Implementing Shared Memory on Clustered Machines*
(Extended Abstract)

Carlo Fantozzi  Andrea Pietracaprina  Geppino Pucci
carlo.fantozzi@dei.unipd.it  andrea@artemide.dei.unipd.it  geppo@dei.unipd.it

Università di Padova, Dipartimento di Elettronica e Informatica
Via Gradenigo 6/A, I-35131, Padova, Italy

Abstract

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

1. Introduction

Providing a shared address space on a distributed-memory parallel system is a fundamental problem which has received vast attention over the last two decades. This problem, often referred to as PRAM simulation, requires the development of a scheme to represent \(m\) shared objects (called variables) among \(n\) interconnected processor/memory pairs in such a way that any \(n\)-tuple of variables can be read/written efficiently by the processors. The time required by a parallel access to an arbitrary \(n\)-tuple of variables is referred to as the slowdown of the scheme.

Both randomized and deterministic schemes are known in the literature for a number of specific processor networks. Randomized schemes (see, e.g., [3, 14]) usually distribute the variables among the memory modules local to the processors through one (or more) hash functions, suitably chosen from a universal class. The properties of these functions guarantee, with high probability, that the variables are uniformly distributed among the modules so that a simple routing strategy is sufficient to access any \(n\)-tuple of variables efficiently.

On the other hand, deterministic schemes require a redundant representation of the address space in order to avoid trivial worst cases where a few memory modules contain all of the requested data. More specifically, every variable is replicated into \(\rho\) copies, which are distributed among the memory modules through a map exhibiting suitable expansion properties, needed to guarantee that the copies relative to any \(n\)-tuple of variables be never confined to a low-bandwidth region of the network. The parameter \(\rho\) is referred to as the redundancy of the scheme. The main idea, originally introduced in [15] and adopted in all subsequent works, is that any access (read or write) to a variable is satisfied by reaching only a subset of its copies, suitably chosen to reduce network congestion while ensuring consistency (i.e., a read access must always return the most updated value of the variable).

Based on the above idea, deterministic schemes have been developed in the literature for a number of interconnections (see e.g., [1, 6, 7, 8]) which achieve slowdowns that are at most a logarithmic factor away from the natural lower bound imposed by the network’s bandwidth and latency characteristics. All of these schemes require redundancy logarithmic in \(m\) and rely on the existence of very powerful expanding graphs of nonconstant degree for which no explicit construction is known. Moreover, each individual scheme is strictly tailored to the specific structure of the underlying interconnection and does not seem to be easily portable to others.

In a recent work [13] a deterministic shared-memory implementation scheme has been devised for an \(n\)-processor mesh, which achieves \(O(\sqrt{n\log n})\) slowdown and features

---

*This research was supported in part by CNR and MURST of Italy.
a novel memory organization based on weakly expanding graphs that can be explicitly constructed for a range of memory sizes. Moreover, the memory map exhibits a hierarchical structure tailored to the natural recursive decomposition of the mesh into submeshes and shows promise to be portable to other clustered architectures. Building on the techniques of [13], in this paper we develop a general scheme to implement shared memory on any parallel machine with a clustered structure, achieving a worst-case slowdown which is optimal, or close to optimal, with respect to the limitations imposed by the machine’s bandwidth/latency characteristics. To achieve generality, the scheme is designed for the D-BSP, a generic machine model whose parameters can be instantiated to closely reflect the characteristics of any clustered machine.

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

1.1. Machine model

The Decomposable Bulk Synchronous Parallel (D-BSP) model was introduced in [4] as an extension of Valiant’s BSP [16] aimed at capturing locality within clusters; in this paper, we employ the following more regular version of the model (referred to as recursive D-BSP in [4]). Let \( n \) be a power of two, and let \( g = (g_0, g_1, \ldots, g_{\log n}) \) and \( \ell = (\ell_0, \ell_1, \ldots, \ell_{\log n}) \). A D-BSP \((n, g, \ell)\) is a collection of \( n \) processor/memory pairs communicating through a router. For \( 0 \leq i \leq \log n \), the \( n \) processors are partitioned into \( 2^i \) fixed, disjoint \( i \)-clusters \( C_0^{(i)}, C_1^{(i)}, \ldots, C_{2^i-1}^{(i)} \) of \( n/2^i \) processors each, where the processors of a cluster are able to communicate among themselves independently of the other clusters. The clusters form a hierarchical, binary decomposition tree of the D-BSP machine: specifically, \( C_j^{(i)} \) contains only processor \( P_j \), for \( 0 \leq j < n \) and \( C_j^{(i)} = C_{2j}^{(i+1)} \cup C_{2j+1}^{(i+1)} \), for \( 0 \leq i < \log n \) and \( 0 \leq j < 2^i \).

