 1$ (λ constant), and $c = \Theta(\log(m/n))$. It is assumed for the moment that each mesh node holds a copy of a read-only table that encodes the structure of the memory organization. In other words, the i-th entry of the table records the locations of the copies of the i-th PRAM variable. This naive representation of the memory map requires $O(m \log(m/n))$ storage per node. In Section 6, we will show how this table may be represented in a distributed fashion, taking only $O\left((m/n)\log^5(m/n)\right)$ storage per node. Note that the total storage of the simulating machine will then be only a polylogarithmic factor away from the size of the PRAM memory.

We simulate a PRAM program by separately simulating its individual steps. Consider the simulation of an arbitrary PRAM step, and let $S \subset U$ denote the set of variables accessed in the step. We assume that each processor accesses a distinct variable (i.e., $|S| = n$), thus restricting our attention to the simulation of the EREW PRAM. Standard techniques can be employed to extend the simulation to the other PRAM variants with no performance loss (e.g., see [HB94]). We assume that the address of the variable referenced by the i-th PRAM processor during the step is made known to the corresponding mesh processor at the start of the simulation. Each mesh processor creates a packet, referred to as a *v-packet*, containing (i) the processor's id; (ii) the name of the PRAM variable it wishes to access and the type of operation (read/write); (iii) a data field for the value to be read/written; and (iv) a bit vector of length $2c - 1$, whose entries correspond to the copies of the variable and are initially set

to zero. Let $\hat{S}$ denote the set of v-packets created by the mesh processors. The simulation consists of the following two *phases*.

Copy Selection Phase A c-bundle for S of congestion at most $q = 4c/\lambda$ is selected. The bit vector of each packet in $\hat{S}$ is set to encode which copies of the corresponding variable are in the c-bundle and which are not.

Access Phase For each variable in S, a copy of the corresponding v-packet is routed to each mesh node containing a selected copy of that variable. Each mesh node performs the accesses relative to the packets it receives and, in the case of reads, sends the accessed values back to the requesting processors.

In the two subsections that follow we will describe the implementation of these two phases. Most of our algorithms will involve the movement or manipulation of sets of packets. In turn, such activities are based on two algorithmic primitives: *k-sorting* and *k-relation routing*. Given a set of packets distributed among the processors of an n'-node mesh, so that each node holds at most k packets, k-sorting is used to rearrange the packets so that node 1 holds the packets with the k smallest keys, node 2 holds the next k smallest keys, and so on. For any value of k and any numbering of the nodes, this can be accomplished in $O\left(k\left(\log k + \sqrt{n'}\right)\right)$ on the two-dimensional mesh, where the term $k \log k$ arises from the need to initially sort the groups of k keys locally at each node. It has to be remarked that in our simulation algorithm, k-sorting (with nonconstant k) is always applied for $k = O(\log(m/n))$ and $n' = \Omega(n/\log(m/n))$. Therefore, for values of m within the bound stated in Theorem 1, the time for the initial local sorting will never dominate the overall running time. In other words, in our scenario k-sorting will always require $O\left(k\sqrt{n'}\right)$ time.

The k-relation routing problem involves routing a set of packets subject to the constraint that no node is the source or destination of more than k packets. Again, this can be done in $O\left(k\sqrt{n'}\right)$ time on an n'-node mesh. Both algorithms can be found in [Kun93].

3.1 Copy Selection Phase

The copy selection is accomplished by performing the whittlings implied by a q-whittling sequence $\{(S_i, F_i, W_i) \ : \ 1 \leq i \leq t\}$, for some $t = O(\log n/\log\log(m/n))$, as defined in Section 2. Note that each whittling could be performed easily by representing the copies of the live variables as individual packets and then employing sorting, prefix and routing in order to select the packets relative to the copies to be included in the c-bundle. However,

the n variables would initially account for $(2c-1)n$ packets, and manipulating such a large set of packets can be expensive: for example, just sorting the packets would require $O(c\sqrt{n})$ time, which is too costly for our purposes. For this reason, the copy selection phase is broken into two *stages*. The first stage performs the first whittling step, using more complex but faster techniques. Since only a relatively small number of variables (less than $n/(2c-1)$) will participate in the second stage, the remaining whittlings can be implemented using the simple technique described above.

Stage 1 In the first stage, we perform the whittling of $S_1 = S$ with respect to a certain set W_1 (specified later) which includes all nodes q-congested relative to $F_1 = E(S)$, with $q = 4c/\lambda$. In what follows, we will regard the mesh as being partitioned into s disjoint submeshes of size $\sqrt{n/s} \times \sqrt{n/s}$, which we will call *cells*. The quantity s, which we assume to be a power of four greater than $\log c$, will be determined by the analysis.

Recall that $\hat{S}$ denotes the set of v-packets relative to the n variables to be accessed. The stage consists of the following steps.

1. Create a replica $\hat{S}_C$ of $\hat{S}$ in each cell C of the mesh.
2. Independently within each cell C do the following:
   (a) Determine the *degree* of each packet in $\hat{S}_C$ with respect to C, that is, the total number of copies of the corresponding variable which reside within C. Compute the total degree of all packets in the cell as the sum of the individual degrees.
   (b) If the total cell degree is less than $q(n/s)$, then generate a set of *c-packets* $\hat{R}_C$ containing one c-packet for each copy of a variable in S residing within C. Each c-packet contains the name of the variable and the node in C storing the copy. This node is called the *target* of the c-packet. Furthermore, c-packets with the same target are said to be *competitors*, while c-packets relative to the same variable are said to be *companions*. Note that a c-packet may have companions belonging to several cells. Finally, we call $\hat{R}$ the set obtained as the union over all cells C of the set $\hat{R}_C$.
   (c) Determine how many competitors each c-packet has. A c-packet with q or fewer competitors is deemed *accessible*, and *inaccessible* otherwise.
3. For those variables with c or more accessible c-packets, select c copies and update the bit-map in the appropriate v-packet in $\hat{S}$ to reflect which copies have been selected. The

bit-vectors for all other v-packets should remain unchanged *i.e.* all bits should remain at zero.

We can easily see that the first stage executes the whittling of S with respect to a set W_1 which comprises all mesh nodes that store more than q copies of variables in S (*i.e.,* the q-congested nodes), plus those nodes belonging to cells with total degree at least $q(n/s)$. Therefore, the nodes in W_1 account for at least $q|W_1|$ copies. Since there are $(2c-1)n$ copies of variables, we must have $|W_1| \leq (2c-1)n/q$, as required by the definition of q-whittling sequence.

Lemma 2 *Stage 1 can be implemented on the mesh in time* $O\left(sc + \sqrt{ns} + q\sqrt{n/s}\right)$.

Proof: Step 1 is executed as follows. Partition the $\sqrt{n} \times \sqrt{n}$ mesh into four $\sqrt{n/4} \times \sqrt{n/4}$ submeshes (quadrants), and each of these in turn into four submeshes of size $\sqrt{n/16} \times \sqrt{n/16}$ (subquadrants), and so on until be have tessellated the mesh into submeshes of size $\sqrt{n/s} \times \sqrt{n/s}$, *i.e.* into individual cells. The replication of $\hat{S}$ in the individual cells, reflects this recursive decomposition of the mesh. First, the set $\hat{S}$ is replicated in each of the four quadrants. This is achieved by having the four mesh nodes that occupy the same relative position in the four quadrants send each other a copy of the packets they hold. At this point, each node holds four packets and each quadrant holds a replica of $\hat{S}$. Each quadrant independently replicates its copy of $\hat{S}$ into its four constituent subquadrants in the same manner, and so on. At the start of the i-th iteration of this process, each node of a $\sqrt{n/4^{i-1}} \times \sqrt{n/4^{i-1}}$ submesh holds 4^{i-1} packets that it must send to three other nodes within this submesh. This is a simple routing pattern in which each node is the source and destination of $3 \cdot 4^{i-1}$ packets (*i.e.,* an instance of the $3 \cdot 4^{i-1}$-relation routing problem), and so can be completed in $O\left(4^{i-1}\sqrt{n/4^{i-1}}\right) = O\left(\sqrt{n}2^{i-1}\right)$ time. Since there are $(1/2)\log s$ iterations in all, we see that the time required to complete Step 1 is $O\left(\sum_{i=1}^{(1/2)\log s} \sqrt{n}2^{i-1}\right) = O\left(\sqrt{ns}\right)$.