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

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

\[
\begin{align*}
g_i^{(\alpha)} &= G \cdot (n/2)^{\alpha}, \\
\ell_i^{(\beta)} &= L \cdot (n/2)^{\beta}, \\
0 &\leq i \leq \log n,
\end{align*}
\]

where \( G \) and \( L \) are two arbitrary positive constants. Note that these parameters capture a wide family of machines whose clusters feature moderate bandwidth/latency properties. Indeed, by combining the routing results of [12, 11] with those in [2] it can be shown that any recursive network topology with bisection bandwidth \( O(n^{1-\alpha}) \) and diameter \( O(n^\beta) \) can be modeled effectively by a D-BSP \((n, g^{(\alpha)}, \ell^{(\beta)})\). For instance, the values \( \alpha = \beta = 1/d \), for any integer \( d \geq 1 \), yield a D-BSP machine whose bandwidth/latency distribution is that of a \( d \)-dimensional array.

1.2. New results

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

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

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

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

The following corollary is an immediate consequence of the theorem.

**Corollary 1** For any value \( m \) upper bounded by a polynomial in \( n \) there exists a scheme to implement a shared memory of size \( m \) on a D-BSP \((n, g^{(\alpha)}, \ell^{(\beta)})\) with optimal slowdown \( \Theta(n^\alpha) \) and constant redundancy, when \( \alpha < \beta \), and slowdown \( \Theta(n^\alpha \log n) \) and redundancy \( O(\log^{1.5} n) \), when \( \alpha \geq \beta \). The scheme requires only weakly expanding graphs of constant degree and can be made fully constructive for \( m = O(n^{3/2}) \) and \( \alpha \geq 1/2 \).

The importance of our results is twofold. First, the proposed scheme is general and can be implemented on any machine supporting the D-BSP abstraction. Second, Corollary 1 shows, for the first time in the literature, that optimal worst-case slowdowns for shared memory access are achievable with constant redundancy for machines where latency overheads dominate over those due to bandwidth.
limitations, as is often the case in network-based parallel machines. Moreover, a simple adaptation of the lower bound argument given in [9] suffices to show that the minimal slowdown achievable for \( \alpha \geq \beta \) by any scheme which dispatches an individual message for each copy to be accessed is \( \Omega \left( n^\beta (\log(m/n)/(\log \log(m/n))^{1-\epsilon} \right) \), and that in order to achieve such slowdown redundancy \( \Omega \left( \log(m/n)/(\log \log(m/n)) \right) \) is necessary.

As mentioned before, our scheme builds upon the one presented in [13], whose design exploits the recursive decomposition of the underlying interconnection to provide a hierarchical, redundant representation of the shared memory based on \( k+1 \) levels of logical modules. More specifically, each variable is replicated into \( r = O(1) \) copies, and the copies are assigned to \( r \) logical modules of level \( 0 \). In turn, the logical modules at the \( i \)-th level, \( 0 \leq i < k \) are replicated into three copies, which are assigned to three modules of level \( i+1 \). This process eventually creates \( r 3^k = \Theta \left( 3^k \right) \) copies of each variable, and \( 3^{k-i} \) replicas of each module at level \( i \). The number (resp., size) of the logical modules decreases (resp., increases) with the level number, and their replicas are assigned for storing to distinct subnetworks of appropriate size.

The key ingredients of the above memory organization are represented by the bipartite graph that governs the distribution of the copies of the variables among the modules of the first level, and those that govern the distribution of the replicas of the modules at the subsequent levels. The former graph is required to exhibit some weak expansion property and its existence can always be proved through combinatorial arguments although, for certain memory sizes, explicit constructions can be given. In contrast, all the other graphs employed in the scheme require expansion properties that can be obtained by suitable modifications of the BIBD graph [5], and can always be explicitly constructed. In fact, the choice of these latter graphs is where the memory organization adopted here mainly differs from the one presented in [13]. In particular, unlike [13], these graphs are not simply subgraphs of the BIBD, but their construction requires some nontrivial rearrangements to make them suitable for the clustered structure of the D-BSP while maintaining good expansion properties.

For an \( n \)-tuple of variables to be read/written, the selection of the copies to be accessed and the subsequent execution of the accesses of the selected copies are performed through suitable protocols, similar to the ones in [13], which can be implemented through a combination of prefix, sorting and routing primitives. We employ efficient D-BSP implementations for these primitives, including a novel optimal implementation of \((k_1,k_2)\)-routing, to obtain the results stated above.

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

2. D-BSP primitives

The scheme described in the following sections makes use of prefix, sorting and routing primitives. The following three propositions provide efficient algorithms for these primitives on a D-BSP.

**Proposition 1** ([4]) Any parallel prefix of \( n \) items can be computed on a D-BSP \((n,g^{(\alpha)},\ell^{(\beta)})\) in optimal time \( T_{\text{pref}}(n) = \Theta \left( n^\alpha + n^\beta \right) \).

We call \( k \)-sorting a sorting problem in which \( k \) keys are initially assigned to each one of the \( n \) D-BSP processors and are to be redistributed so that the \( k \) smallest keys will be held by processor \( P_i \), the next \( k \) smallest ones by processor \( P_{i+1} \), and so on. We have:

**Proposition 2** Any instance of \( k \)-sorting, with \( k = \text{poly}(n) \), can be executed in optimal time \( T_{\text{sort}}(k,n) = \Theta \left( n^\alpha + n^\beta \right) \) on a D-BSP \((n,g^{(\alpha)},\ell^{(\beta)})\).

**Proof.** Simulate bitonic sorting on a hypercube [10] using the merge-split rather than the compare-swap operator, and mapping processors adjacent along the \( i \)-th dimension of the cube to D-BSP processors within the same \((\log n - i - 1)\)-cluster, for \( 0 \leq i < \log n \).

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

**Proposition 3** Any instance of \((k_1,k_2)\)-routing can be executed on a D-BSP \((n,g^{(\alpha)},\ell^{(\beta)})\) in optimal time

\[
T_{\text{route}}(k_1,k_2,n) = \Theta \left( k_{\min}^{\alpha}k_{\max}^{\ell}n^\alpha + n^\beta \right),
\]

where \( k_{\min} = \min\{k_1,k_2\} \) and \( k_{\max} = \max\{k_1,k_2\} \).

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

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

Note that the above algorithm does not require that the values of \( k_1 \) and \( k_2 \) be known \textit{a priori}. It is easy to see that at the end of iteration \( i \) of the first cycle each processor holds at most \( a_i = \min\{k_1, k_22^i\} \) messages, while at the end of iteration \( i \) of the second cycle, each message is in its destination \((i + 1)\)-cluster, and each processor holds at most \( d_i = \min\{k_2, k_12^i\} \) messages. Note also that iteration \( i \) of the first (resp., second) cycle can be implemented through a constant number of prefix operations and one routing of an \( a_i \)-relation (resp., \( d_i \)-relation) within \( i\)-clusters.

Putting it all together, the running time of the above algorithm on a D-BSP \((n, g^{(\alpha)}, \ell^{(\beta)})\) is

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

The theorem follows by plugging in the above formula the bounds for \( a_i \) and \( d_i \) derived before. \( \square \)

3. Memory organization

The Hierarchical Memory Organization Scheme (HMOS) that governs the distribution of the copies of the \( m \) shared variables among the \( n \) processors’ local memories is a cascade of bipartite graphs obtained as a modification of the one introduced in [13]. In what follows we briefly recall how the HMOS is structured and point out the differences with the version in [13].