Step 2.a involves calculating the degree of each variable packet, hence requires $O(c)$ time per packet and $O(sc)$ time per node, since each node hold s packets. The total cell degree for C can easily be computed within each cell from the individual degrees in $O\left(\sqrt{n/s}\right)$ time.

A straightforward implementation of Step 2.b, where each node generates the appropriate number of c-packets for each v-packet it holds, would result in an unbalanced distribution of the c-packets among the nodes, which would in turn make the execution of subsequent steps more expensive. Therefore, we resort to the following more careful implementation of this step. Within each cell C, partition $\hat{S}_C$ into *degree classes* $\hat{S}_C^{(i)}$ for $0 \leq i \leq \lceil \log(2c-1) \rceil$,

where $\hat{S}_C^{(i)}$ contains v-packets of variables with at least 2^i and at most $2^{i+1}-1$ copies in C. (We ignore variables of degree zero and note that no v-packet can have degree greater than $2c-1$.) Redistribute the v-packets of each degree class so that each node of C receives at most $\lceil |\hat{S}_C^{(i)}| s/n \rceil$ v-packets in Class i. This redistribution can be performed by first determining the destinations of packets in all classes using $\lceil \log(2c-1) \rceil$ prefix operations (one per degree class), and then invoking a routing step within C in which each node is the source of at most s packets and the destination of at most $\sum_{i=1}^{\lceil \log(2c-1) \rceil} \lceil |\hat{S}_C^{(i)}| s/n \rceil = O(s + \log c) = O(s)$ packets. This requires overall time $O\left(\sqrt{n/s} \log c + \sqrt{ns}\right) = O(\sqrt{ns})$.

After the above redistribution of the v-packets, each node examines each v-packet it holds and generates the appropriate number of c-packets for the corresponding variable. Since each node receives no more than $\lceil |\hat{S}_C^{(i)}| s/n \rceil$ v-packets from the i-th degree class and each such packet may result in up to $2^{i+1}-1$ c-packets, it follows that each node may hold up to $\sum_{i=1}^{\lceil \log(2c-1) \rceil} \lceil |\hat{S}_C^{(i)}| s/n \rceil (2^{i+1}-1)$ c-packets. Now, since $\sum_{i=1}^{\lceil \log(2c-1) \rceil} |\hat{S}_C^{(i)}| (2^{i+1}-1) \leq 2|\hat{R}_C| < 2q(n/s)$ by assumption, we can conclude that each node generates $O(q+c) = O(q)$ c-packets. Putting it all together, Step 2.b can be completed in time $O(\sqrt{ns} + q)$ time.

Turning to Step 2.c, we can count the competitors of each packet within each cell by first sorting the c-packets by target, to group competitors together, and then using parallel prefix to determine which packets are accessible and which are inaccessible. If a c-packet is deemed accessible, then the appropriate entry in the bit vector of the corresponding v-packet is set. Since each node initially holds $O(q)$ packets and the cell has diameter $\sqrt{n/s}$, this step requires $O\left(q\sqrt{n/s}\right)$ time.

As for Step 3, we gather the accessibility information gained for each variable in different cells by "coalescing" the replicas of the v-packets created in Step 1. When two replicas of a v-packet are coalesced, a single v-packet results, whose bit-vector is obtained as the bitwise OR of the two bit-vectors associated to the replicas. The structure of the gathering process follows the reverse pattern of the replication process of Step 1, and is executed in the same time. At the end of the process, each processor checks the bit-vector of its v-packet. If the bit-vector contains more than c 1-bits, then c of these, chosen arbitrarily, are retained (corresponding to a c-bundle), while the others are reset to 0. Otherwise, all bits are reset to 0.

By combining the complexities of all the steps, we conclude that the running time of the first stage is $O\left(sc + \sqrt{ns} + q\sqrt{n/s}\right)$. □

Following the completion of the first stage the v-packets are back to their originating

processors and encode a c-bundle for all variables except for the set S_2 of variables confined to W_1. Since $|W_1| \leq (2c-1)n/q$, by Lemma 1 it follows that $|S_2| \leq n/(2c-1)$. Stage 2 will select a c-bundle for S_2 by performing all the remaining whittlings of the q-whittling sequence.

Stage 2 Since $|S_2| \leq n/(2c-1)$, the live variables account for at most n copies, *i.e.* at most one per mesh node. As a consequence, we can perform the remaining $t-1$ whittlings using standard 1-sorting, prefix and 1-routing primitives and without exceeding our target time performance. In fact, in each whittling step the number of live variables decreases geometrically, thus we can execute the whittlings within smaller and smaller submeshes so that the cost of the aforementioned primitives also decreases geometrically.

Let S_i denote the live variables at the beginning of the i-th whittling step, and recall that by Lemma 1 we have $|S_i| \leq n/(2c-1)^{i-1}$. We define a sequence of submeshes $M_2, M_3, \cdots, M_t$ that are nested one inside the other, where M_i is a $\sqrt{n/(2c-1)^{i-2}} \times \sqrt{n/(2c-1)^{i-2}}$ submesh. (For concreteness, we will assume that M_{i+1} occupies the lower left hand corner of M_i.) Note that M_2 is the entire mesh. Stage 2 consists of $t-1$ iterations numbered, for convenience, from 2 to t. Iteration i performs the i-th whittling step in the whittling sequence and is executed entirely within M_i. In particular, for $i \geq 2$ at the beginning of Iteration i, the v-packets corresponding to the live variables (set S_i) are distributed among the nodes of M_i, with at most one v-packet per node. At the end of the iteration, the v-packets corresponding to dead variables in $S - S_{i+1}$, are evenly distributed among the nodes outside M_{i+1} (for notational convenience, we assume that M_{t+1} is an "empty" mesh). Iteration i is implemented as follows.

1. For each variable in S_i create $2c-1$ c-packets and distribute the c-packets among the nodes of M_i, assigning at most one packet to each node.

2. For each c-packet, determine how many competitors it has. A c-packet with at most q competitors is said to be *accessible* and is said to be *inaccessible* otherwise.

3. For each variable x in S_i determine how many of the associated c-packets are accessible. If c or more are accessible, set the bit-vector positions in x's v-packet corresponding to the first c accessible copies of that variable; otherwise reset all bits in the bit-vector for x to zero. In the former case x becomes dead while in the latter x is alive and will belong to S_{i+1}.

4. Delete all c-packets. Route the v-packets corresponding to the dead variables in S_i to

distinct nodes of $M_i - M_{i+1}$, while those corresponding to variables that remain alive, to distinct nodes of M_{i+1}.

It is easy to see that the above steps perform the whittling of S_i with respect to W_i. Note that at most $|S_i| \leq n/(2c-1)^{i-1}$ die during Iteration i and that at most $|S_{i+1}| \leq n/(2c-1)^i$ variables remain alive at its conclusion. Since M_i contains $n/(2c-1)^{i-2}$ nodes and M_{i+1} contains $n/(2c-1)^{i-1}$ nodes, if $c \geq 3$ there are always enough nodes in $M_i - M_{i+1}$ and in M_{i+1} to implement the last step of Iteration i.

Lemma 3 *Stage 2 can be implemented on the mesh in time* $O(\sqrt{n})$.

Proof: Iteration i requires a constant number of prefix, 1-sorting and 1-relation routing on M_i, which all require time $O\left(\sqrt{n/(2c-1)^{i-2}}\right)$, therefore the combined cost of the $t-1$ iterations is

$$O\left(\sum_{i=2}^{t}\sqrt{\frac{n}{(2c-1)^{i-2}}}\right) = O(\sqrt{n}).$$

□

Once the sequence of whittlings has been completed, the information on which copies have been selected is encoded in the bit-vectors of the various v-packets that lie scattered among the nodes of the mesh, with at most 2 packets per node (at most one from Stage 1 and at most one from Stage 2). Finally, the v-packets are sent back to their originating processors in $O(\sqrt{n})$ time.

Combining the contributions of the two stages, we obtain the overall running time for the copy selection phase.