Let \( V \) denote the set of variables, and let \( U_i \), \( 0 \leq i \leq k \), denote a set of nodes referred to as \( i\)-modules, where \( k = O(\log \log n) \) is a suitable nonnegative integer that will be specified in the analysis. The \( U_i \)'s can be regarded as nested collections of variables, and are obtained as follows. First, each variable is replicated into \( r = O(1) \) copies, \( r \) odd, which are assigned to distinct 0-modules. The contents of each 0-module, viewed as an indivisible unit, are in turn replicated into 3 copies, which are assigned to distinct 1-modules. The contents of each 0-module, viewed as an indivisible unit, are in turn replicated into 3 copies, which are assigned to distinct 1-modules. In general, the contents of each \((i-1)\)-module, viewed as an indivisible unit, are replicated into 3 copies, which are assigned to distinct \( i\)-modules, for \( 0 < i \leq k \). It is easy to see that the above process will eventually create \( 3^{k-1} \) replicas of each \( i\)-module and \( r3^k \) copies per variable. We will reserve the term \textit{copy} to denote the replica of a variable, and \textit{i-block} to denote the replica of an \( i\)-module. (Note that with this terminology \( k\)-modules and \( k\)-blocks coincide.) A \( k\)-module is composed of \((k-1)\)-blocks, which in turn are composed of \((k-2)\)-blocks, and so on. Finally, \( 0\)-blocks contain copies of variables.

The mapping between variables and \( 0\)-modules is represented by a bipartite graph \((V, U_0)\), where each variable \( v \in V \) is adjacent to the \( r \) 0-modules whose \( 0\)-blocks hold the copies of \( v \). Similarly, the mapping between \((i-1)\)-modules and \( i\)-modules is represented by a bipartite graph \((U_{i-1}, U_i)\), \( 1 \leq i \leq k \), where each \((i-1)\)-module \( u \) is adjacent to the \( 3 \) \( i\)-modules whose \( i\)-blocks contain the \((i-1)\)-blocks of \( u \). We assume that \( m = n^\tau \), for some constant \( \tau > 1 \). Let us fix \( m_0 = |U_0| = n \) and \( m_i = |U_i| = \Theta \left( n^{1/2^i} \right) \), for \( 1 \leq i \leq k \). We assume that the quantity \( d_i = \lfloor 3m_{i-1}/m_i \rfloor \) is a power of 2, and that every \( u \in U_i \) has degree at most \( d_i \) in \((U_{i-1}, U_i)\), for \( 1 \leq i \leq k \). Later, we will show how the various graphs can be chosen to satisfy all of the above constraints on the parameters, while exhibiting suitable expansion properties, which are needed to ensure low parallel access time.

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

\[
t_i = \Theta \left( 3^{i-k} n^{1-1/2^i} \right), \quad 1 \leq i \leq k.
\]

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

As mentioned before, suitable expansion properties are required of the graphs underlying the HMOS structure in order to ensure that, for any set of variables to be accessed, their copies be well spread among the processors’ memories, thus yielding low access time. As in [13], we choose \((V, U_0)\) to have input degree \( r \) and output degree \( rm/n \), and to exhibit the following property (referred to as \((n/m, \epsilon, \mu)\)-expansion): for any subset \( S \subseteq V \), \( |S| \leq n \), and for any set \( E \) of \( \mu |S| \) edges, \( \mu \) outgoing edges for each node in \( S \), the set \( \Gamma^+(S) \subseteq U_0 \) reached by the chosen edges has size at least \( \Omega \left( |S|^{1-\epsilon} \right) \), where \( \epsilon < 1 \) is a positive constant and
There is exactly one node graph has the following property: for every subset \( \nu \) and any selection of \( \nu \) \( \nu \subseteq \{z \} \times \{z \} \), the nodes of \( B \) graph \( G = (X, Y) \) with \( |X| = w \) can be explicitly constructed such that

1. \( \sqrt{6w} < |Y| < 6 + 6\sqrt{6w} \);
2. Every node of \( X \) has degree 3;
3. Every node of \( Y \) has degree at most \( \lfloor 3w/|Y| \rfloor \), and \( d \) is a power of 2;
4. For every subset \( S \subseteq X \) whose nodes are all adjacent to some \( y \in Y \) and for every selection of \( \nu \) \( \nu \subseteq \{z \} \times \{z \} \) edges, \( \nu \) incident on every \( s \in S \), the nodes of \( Y \) reached by the selected edges are at least \( \lfloor \nu/4 \rfloor + 1 \).