Theorem 7 *A c-bundle of congestion $O(\log(m/n))$ for an arbitrary set of n variables can be selected in time $O\left(\sqrt{n\log(m/n)}\right)$ on the mesh.*

Proof: Choose $s = q = 4c/\lambda = \Theta(\log(m/n))$ and note that for $m \leq 2^{\Theta\left(n^{1/3}\right)}$, as required in Theorem 1, $sc = O\left(\sqrt{n\log(m/n)}\right)$. Then, the theorem follows by adding up the complexities of Stage 1 and Stage 2 given in Lemmas 2 and 3, respectively. □

3.2 Access Phase

After copy selection, the bit-vectors of the v-packets in $\hat{S}$ encode a c-bundle of congestion at most q for the set of variables to be accessed. The actual access is performed using a protocol similar to Stage 1 of copy selection.

1. Let $s = q = 4c/\lambda = \Theta(\log(m/n))$. Create a replica $\hat{S}_C$ of $\hat{S}$ in each cell C of n/s nodes of the mesh.

2. Independently within each cell C do the following:

   (a) Generate a set $\hat{R}_C$ of c-packets containing one c-packet for each selected copy of a variable in S residing within C. Now, each c-packet contains the identity of the node in C storing the copy, plus all the fields of the corresponding v-packet, with the exception of the bit-vector.

   (b) Route each c-packet in $\hat{R}_C$ to its target within C.

   (c) Each node in C performs the memory accesses associated with the received c-packets. In case of writes, the value of the copy is set to the one carried by the c-packet and timestamped with the current PRAM step. In case of reads, the data field of the c-packet is loaded with the value and time-stamp of the referenced copy.

   (d) Route c-packets carrying read requests back to the nodes where they were generated. For each read request, the data field of the corresponding v-packet is loaded with the data field of the c-packet carrying the most recently updated time-stamp.

3. Complete the read accesses by carrying back to each originating node the (replica of the) v-packet carrying the most recent time-stamp.

Theorem 8 *The memory accesses relative to the c-bundle determined by the copy selection phase can be performed in time* $O\left(\sqrt{n\log(m/n)}\right)$ *on the mesh.*

Proof: Steps 1, 2.a, 3 of the access phase have, respectively, the same structure of Steps 1, 2.b, 3 of the first stage of copy selection, hence can be completed in $O\left(\sqrt{n\log(m/n)}\right)$ time using the same ideas. Both Steps 2.b and 2.d involve a k-relation routing (with $k = s + q$) hence require $O\left((s+q)\sqrt{n/s}\right) = O\left(\sqrt{n\log(m/n)}\right)$ time altogether. Finally Step 2.c can be completed in $O(q) = O(\log(m/n))$ time by individual nodes. The theorem follows by combining the complexities of the individual steps. □

By combining Theorem 7 with Theorem 8 we establish that any step of an (n, m)-PRAM can be simulated with slowdown $O\left(\sqrt{n\log(m/n)}\right)$ on an $\sqrt{n} \times \sqrt{n}$ mesh, using $O(\log(m/n))$ copies per variable. Section 6 shows how the read-only tables encoding the memory map held by each node may be replaced by a space-efficient, distributed representation using only $O\left((m/n)\log^5(m/n)\right)$ storage per node, thus completing the proof of Theorem 1 as it relates two-dimensional meshes.

4 Higher-Dimensional Meshes

The overall structure of the simulation scheme for the two-dimensional mesh described in Section 3 also applies to higher-dimensional meshes. Specifically, we adopt the same memory organization and pick the same values for the parameters s and q. In what follows, we sketch how to implement the individual steps of the copy selection and access phases efficiently on an $n^{1/d} \times n^{1/d} \times \cdots \times n^{1/d}$ d-dimensional mesh (called, for short, an *n-node d-mesh*) with constant d. As for the case of the two-dimensional mesh, we note that for $m \leq 2^{\Theta\left(n^{1/(d+1)}\right)}$, $k = O\left(\log(m/n)\right)$ and $n' = \Omega\left(n/\log(m/n)\right)$, the k-sorting problem and the k-relation routing problem may both be solved in time $O\left(k(n')^{1/d}\right)$ on an n'-node d-mesh [Kun93].

Let us first consider the first stage of copy selection, and interpret a cell C as an (n/s)-node d-submesh. Step 1 prescribes that the set $\hat{S}$ be replicated in each cell. We adopt the same recursive approach developed for $d = 2$ and perform the replication in substeps. At the i-th substep, each $(n/2^{d(i-1)})$-node d-submesh replicates its copy of $\hat{S}$ in each of its 2^d component $(n/2^{di})$-node d-submeshes . Such replication involves an instance of the 2^{di}-relation routing problem on an $n/2^{d(i-1)}$-node d-mesh and can therefore be completed in $O\left(2^{(d-1)i}n^{1/d}\right)$ time. Hence, the total time required by Step 1 is

$$O\left(\sum_{i=1}^{(1/d)\log s} 2^{(d-1)i}n^{1/d}\right) = O\left(n^{1/d}s^{1-1/d}\right).$$

Step 3 is accomplished in the same time by performing the routings of Step 1 in the reverse order. Let us finally consider Step 2. In this step, the nodes of each cell C first determine whether the cell degree is less than $q(n/s)$ (Step 2.a). If this is the case, a balanced collection of c-packets relative to the copies of variables in S residing in C is generated (Step 2.b) and finally checked for accessibility (Step 2.c). The operations involved are $\lceil \log(2c-1) \rceil$ prefix computations as well as a constant number of s-sorting, s-relation routing and q-sorting operations, plus an additional $O\left(q+s\right)$ work per node. By plugging in the chosen values for s and q we see that Step 2 of copy selection can be completed in time

$$O\left(sc + n^{1/d}s^{1-1/d} + q(n/s)^{1/d}\right) = O\left(n^{1/d}(\log(m/n))^{1-1/d}\right). \tag{3}$$

Since this subsumes the cost of Steps 1 and 3, this expression also captures the cost of the entire first stage of copy selection. In the second stage, we perform the i-th whittling entirely within an $n/(2c-1)^{i-2}$-node d submesh. As in the two-dimensional case, this whittling can be

easily implemented by means of 1-sorting and parallel prefix. Since the size of the submeshes is geometrically decreasing, the overall running time is dominated by the time for $i = 2$, which is $O\left(n^{1/d}\right)$. Hence, the entire copy selection algorithm can be completed within the time given by Equation 3.

Finally, recall that Steps 1, 2.a, 3 of the access phase mirror Steps 1, 2.b, 3 of copy selection, hence they require time $O\left(n^{1/d}(\log(m/n))^{1-1/d}\right)$ altogether. The remaining steps can be realized as two instances of an $(s+q)$-relation routing in each cell plus $O(q)$ work per node. Therefore, the total time required by the access phase is again $O\left(n^{1/d}(\log(m/n))^{1-1/d}\right)$ time.

The above discussion establishes that any step of an (n, m)-PRAM with $m \leq 2^{\Theta\left(n^{1/(d+1)}\right)}$ can be simulated with slowdown $O\left(n^{1/d}(\log(m/n))^{1-1/d}\right)$ on a d-dimensional mesh (with constant d) using $O(\log(m/n))$ copies per variable. In Section 6 we will show how the space requirements per node may be reduced to $((m/n)\log^3(m/n))$ storage per node. This will complete the proof of Theorem 1.

5 The Pruned Butterfly

An n-leaf *fat-tree* is a routing network whose coarse structure resembles that of an n-leaf binary tree. More specifically, leaves correspond to processing elements, non-leaf nodes represent clusters of routing switches and edges represent communication channels of bandwidth that increases from the leaves to the root. The first architecture of this kind was proposed by Leiserson [Lei85], and was later followed by a number of related networks differing in the detail of how components at different levels of the tree are interconnected. In this paper we adopt the *pruned butterfly* fat-tree of Bay and Bilardi [BB95], an example of which is illustrated in Figure 1. Each dotted ellipse in the figure identifies a cluster of routing switches that collectively correspond to a single node in the binary tree. The bundle of edges joining switches of a cluster to switches of its parent cluster constitutes a channel whose bandwidth equals the cardinality of the bundle. The *depth* of a cluster is the distance of its component switches from the root. At any given depth, the clusters are numbered from left to right, beginning at zero. Individual switches within each cluster are also numbered from left to right, beginning at zero.