Proof (sketch). Our construction is based on a \( (z, 3) \)-BIBD (Balanced Incomplete Block Design) \( [5] \), that is, a bipartite graph \( G_{\text{BIBD}} = (A, B) \) with \( z = |B| \) a power of 3, \( |A| = z(z - 1)/6 \), and such that for any two nodes \( b_1, b_2 \in B \) there is exactly one node \( a \in A \) adjacent to both. Such a graph has the following property: for every subset \( S \subseteq A \) and any selection of \( \nu \) \( \nu \subseteq \{z \} \times \{z \} \) edges, \( \nu \) incident on every \( s \in S \), the nodes of \( B \) reached by the selected edges are at least \( (\nu - 1)/4 \) + 1. We fix \( z \) to be the smallest power of 3 such that \( |A| \geq w \). By employing the techniques presented in \( [13] \) we can extract a subgraph \( G' = (X, B) \) of \( G_{\text{BIBD}} \) with \( X \subseteq A \), \( |X| = w \), and such that each node of \( X \) has degree 3 and each node of \( B \) has degree either \( \lfloor 3w/z \rfloor \) or \( \lceil 3w/z \rceil \).

Let \( d \) be the largest power of 2 not exceeding \( \lfloor 3w/z \rfloor \), and let \( B = \{ b_i : 0 \leq i < z \} \). We index the edges of \( G' \) so that all edges incident on the same node \( b_i \) have consecutive indices, and, for \( j < i \) edges incident on \( b_j \) have indices smaller than those incident on \( b_i \). Observe that by the BIBD property, for every \( 0 < i < z \) there is at most one node \( x \in X \) adjacent to both \( b_{i-1} \) and \( b_i \). This allows us to rearrange the indices of the edges incident on \( b_i \), for every \( i \), so that if there are two edges \( (x, b_{i-1}) \) and \( (x, b_i) \), for some \( x \in X \), the indices of these two edges differ by at least \( d \). Finally, we construct \( G = (X, Y) \) by grouping the edges of \( G' \) into \( \lfloor 3w/d \rfloor \) bundles of \( d \) consecutively indexed edges each (the last bundle may have less than \( d \) edges), and by creating a distinct node of \( Y \) for each bundle, which becomes the new right endpoint for all edges in the bundle.

The four properties of \( G \) stated in the lemma follow easily from the above construction.

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

4. Access protocol

Suppose that \( n \) shared variables are distributed among the \( n \) D-BSP processors according to the HMOS described above. In this section, we show how any \( n \)-tuple of variables can be efficiently accessed, with every processor requiring read or write access to a distinct variable. (The case of concurrent accesses to the same variable requires only minor modifications that are omitted for brevity.) The access protocol has the same overall structure as the one presented in \( [13] \), but it differs in the implementation which fully exploits the explicit hierarchical nature of the machine model.

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

4.1. Copy selection

The hierarchical structure of the HMOS provides a geographical distribution of the copies into the D-BSP clusters. Copy selection essentially aims at limiting the number of copies that have to be accessed in any block at any level of the HMOS, in order to reduce the traffic into/from the cluster storing the block. Consider a directed version \( \mathcal{H} \) of the HMOS, where edges in every constituent bipartite graph are directed from the left to the right node set, and note that the \( r^{|S|} \) copies of a variable \( v \) are in one-to-one correspondence with the source-sink paths of the subdag \( \mathcal{H}_v \subseteq \mathcal{H} \) induced by \( v \) and by all of its descendants. In order to guarantee consistency, we require that the selected copies for every variable \( v \in S \) form a target set, defined as follows. A set of copies \( C_v \) of \( v \) is a target set for \( v \) if, by coloring the nodes in \( \mathcal{H}_v \), along the paths corresponding to the copies in
Copy selection for the set of variables $S$ is accomplished in $k + 1$ iterations, numbered from 0 to $k$, during which in every $\mathcal{H}_v$, $v \in S$, the paths corresponding to the copies being selected are colored level by level from the source to the sinks. Since, in general, a node of the HMOS belongs to several $\mathcal{H}_v$’s, we use a distinct color $\gamma_v$ for each variable $v \in S$, and allow a node to be assigned a set of colors. Initially, the color set of every node is empty. Iteration $0$ assigns the set $\{\gamma_v\}$ to the source $v$ and adds $\gamma_v$ to the color sets of $\mu$ of its adjacent 0-modules, for every $v \in S$. Iteration $i$, $0 < i \leq k$, selects two adjacent $i$-modules for every colored $(i - 1)$-module $u$, and adds to their color sets the one assigned to $u$. (Clearly, if an $i$-module is selected for two or more colored $(i - 1)$-modules, it receives the union of the color sets of these $(i - 1)$-modules.) In this fashion, for every $v \in S$, at the end of Iteration $i$, the nodes whose color sets contain $\gamma_v$ form exactly $\mu 2^k$ distinct paths from the source to nodes at level $i$ in $\mathcal{H}_v$. We call such paths $\gamma_v$-colored paths. We choose $C_v$ as the set of copies of $v$ corresponding to the $\mu 2^k$ $\gamma_v$-colored paths in $\mathcal{H}_v$ at the end of the last iteration.

We define the weight $w(u)$ of an $i$-module $u \in U_i$, $0 \leq i \leq k$, as the sum, over all $v \in S$, of the number of $\gamma_v$-colored paths containing $u$. The above coloring ensures that for every $u \in U_i$ with nonempty color set, only $2^{k-i}$ $i$-blocks of $u$ contain copies in $\bigcup_{v \in S} C_v$, with exactly $w(u)$ copies per block. Using the expansion properties of the HMOS, we are able to establish suitably low bounds for the $w(u)$’s. The following lemmas provide such bounds and evaluate the running times of the iterations on a D-BSP $(n, g^{(i\alpha)}, \ell^{(3)})$.

**Lemma 2** If $(V, U_0)$ has $(n/m, \epsilon, \mu)$-expansion, Iteration $0$ can be executed in time $O\left( n^{\alpha} + n^3 \right)$, in such a way that, for every $u \in U_0$

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

**Proof (sketch).** At the beginning of the iteration, every processor in charge of a variable $v \in S$ creates $r$ packets, each composed of the processor’s id and the name of a distinct 0-module adjacent to $v$ in $\mathcal{H}_v$. Upon creation, all packets are regarded as unmarked. Then, the following steps are executed until $\mu$ packets for each variable are marked and all other packets are destroyed.

1. **(Sort)** Sort all unmarked packets by their second component (i.e., the associated 0-module).

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

Finally, for every $v \in S$ the nodes in $\mathcal{H}_v$ corresponding to the $\mu$ marked packets picked for $v$ receive color $\gamma_v$. Based on the expansion of $(V, U_0)$ it can be shown [13, Lemma 4.4] that at the end of the whole procedure the bound on $w(u)$ holds for every $u \in U_0$, moreover, after the $i$th execution of the Sort, Select and Count steps, packets of at most $n/2^i$ variables remain unmarked. Hence, the $i$th execution of these three steps can be implemented through $r$-sorting and prefix within an $(i - 1)$-cluster, which yields the desired running time.

**Lemma 3** For $1 \leq i \leq k$, Iteration $i$ can be executed in time $O\left( 2^i n^{((1-\epsilon)/2^i)} \right)$ in such a way that, for every $u \in A_i$,

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

**Proof (sketch).** We employ the same implementation of Iteration $i$ as in [13], which essentially requires a constant number of $O\left( 2^i \right)$-sorting and prefix operations. The running time follows from Propositions 1, 2 and 3, while the bound on $w(u)$ is obtained by exploiting the expansion properties of the graphs defined in Lemma 1. (More details will be provided in the full version of this abstract.)

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

**Theorem 2** Copy selection requires time $O\left( 2^kn^{(\alpha)} + kn^3 \right)$ and ensures that every $i$-block, $0 \leq i \leq k$, contains at most $\left( 2^i n^{((1-\epsilon)/2^i)} \right)$ selected copies.

**4.2. Access to the selected copies**

After copy selection is completed, for every $v \in S$ the processor in charge of $v$ creates $\mu 2^k$ distinct messages to request the copies in $C_v$. Each message is routed to its destination (i.e., the processor whose memory stores the requested copy) where the read/write access is performed. (The return of data to the requesting processors is dealt with in a similar fashion and we omit its description.) Messages are delivered to the destinations in $k + 1$ stages through smaller and smaller clusters, taking advantage of the good distribution of the copies among the $i$-blocks, for every $i$. 