Assume that n is a power of four. Formally, an n-leaf pruned butterfly is a graph $G =$

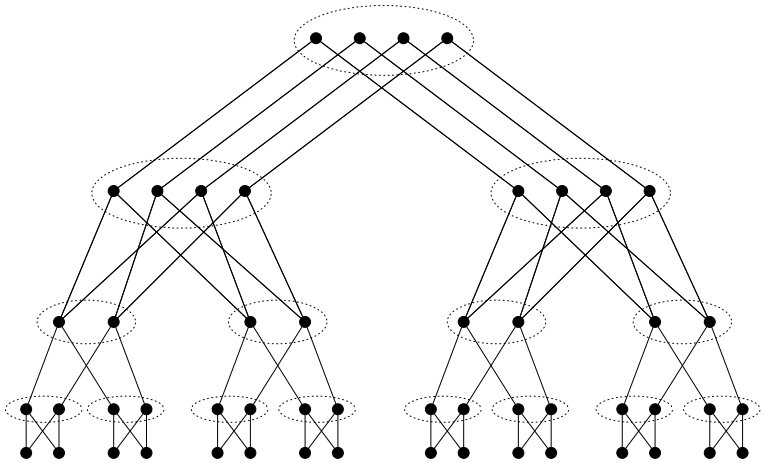

Figure 1: A pruned butterfly with 16 leaves

(V, E) whose vertices are indexed as follows :

$$V = \{< i, j, k >: 0 \leq i \leq \log n, 0 \leq j < 2^i, 0 \leq k < \sqrt{n}2^{-\lfloor i/2 \rfloor}\}.$$

With the above indexing, the j-th processor-memory node from left to right corresponds to vertex $< \log n, j, 0 >$. For $0 \leq i < \log n$, vertex $< i, j, k >$ corresponds to the k-th switch of the j-th cluster at depth i of the tree. The set of edges E is defined as follows:

$$E = \bigcup_{i=1}^{\log n} \bigcup_{j=0}^{2^i-1} E_{ij},$$

where E_{ij} contains the edges connecting switches in the j-th cluster at depth i to those in its parent cluster. When i is even, we let

$$\begin{aligned} E_{ij} \quad = \quad & \{(< i, j, k >, < i-1, \lfloor j/2 \rfloor, k >), \\ & (< i, j, k >, < i-1, \lfloor j/2 \rfloor, k + \sqrt{n}2^{-i/2} >) : \\ & 0 \leq k < \sqrt{n}2^{-i/2}\}. \end{aligned}$$

When i is odd, we let

$$E_{ij} = \{(< i, j, k >, < i-1, \lfloor j/2 \rfloor, k >: 0 \leq k < \sqrt{n}2^{-\lfloor i/2 \rfloor}\}.$$

Note that $|E_{ij}| = \sqrt{n}2^{1-\lceil i/2 \rceil}$, therefore channel bandwidths double every other level from the leaves to the top, ranging from 2 to $\sqrt{n}$.

G is interpreted as an n-node machine by identifying the n processor-memory nodes with the n leaves, a routing switch with each internal vertex, and a wire capable of transmitting a single packet along its length in unit time with each edge. Moreover we assume that each routing switch is provided with an adder, so that parallel prefix computations may be completed efficiently. Intuitively, to route a message from leaf i to leaf j, the message is routed upwards in the tree to the cluster that is the least common ancestor of i and j and thence downwards to its destination.

This architecture has a number of interesting properties. For example, it is a subgraph of the butterfly network and it embeds the n-leaf mesh of trees architecture with constant dilation and load. Furthermore, the original bit-serial formulation of the pruned butterfly presented by Bay and Bilardi is *area universal*: the n-leaf pruned butterfly can be laid out in $O\left(n \log^2 n\right)$ area and can route any set of messages almost as efficiently as any circuit of similar area. (See [BB95] and the references contained therein for a fuller discussion of area-universality and the capabilities and properties of the pruned butterfly.) An important routing property of the n-leaf pruned butterfly is the following. Consider a collection of $k \leq \sqrt{n}$ packets, stored one per node among the leaves, with the i-th packet residing at leaf $< \log n, s_i, 0 >$ and destined to leaf $< \log n, d_i, 0 >$. We refer to the collection as a *wave* if $s_1 < s_2 < \ldots < s_k$, and $d_1 < d_2 < \ldots < d_k$. In [BB95] it is shown that any wave can be easily routed in $O(\log n)$ time. Moreover, a sequence of t waves may be routed in a pipelined fashion in time $O(t + \log n)$.

5.1 Sorting and Routing on the Pruned Butterfly

In this subsection, we develop algorithms for k-sorting and k-relation routing on the pruned butterfly, which will needed for the shared memory simulation.

Lemma 4 *Any instance of k-sorting can be performed in $O(k(\log k + \sqrt{n}))$ time on the n-leaf pruned butterfly.*

Proof: We will consider the input packets sorted when the k packets with the smallest keys are in the 0-th leaf, the next k smallest occupy the 1-st leaf, and so on. Our sorting strategy is based on an adaptation of Batcher's bitonic sorting algorithm to handle the case where $k = 1$. The generalization to larger values of k is standard, and can be obtained by first sorting the sequence of input packets at each node (possibly padded with extra dummy packets with key=∞ to bring its length to k) in $O(k \log k)$ time, and then replacing each

constant-time compare-exchange operation in the algorithm for $k = 1$ with an $O(k)$-time merge-split operation [Knu73].

When $k = 1$, let $x_0, x_1, \ldots, x_{n-1}$ be the n-tuple of variables that we wish to sort, with x_i residing at the i-th leaf. The bitonic sorting algorithm is structured as a sequence of $\log n$ merging *phases*. During the i-th phase, for $1 \leq i \leq \log n$, distinct pairs of sorted subsequences of length 2^{i-1} are merged into subsequences of length 2^i. In turn, the i-th phase is made of (i, j)-*stages*, for $j = i-1, i-2, \ldots 1, 0$. An (i, j) stage executes as follows:

for all $0 \leq k \leq n-1$ **do in parallel**
  if $[k]_j = 0$ **then** $Oper(k, k + 2^{j-1}, i, j)$.

In the above code, $[k]_j$ denotes the j-th bit in the binary representation of k, while $Oper(k, k + 2^{j-1}, i, j)$ denotes a compare-exchange operation applied to the variables x_k and $x_{k+2^{j-1}}$. The "orientation" of the exchange depends on i and j. Note that the leaves containing these two variables fall within the same subtree of 2^j leaves. Thus, any (i, j)-stage can be performed within such subtrees. We now describe the implementation of an (i, j)-stage for the subtree with leaves $0, \cdots, 2^j - 1$, and note that the same algorithm can be executed simultaneously within each 2^j-leaf subtree.

1. Transfer the values of $x_0, x_1, \ldots, x_{2^{j-1}-1}$ (residing at distinct leaves of the left subtree) to the right subtree, so that leaf $k+2^{j-1}$ holds both x_k and $x_{k+2^{j-1}}$, for $0 \leq k \leq 2^{j-1}-1$.

2. Perform $Oper(k, k + 2^{j-1}, i, j)$ at leaf $k + 2^{j-1}$, for $0 \leq k \leq 2^{j-1} - 1$.

3. Transfer the updated values of $x_0, x_1, \cdots, x_{2^{j-1}-1}$ back into the leaves of the left subtree.

Clearly, Step 2 can be completed in $O(1)$ time, since *Oper* involves a simple comparison-exchange. Steps 1 and 3 have a similar structure and involve the routing of the 2^{j-1} values stored at the leaves of the left (resp., right) 2^{j-1}-leaf subtree to the leaves of its sibling subtree. We may decompose this routing into $\left\lceil 2^{j-1}/2^{\lfloor j/2 \rfloor} \right\rceil = O\left(\sqrt{2^j}\right)$ waves which may then be routed in a pipelined fashion in $O\left(\sqrt{2^j}\right)$ time. Hence, any (i, j)-stage can be completed in $O\left(\sqrt{2^j}\right)$ time.