For $1 \leq i \leq k$, let $\tau_i = \log n - \log t_i$ and recall that every $i$-block is assigned to a distinct cluster with $t_i$ processors, that is, a $\tau_i$-cluster, where $t_i = \Theta \left( 3^{i-1} n^{1-1/2} \right)$. Let also $\tau_{k+1} = 0$. The stages are numbered from $k + 1$ down to 1. For $k + 1 \geq i \geq 1$, Stage $i$ is executed in parallel and independently in every $\tau_i$-cluster and sends all messages residing in the cluster to arbitrary positions in the internal $\tau_{i-1}$-clusters associated with their destination $(i-1)$-blocks, in such a way that the processors in the same $\tau_{i-1}$-cluster receive approximately the same number of messages. The intermediate destination of each message is easily established via sorting and ranking. Finally, in Stage 1 every message is sent to its final destination processor, in parallel and independently within every $\tau_1$-cluster.

Let $\delta_i$, be the maximum number of messages held by any processor at the beginning of Stage $i$, $k + 1 \geq i \geq 1$. Based on Theorem 2, the definition of $\tau_i$ and the bound on $t_i$ we have

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

It is easy to see that Stage $i$ can be implemented by means of a constant number of $\delta_i$-sorting, $(\delta_i, \delta_{i-1})$-routings and prefix operations within $\tau_i$-clusters. Therefore, from Propositions 1, 2 and 3 it follows by straightforward calculations that, for $\epsilon \leq \alpha/(2 - \alpha)$, the time required to access the selected copies is

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

By combining this with the complexity of copy selection (Theorem 2) we have:

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

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

In order to achieve the best possible tradeoff between performance and redundancy on a specific D-BSP $(n, g^{(\alpha)}, E^{(\beta)})$, we need to choose a suitable value for $k$ as a function of $\alpha$ and $\beta$. More specifically, when $\alpha < \beta$, it suffices to choose $k = \Theta (1)$ large enough to obtain optimal $O(n^\beta)$ slowdown with constant redundancy. This result demonstrates that optimal worst case slowdown is achievable on machines where delays due to latency dominate over those due to bandwidth. Instead, when $\alpha \geq \beta$, by choosing a suitable $k = \Theta (\log \log n)$, we obtain a minimal slowdown of $\Theta (n^\alpha \log n)$ with redundancy $\Theta \left( \log \log n \right) = O (\log^{1.59} n)$. Two natural questions related to this latter case are whether there exists an alternative strategy achieving $O(n^\alpha)$ slowdown, or whether the same slowdown of our scheme can be achieved by using constant redundancy. Unfortunately, the answers to both these questions are negative. Indeed, a simple adaptation of the lower bound argument given in [9] suffices to show that the minimal slowdown achievable for $\alpha \geq \beta$ by any scheme which dispatches an individual message for each copy to be accessed is

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

and that in order to achieve such slowdown redundancy $\Omega \left( \log (m/n)/\log (m/n) \right)$ is necessary. (More details on the lower bound will be provided in the final version of this work.)

## 5. Constructive issues

It must be remarked that Theorem 3 relies on the $(n/m, \epsilon, \mu)$-expansion of $(V, U_0)$, with $\epsilon < \alpha/(2 - \alpha)$. Although such level of expansion is considerably milder than those required by most schemes in the literature (e.g., $\epsilon = O (1/\log n)$ in [1, 8, 15]), in general we can only show the existence of $(V, U_0)$ through the probabilistic method. However, unlike the aforementioned schemes, ours can take advantage of the few known explicit constructions. For instance, using a BIBD for $(V, U_0)$ and the techniques described in [13], when $\alpha \geq 1/2$ and $m = O (n^{3/2})$ we can obtain fully constructive schemes which achieve

$$
O \left( 2^k n^{\alpha + \epsilon / \beta} + kn^\beta \right)
$$

slowdown, for a suitable constant $\epsilon > 0$. Hence, by choosing appropriate values for $k$, the performances discussed in the previous paragraph can still be obtained. Note that the bound on the memory size for the constructive schemes is sufficient to allow all NC algorithms to be ported from the PRAM to the D-BSP model.

### References