Now, recall that the i-th merging phase of the sorting algorithm consists of a sequence of (i, j)-stages for $j = i-1, i-2 \ldots, 1, 0$. Hence, the total running time of the algorithm may be bounded as follows

$$\sum_{i=1}^{\log n} \sum_{j=0}^{i-1} O\left(\sqrt{2^j}\right) = \sum_{i=1}^{\log n} O\left(\sqrt{2^i}\right) = O\left(\sqrt{n}\right).$$

□

Recall that a k-relation routing problem involves routing a set of packets from source to destination subject to the constraint that no node is the source or destination of more than k packets. We have:

Lemma 5 *Any instance of the k-relation routing problem may be routed in $O\left(k(\log k + \sqrt{n})\right)$ time on the n-leaf pruned butterfly.*

Proof: Let $\hat{S}$ denote the set of packets to be routed. The routing algorithm is made of the following steps:

1. Sort the packets in $\hat{S}$ by their destination among the n leaves of the pruned butterfly.
2. For $0 \leq i < k\sqrt{n}$, route the packets whose rank in the sorted sequence is equal to $i \bmod k\sqrt{n}$.

By Lemma 4, Step 1 above requires time $O\left(k(\log k + \sqrt{n})\right)$. As for Step 2, it is easy to see that each of the $k\sqrt{n}$ routings is a wave. Therefore, all iterations can be pipelined and completed in time $O\left(k\sqrt{n}\right)$. Thus, the entire routing algorithm can be completed in $O\left(k(\log k + \sqrt{n})\right)$ time. □

5.2 The Simulation Algorithm

A closer look at the simulation algorithm devised for the two-dimensional mesh reveals that both the copy selection and access phases are implemented in terms of k-sorting, k-relation routing, prefix computations, and rely upon a recursive decomposition of the network into subnetworks of the same topology. Note that the pruned butterfly exhibits such a decomposition. Specifically, for n and $s \leq n$ arbitrary powers of two, an n-leaf pruned butterfly can be decomposed into s (n/s)-leaf pruned butterflies. Since the complexity of routing and sorting are asymptotically the same for the pruned butterfly and the mesh, we conclude that any step of an (n, m)-PRAM can be simulated with slowdown $O\left(\sqrt{n\log(m/n)}\right)$ on an n-leaf pruned butterfly using $O\left(\log(m/n)\right)$ copies per variable. The techniques of the next section show how space requirement per node may be limited to $O\left((m/n)(\log(m/n))^3\right)$ thus completing the proof of Theorem 2.

6 Space-Efficient Simulations

The simulations presented in the paper are based on a memory organization whose structure is modelled by a bipartite graph $G = (U, V; E)$, with $|U| = m$, $|V| = n$, and where every vertex in U has degree $2c - 1 = \Theta(\log(m/n))$. This graph may be represented by means of a read-only table $T_G = [t_1, t_2, \cdots, t_m]$ consisting of m *entries*, where the i-th entry $t_i = (t_i(1), t_i(2), \cdots, t_i(2c-1))$ contains the addresses of the copies of the i-th variable. We call each such address an *item*. In this section, we show that such a table may be represented in a distributed fashion among the nodes of the simulating network, so that the maximum number of items stored per node is $O\left((m/n)\log^3(m/n)\right)$ and that any N-tuple (with $N \geq n$) of entries (corresponding to the variables to be accessed in the PRAM step) may be read in time proportional to the slowdown of the simulation step. (The need to access an N-tuple rather than an n-tuple will be discussed later.) We sketch the required techniques for the two-dimensional mesh, which are akin to those presented in [Her96], though somewhat simpler. The result extends immediately to the other interconnections considered in the paper.

Let $n' \leq n$ be a parameter to be fixed later. Partition the $\sqrt{n} \times \sqrt{n}$ mesh into n/n' *tiles* of size $\sqrt{n'} \times \sqrt{n'}$. Each tile will contain a complete copy of T_G distributed as follows. Partition T_G into m/b *pages* of b entries each, and partition each tile into n'/b *blocks* each of size $\sqrt{b} \times \sqrt{b}$. Within each tile, replicate and distribute the m/b pages among the n'/b blocks that make up that tile according to a smooth $(\lambda, 2c'-1, c', 1/(2c'-1))$-generalized expander $H = (U_H, V_H; E_H)$ such that $|U_H| = m/b$, $|V_H| = n'/b$, $c' = 2c - 1$, and where parameters $\lambda < 1$, $c = \Theta(\log(m/n))$ are as defined in Section 2. The maximum number of pages mapped to any individual block is $O((m/n')c'))$, which amounts to $O((m/n')bc^2)$ items in all. The items mapped to a particular block are distributed evenly among the nodes of the block, with $O((m/n')c^2)$ items per node. Within each node, the individual items are held in a static dictionary in order to facilitate retrieval.

Note that there are a total of $(n/n')(2c'-1)$ copies of each entry and that to read an entry it suffices to read any one copy. Note also that the structure of the graph H can be represented by means of a read-only table T_H of m/b entries. This latter table is replicated and represented in every block in the network, with each node of each block holding m/b^2 entries of T_H.

To read an N-tuple of entries of T_G, each tile deals locally with the reads relating to its own nodes, independently of other tiles, by executing the following steps. (It is assumed that each node handles N/n entries.)

1. Generate a set S containing $2c' - 1$ numbered request packets $r_1(x), r_2(x), \ldots, r_{2c'-1}(x)$ for each referenced entry x. Packet $r_i(x)$ bears the name of the processor that generated it, the entry to which it refers, and the name of the block that contains the i-th copy of that entry within the tile in question.

2. Select a subset S' of the packets that contains c' packets $r'_1(x), \cdots, r'_{c'}(x)$ per referenced entry such that the number of selected packets relating to any individual block is $O((N/n)bc')$.

3. Route each packet in S' to the appropriate block, ensuring that the number of packets routed to any individual node is $O((N/n)c')$.

4. Within each block, circulate the packets around a Hamiltonian cycle. (For the pruned butterfly, use an Eulerian cycle.) As a packet, say $r'_i(x)$, visits a node, check whether that node contains a copy of entry t_x. If so, load a copy of item $t_x(i)$ into the packet.

5. Route each packet back to the node that generated it.

Notice that the $c' = 2c - 1$ selected packets relating to entry x are ultimately returned to the node that generated them, each bearing the value of a distinct item of that entry.

In order to discover the locations of the various copies of the entries, which are needed to generate packets during Step 1, the nodes need to query T_H. Since each block maintains a private copy of this table and each block generates $(N/n)b(2c' - 1)$ request packets, this operation can be accomplished in the same fashion as that outlined for Step 4 and has the same $O((N/n)bc')$ running time. Steps 3 and 5 involve $((N/n)c')$-relation routing within an n'-node tile so these contribute $O\left((N/n)c'\sqrt{n'}\right)$ to the running time.

As for Step 2, note that for each page of T_G the number of entries referenced may be up to b, the page size. For a particular tile, let P_i denote the set of pages where the number of referenced entries lies in the interval $[2^i, 2^{i+1})$. Clearly, $\sum_{i=0}^{\log b} 2^i|P_i| \leq 2(N/n)n'$. Since H is a generalized expander, it is possible to construct a c'-bundle for the pages in each P_i that has degree $O((|P_i|/(n'/b))c')$. Each edge in such a bundle corresponds to at most 2^{i+1} request packets, and so the total number of selected packets over all the P_i is at most $\sum_{i=0}^{\log b} 2^{i+1}(|P_i|/(n'/b))c' = O((N/n)bc')$. The algorithmic techniques required to perform the selection include straightforward combinations of sorting and parallel prefix akin to those employed during the second stage of the copy selection process of Section 3, and this step also has a running time of $O\left((N/n)c'\sqrt{n'}\right)$.

Thus, the overall running time is $O\left((N/n)(b+\sqrt{n'})c'\right)$. Recall that in our intended application, namely the reading of the addresses of variable copies during Step 2.b of the copy selection phase of the algorithm of Section 3 (and the corresponding step of the subsequent access phase), each mesh node generates $O(s) = \Theta(\log(m/n))$ such lookup requests. Thus, $N = sn$, so by choosing $b = \sqrt{n'} = \sqrt{n/\log^3(m/n)}$ and $c' = O(\log(m/n))$, this running time simplifies to $O\left(\sqrt{n\log(m/n)}\right)$. Moreover, the distributed representation of the memory map T_G requires $O((m/n')c'(2c-1)) = O\left((m/n)\log^5(m/n)\right)$ storage per node, while the representation of T_H contributes a further $O((m/b^2)c') = O\left((m/n)\log^4(m/n)\right)$ per node. Hence, the total storage requirement per node is $O\left((m/n)\log^5(m/n)\right)$.

7 Lower Bound

In this section, we prove a lower bound on the worst-case slowdown incurred when simulating a PRAM step on a processor network. Unlike previous approaches [AHMP87, KU88, HB94], which do not account for the network topology, we obtain a bound that is based on the bandwidth characteristics of the simulating network. As a result, while previous lower bounds were significant only for very powerful networks such as expanders, our lower bound can be specialized, yielding nontrivial results, to a broad family of topologies, including low-bandwidth ones such as d-dimensional meshes and the pruned butterfly. The bound is based on the notion of balanced decomposition tree [BL84], which provides a partition of the network into disjoint regions of known bandwidth. We first formulate the general lower bound in terms of such a decomposition, and then show how to specialize it to meshes and to the pruned butterfly.

Consider the simulation of an arbitrary (n, m)-PRAM program on an n-processor network $\mathcal{N}$. For convenience, we assume that each PRAM step involves either the reading (*read step*) or the writing (*write step*) of some n-tuple of variables. The simulation must satisfy the following standard constraints, which are also required in the lower bounds quoted earlier.

- The simulation must *on-line*, in the sense that each PRAM step is made known to the simulation algorithm only after the simulation of previous read steps has been completed. Thus, read steps are simulated one-by-one according to the order specified by the PRAM program. Each read must succeed in accessing the correct (*i.e.* most recently written) value of the variable in question. Note that no restriction is placed on the execution of write operations.

- The simulation must be *point-to-point* in the sense that a processor that wants to write a variable must dispatch a distinct message for each copy of the variable it wants to update.

Note that the point-to-point constraint rules out the splitting and combining techniques that are at the core of the simulations presented in this paper. However, at the end of the section, we modify the argument to obtain a non point-to-point lower bound, formulated in terms of the global space used to represent the PRAM memory, which applies to our upper bounds.

We assume that the simulating network $\mathcal{N}$ has a $[w_0, w_1, \ldots, w_{\log n}]$ *balanced decomposition tree*, as defined in [BL84]; that is, for any i, $0 \leq i \leq \log n$, $\mathcal{N}$ can be partitioned into 2^i disjoint *i-regions*, $R_1^{(i)}, \ldots, R_{2^i}^{(i)}$, where each i-region contains $\lceil n/2^i \rceil \pm 1$ processors and is connected to the rest of the network by at most w_i edges. Clearly, every network has a balanced decomposition tree, for suitable values of the w_i's.

Definition 3 *Let h and k be two integers, with $1 \leq h, k \leq \log n$, and let t be an arbitrary time step during the course of the simulation. For any shared variable $u \in U$, we define $r_{h,k}^t(u)$ to be the minimum, taken over all h-regions $R_j^{(h)}$, of the number of k-regions containing valid (i.e., most recently updated) copies of u that lie outside $R_j^{(h)}$ at the beginning of step t. (We assume $r_{h,k}^0(u) = 0$, for every h, k and u.) We also define the* average redundancy *at time t with respect to h and k as $r_{h,k}^t = \sum_{u \in U} r_{h,k}^t(u)/m$.*

The lower bound argument is similar in spirit to the ones in [AHMP87, KU88, HB94], namely, it relies on finding a sequence of PRAM steps which are "hard" to simulate. Such a sequence will contain a judicious mixture of write and read steps suitably chosen to expose a tradeoff of the following kind: unless the simulation devotes a sufficient amount of effort to each write step to ensure that the valid copies of the variables written are "nicely distributed" among different regions of the network, an adversary is always guaranteed to be able to find a read instruction that will be relatively expensive to simulate.

In the subsequent analysis, we will make use of the following technical lemma, whose proof is embedded in that of Lemma 1 in [PP97].

Lemma 6 ([PP97]) *Consider a fixed partition of the network into p disjoint regions, and a set of m' PRAM variables, such that, for each variable, there are at most $r' \geq 1$ distinct regions containing valid copies of the variable. Then, for any $n' \leq m'$, there exists a set of n'*

variables whose valid copies are all stored in memory modules residing in at most

$$\Phi(r', p, n', m') = 2 \cdot \max\left\{ r', p\left(\frac{n'}{m'}\right)^{\frac{1}{r'}} \right\}$$

regions.

A lower bound on the complexity of a read operation as a function of the redundancy of the simulation scheme is proved in the following lemma.

Lemma 7 *Fix an arbitrary time step t during the course of the simulation. For every h and k, with $1 \leq h, k \leq \log n$, at time t an adversary could issue a read step involving n distinct variables, whose simulation requires time at least $g_{h,k}(r^t_{h,k})$, where*

$$g_{h,k}(r) = \begin{cases} 1 & \text{if } \Phi(2r, 2^k, n, m/2^{h+1}) \geq 2^{k-2}, \\ \frac{n}{4\left(w_h + w_k \Phi(2r, 2^k, n, m/2^{h+1})\right)} & \text{otherwise} \end{cases}$$

and $r^t_{h,k}$ is the average redundancy at time t with respect to h and k .

Proof: Fix h and k and let $r = r^t_{h,k}$. The case $\Phi(2r, 2^k, n, m/2^{h+1}) \geq 2^{k-2}$ is trivial, hence we assume that $\Phi(2r, 2^k, n, m/2^{h+1}) < 2^{k-2}$. We will identify a set of $\Theta(n)$ variables all of whose valid copies are confined within a low-bandwidth portion of the network, and therefore are expensive to read by processors outside the region. Let $\hat{U} = \{u \in U : r^t_{h,k}(u) \leq 2r\}$. Clearly, $|\hat{U}| \geq m/2$ and there exists an h-region $R^{(h)}_{j_0}$ for which there are at least $m/2^{h+1}$ variables in $\hat{U}$ achieving their minimum redundancy with respect to $R^{(h)}_{j_0}$. Let $\hat{U}^{(h)}_{j_0} \subseteq \hat{U}$ be the set containing these variables. Note that since $\Phi(2r, 2^k, n, m/2^{h+1}) < 2^{k-2}$ we must have $m/2^{h+1} \geq n$ and, thus, $|\hat{U}^{(h)}_{j_0}| \geq n$. We distinguish between two cases, depending on whether r is less than $1/2$ or not.

If $r < 1/2$, then there exists an n-tuple of variables in $\hat{U}^{(h)}_{j_0}$ whose valid copies are all within $R^{(h)}_{j_0}$. Since $h \geq 1$ and so $R^{(h)}_{j_0}$ contains no more than $n/2$ processors, we can stipulate that $n/2$ variables of the n-tuple be read by processors outside $R^{(h)}_{j_0}$. At least one copy per variable must then be transmitted along the wires connecting the region with the rest of the network, therefore such a read instruction will require at least $n/(2w_h)$ time.

The case $r \geq 1/2$ is more involved. Fix an arbitrary subset W of $\hat{U}^{(h)}_{j_0}$ containing exactly $m/2^{h+1}$ variables. Each variable $u \in W$ may have a number of valid copies within $R^{(h)}_{j_0}$ plus at most $2r$ valid copies scattered among k-regions external to $R^{(h)}_{j_0}$ (call them *expensive copies*). By plugging $r' = 2r$, $p = 2^k$, $n' = n$, and $m' = m/2^{h+1}$ into the statement of Lemma 6,

we conclude that there are n variables in W whose expensive copies are all contained in at most $\Phi(2r, 2^k, n, m/2^{h+1})$ k-regions. Since $\Phi(2r, 2^k, n, m/2^{h+1}) < 2^{k-2}$, the union of $R_{j_0}^{(h)}$ and these k-regions contains no more than $3n/4$ processors. Therefore we can stipulate that $n/4$ variables of the n-tuple be read by processors outside the union. The lemma follows by observing that reading these variables would take time at least

$$\frac{n}{4\left(w_h + w_k \Phi(2r, 2^k, n, m/2^{h+1})\right)},$$

and that the above term is strictly less than $n/(2w_h)$.

□

We observe that the function $g_{h,k}(r)$ defined in the above lemma is a non-increasing function of r.

The following lemma is similar to Lemma 7 in [HB94], and captures the contribution of the write steps to the running time in terms of the average redundancy.

Lemma 8 *Consider an arbitrary time step t during the course of a point-to-point simulation, and let $r = r_{h,k}^t$. Then, t and r satisfy the following inequality:*

$$t \geq r \frac{m}{2^h w_h}.$$

Proof: For each variable u, let r_u denote the number of valid copies of u lying outside the h-region which contains the processor that most recently updated u (before time t). (Note that r_u as well as $r_{h,k}^t(u)$, for any h and k, are equal to 0 if no processor wrote u before time t.) Under the point-to-point assumption, such a processor must have dispatched at least r_u distinct messages that crossed the boundaries of its h-region. As a consequence, we have that a total of at least $\sum_{u \in U} r_u \geq rm$ messages must have crossed boundaries of h-regions, hence, there must be an h-region whose boundary was crossed by at least $rm/2^h$ distinct messages, which accounts for a total of at least $rm/(2^h w_h)$ time. □

Theorem 9 *For any $T \geq 2m/n$, there exists a T-step (n, m)-PRAM program, whose point-to-point, on-line simulation on an n-processor network with a $[w_0, w_1, \ldots, w_{\log n}]$ balanced decomposition tree, requires worst-case time*

$$\Omega\left(T \left\{ \max_{1 \leq h,k \leq \log n} \min_{r \geq 0} \left\{ g_{h,k}(r) + r \frac{n}{2^h w_h} \right\} \right\}\right),$$

where $g_{h,k}(r)$ is the function defined in Lemma 7.

Proof: We construct a PRAM program with $\lfloor T/(2m/n) \rfloor$ batches of m/n instructions, each batch consisting of m/n write steps that update all the variables, followed by m/n read steps suitably chosen by the adversary according to Lemma 7.

Consider the simulation of one such batch, for some h and k, with $1 \leq h, k \leq \log n$. Let r be the maximum value of $r_{h,k}^t$ at the start (time t) of the simulation of any read step. By Lemma 8, the simulation of all the write steps requires time at least $rm/(2^h w_h)$. By Lemma 7, the simulation of each read step requires time at least $g_{h,k}(r)$. Hence, the simulation time for the batch is at least

$$\frac{m}{n}\left(g_{h,k}(r) + \frac{rn}{2^h w_h}\right).$$

The theorem follows by taking the minimum over all possible values of r and the maximum over all choices of h and k of the simulation time of a batch given above, and then by summing the contributions of the $\lfloor T/(2m/n) \rfloor$ batches. □

We are now ready to prove Theorem 3, stated in the Introduction, which specializes the general lower bound of Theorem 9 to the case of d-dimensional meshes (with d constant).

Proof of Theorem 3: Let us first concentrate on one-dimensional meshes ($d = 1$). We establish this case separately, by means of a simple, diameter-based argument as follows. Consider a PRAM program consisting of T steps where in odd steps a processor v updates a variable u and in even steps all other processors read u. Such a sequence requires $\Omega(Tn)$ time to be simulated on the linear array since distinct pairs of consecutive write and read steps must be simulated in disjoint time intervals, because of the on-line hypothesis, and in each such pair the newly written value of u must travel at distance $\Theta(n)$.

Consider now the case $d > 1$. A natural halving process of an n-node d-dimensional mesh generates a balanced decomposition tree with $w_i = \Theta\left((n/2^i)^{1-1/d}\right)$, for $0 \leq i \leq \log n$. Define

$$\Delta = \frac{\log(m/n)}{\log\log(m/n)},$$

and fix h and k as the minimum indices such that $2^h \geq \Delta$ and $2^k \geq \Delta^{(2d-1)/(d-1)}$. Since $m \geq 16n$ and $d > 1$, we have $\Delta \geq 2$, hence $h, k \geq 1$. Let $\bar{m}n$ be the largest value of m for which the chosen value for 2^k is at most n (note that this also implies $2^h \leq n$). We first prove the lower bound under the assumption $m \leq \bar{m}n$. Let us define $\bar{r} = \Delta/16$ and note that

$$r\frac{n}{2^h w_h} = \Omega\left(n^{\frac{1}{d}}\Delta^{1-\frac{1}{d}}\right) \qquad \text{for } r \geq \bar{r}. \tag{4}$$

Using the chosen values of h, $\bar{r}$ and Δ, we see that

$$\left(\frac{m}{2^{h+1}n}\right)^{1/2\bar{r}} \geq \frac{(m/n)^{8\log(m/n)/\log\log(m/n)}}{(2^{h+1})^{8/\Delta}} \geq \frac{\log^8(m/n)}{8^4},$$

where the simplification of the denominator relies on the facts that $2^{h+1} < 4\Delta$ and $(4\Delta)^{8/\Delta} \leq 8^4$ (since $4\Delta \geq 8$ and $x^{1/x}$ is decreasing for $x > 2$). With this it is easy to establish that when $r < \bar{r}$, we have $\Phi(2r, 2^k, n, m/2^{h+1}) < 2^{k-2}$, which according to the definition of $g_{h,r}(r)$ in Lemma 7 implies that

$$g_{h,k}(r) = \frac{n}{4\left(w_h + w_k\Phi(2r, 2^k, n, m/2^{h+1})\right)}$$

Substituting for w_h, w_k and Φ, this simplifies to

$$g_{h,k}(r) = \Theta\left(n^{\frac{1}{d}} \min\left\{(2^h)^{1-\frac{1}{d}}, \frac{1}{r}(2^k)^{1-\frac{1}{d}}, \left(\frac{1}{2^k}\right)^{\frac{1}{d}}\left(\frac{m}{n2^{h+1}}\right)^{\frac{1}{2r}}\right\}\right).$$

Using the chosen values for 2^h, 2^k and the facts that $g_{h,k}(r)$ is non-increasing in r and $(m/(2^{h+1}n))^{1/(2\bar{r})} = \Omega\left(\log^8(m/n)\right) = \Omega\left(\Delta^8\right)$, we have

$$g_{h,k}(r) \geq g_{h,k}(\bar{r}) = \Omega\left(n^{\frac{1}{d}}\Delta^{1-\frac{1}{d}}\right) \qquad \text{for } r < \bar{r}. \tag{5}$$

By combining Equations 4 and 5 the lower bound on the simulation time in the case $m \leq \bar{m}$ is

$$\Omega\left(Tn^{\frac{1}{d}}\left(\frac{\log(m/n)}{\log\log(m/n)}\right)^{1-\frac{1}{d}}\right).$$

Straightforward but tedious calculations show that our choice of h and k yields the best possible bound. Moreover, the lower bound is an increasing function of m, hence, for $m \geq \bar{m}$, the simulation time is at least

$$\Omega\left(Tn^{\frac{1}{d}}\left(\frac{\log(\bar{m}/n)}{\log\log(\bar{m}/n)}\right)^{1-\frac{1}{d}}\right) = \Omega\left(Tn^{\frac{1}{d}}n^{\frac{d^2-2d+1}{2d^2-d}}\right),$$

since

$$\frac{\log(\bar{m}/n)}{\log\log(\bar{m}/n)} = \Theta\left(n^{\frac{d-1}{2d-1}}\right),$$

and the theorem follows □

Note that the argument used to prove Theorem 9 does not exploit the fine-grained struc-

ture of the interconnection but solely depends on the bandwidth distribution, as captured by the decomposition tree. Consequently, we get the same specialized version of the lower bound for networks of different topologies which have similar decomposition trees. An example is provided by the n-leaf pruned butterfly that has the same decomposition tree (up to constant factors) as the two-dimensional mesh, although the two topologies are very different. Hence, the proof of Theorem 4, stated in the Introduction, is virtually identical to that of Theorem 3 for $d = 2$, and is omitted for brevity.

Recall that the simulations presented in this paper achieve high levels of efficiency by making a crucial use of splitting and combining techniques. More specifically, a processor issuing a memory request generates a single variable packet for each subset of copies residing in a suitably sized region of the network. Once the variable packet is shipped within its destination region, it is split into multiple copy packets, destined to the individual copies of the variable. In this way, the cost of the "long leg" of the journey to access a copy is paid only once for all the copies residing within the same region.

Unfortunately, the point-to-point assumption made by our lower bound argument precludes the splitting and combining of messages destined to distinct copies, therefore Theorems 3 and 4 do not apply to our simulations. Note however that the argument uses this assumption only to establish a bound on the cost of write operations. As a consequence, we can prove a lower bound solely based on the cost of read operations, which holds in an unrestricted model where splitting/combining may occur. The lower bound, stated in Theorem 5 in the Introduction and proved below, is obtained by making sure that the average redundancy does not grow too large during the simulation. This can be achieved by establishing that the total amount of space used to represent the PRAM variables in the local memory modules can never exceed a fixed threshold mr.

Proof of Theorem 5: Consider the case of d-dimensional meshes. For $d = 1$, the bound can be trivially obtained through the same diameter-based argument employed in the proof of Theorem 3. Hence, assume $d > 1$. We consider a PRAM program that first executes m/n write steps to update all the variables, and then executes $T - (m/n) = \Theta(T)$ read steps suitably chosen by the adversary. Since the average number of copies per variable is r, it is immediate to argue that, for every $1 \leq k \leq \log n$, at the beginning of each read step there are at least $m/2$ variables each of which has updated copies in at most $2r$ k-regions. By Lemma 6 this implies that there exist $\min\{2^k, \Phi(2r, 2^k, n, m/2)\}$ k-regions that contain all updated copies of at least n variables. If $\Phi(2r, 2^k, n, m/2) \leq 2^{k-1}$, the adversary can require

that $n/2$ such variables be read by processors outside the $\Phi(2r, 2^k, n, m/2)$ k-regions, which takes time

$$\Omega\left(\frac{n}{w_k \Phi(2r, 2^k, n, m/2)}\right) = \Omega\left(n^{\frac{1}{d}} \min\left\{\frac{1}{r} 2^{k\left(1-\frac{1}{d}\right)}, \left(\frac{1}{2^k}\right)^{\frac{1}{d}} \left(\frac{m}{2n}\right)^{\frac{1}{2r}}\right\}\right). \tag{6}$$

Let us fix k as the minimum index such that

$$2^k \geq \left(\frac{\log(m/n)}{\log\log(m/n)}\right)^{\alpha+\frac{d}{d-1}},$$

and let $\bar{m}$ be the largest value of m for which the chosen value for 2^k is at most n. As in the proof of Theorem 3, we first consider the case $m \leq \bar{m}$. In this case, we have $1 \leq k \leq \log n$. Moreover, it is easy to verify that $(m/2n)^{1/(2r)} \geq (\log(m/n)/\log\log(m/n))^{2\alpha+1}$ and $\Phi(2r, 2^k, n, m/2) \leq 2^{k-1}$. By plugging the value for 2^k in the right-hand side of Equation 6 we see that the cost of each read operation is

$$\Omega\left(n^{\frac{1}{d}} \left(\frac{\log(m/n)}{\log\log(m/n)}\right)^{\alpha\left(1-\frac{1}{d}\right)}\right).$$

When $m \geq \bar{m}$, we have

$$r \leq \frac{1}{8\alpha} n^{\frac{1}{\alpha+d/(d-1)}} \leq \frac{1}{8\alpha} \frac{\log(\bar{m}/n)}{\log\log(\bar{m}/n)},$$

and the cost of each read can be easily bounded from below by

$$\Omega\left(n^{\frac{1}{d}} \left(\frac{\log(\bar{m}/n)}{\log\log(\bar{m}/n)}\right)^{\alpha\left(1-\frac{1}{d}\right)}\right) = \Omega\left(n^{\frac{1}{d}} n^{\frac{\alpha}{\alpha+d/(d-1)}\left(1-\frac{1}{d}\right)}\right).$$

The unrestricted lower bound for the pruned butterfly is obtained by setting $d = 2$ in the above calculations. □

Theorem 5 shows that our simulations use an amount of redundancy which is only a doubly logarithmic factor higher than the minimum redundancy needed to achieve the same slowdown.

8 Conclusions

In this paper we have presented upper and lower bounds for the problem of simulating a shared memory abstraction on network-based machines such as d-dimensional meshes and the

pruned butterfly. An interesting feature of our scheme is its generality. Indeed, the simulation algorithm relies on a recursive decomposition of the underlying network into subnetworks of the same topology, and employs a restricted set of basic primitives such as prefix, k-sorting and k-relation routing. As a consequence, the algorithm is efficiently portable to any other machine with a recursive structure and on which optimal algorithms for the above primitives are known. As for the lower bound, we have developed a generic, bandwidth-based argument that can be applied to any specific interconnection using the parameters of its decomposition tree.

Regarding the upper bound, it must be remarked that we make use of memory organizations based on generalized expanders. As it was mentioned in the Introduction, the explicit, deterministic construction of generalized expanders is a long-standing open question, although it can be shown that a random bipartite graph would exhibit the required expansion property with high probability. This limitation suffered by our scheme is shared by all other deterministic mesh-based schemes in the literature, with the exception of the scheme of [PPS94], which only applies to very small memory sizes ($m = O(n^{1.5})$) and exhibits a higher slowdown than ours.

Finally, the general lower bound presented in Section 7 is proved under the point-to-point assumption, which stipulates that packets sent to copies of a variable can neither be split nor combined. This constraint rules out the techniques that are at the core of the simulations presented in this paper, hence the bound does not apply to our algorithms directly. However, we have been able to modify the argument to obtain one which applies to our algorithms, by introducing an upper limit to the global space used to represent the PRAM variables. In particular, we are able to show that in order to match the slowdowns exhibited by our simulations, any deterministic scheme must use about the same amount of space to represent the variables. However, the search for a nontrivial, totally unrestricted lower bound for deterministic PRAM simulation on network-based machines remains a challenging open problem.

Acknowledgements

The authors would like to thank Gianfranco Bilardi for a some helpful discussions on sorting on the pruned butterfly, and the anonymous referee who indentified some shortcomings in the original version of the manuscript.

References

[AHMP87] Alt, H., Hagerup, T., Mehlhorn, K., and Preparata, F.P. (1987), Deterministic simulation of idealized parallel computers on more realistic ones, *SIAM J. Comput.*, **16** (5), 808–835.

[BB95] Bay, P., and Bilardi, G. (1995), Deterministic on-line routing on area-universal networks, *J. ACM*, **42** (3), 614–640.

[BL84] Bhatt, S.N. and Leighton, F.T. (1984), A framework for solving VLSI graph layout problems, *J. Comput. System Sci.*, **28** (2), 300–342.

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

[HB94] Herley, K.T., and Bilardi, G. (1994), Deterministic simulations of PRAMs on bounded-degree networks, *SIAM J. Comput.*, **23** (2), 276–292.

[KU88] Karlin, A.R., and Upfal, E. (1988), Parallel hashing: An efficient implementation of shared memory, *J. ACM*, **35** (4), 876–892.

[Knu73] Knuth, D.E. (1973), "The Art of Computer Programming, volume 3: Sorting and Searching," Addison Wesley, Reading, Mass.

[Kun93] Kunde, M. (1993), Block gossiping on grids and tori: Deterministic sorting and routing match the bisection bound, *in* "Proceedings, 1st European Symposium on Algorithms" (T. Lengauer, Ed.), pp. 272–283, Springer-Verlag LNCS 726, Berlin, Germany.

[Lei85] Leiserson, C.E. (1985), Fat-trees: Universal networks for hardware-efficient supercomputing, *IEEE Trans. Comput.*, **C-34** (10), 892–901.

[PP95] Pietracaprina, A., and Pucci, G. (1995), Improved deterministic PRAM simulation on the mesh, *in* "Proceedings, 22nd International Colloquium on Automata, Languages and Programming" (Z. Fülöp and F. Gécseg, Eds.), pp. 372–383, Springer-Verlag LNCS 944, Berlin, Germany.

[PP97] Pietracaprina, A., and Pucci, G. (1997), The complexity of deterministic PRAM simulation on distributed memory machines, *Theory Comput. Syst.*, **30** (3), 231–247.

[PPS94] Pietracaprina, A., Pucci, G., and Sibeyn, J.F. (1994), Constructive deterministic PRAM simulation on a mesh-connected computer, *in* "Proceedings, 6th ACM Symposium on Parallel Algorithms and Architectures," pp. 248–256. (Journal version to appear in *SIAM J. Comput.*)

[UW87] Upfal, E., and Widgerson, A. (1987), How to share memory in a distributed system, *J. ACM*, **34** (1), 116–127.