On the Effectiveness of D-BSP as a Bridging Model of Parallel Computation*

Gianfranco Bilardi, Carlo Fantozzi, Andrea Pietracaprina, and Geppino Pucci

Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy.
{bilardi,fantozzi,andrea,geppo}@artemide.dei.unipd.it

Abstract. This paper surveys and places into perspective a number of results concerning the D-BSP (Decomposable Bulk Synchronous Parallel) model of computation, a variant of the popular BSP model proposed by Valiant in the early nineties. D-BSP captures part of the proximity structure of the computing platform, modeling it by suitable decompositions into clusters, each characterized by its own bandwidth and latency parameters. Quantitative evidence is provided that, when modeling realistic parallel architectures, D-BSP achieves higher effectiveness and portability than BSP, without significantly affecting the ease of use. It is also shown that D-BSP avoids some of the shortcomings of BSP which motivated the definition of other variants of the model. Finally, the paper discusses how the aspects of network proximity incorporated in the model allow for a better management of network congestion and bank contention, when supporting a shared-memory abstraction in a distributed-memory environment.

1 Introduction

The use of parallel computers would be greatly enhanced by the availability of a model of computation that combines the following properties: *usability*, regarded as ease of algorithm design and analysis, *effectiveness*, so that efficiency of algorithms in the model translates into efficiency of execution on some given platform, and *portability*, which denotes the ability of achieving effectiveness with respect to a wide class of target platforms. These properties appear, to some extent, incompatible. For instance, effectiveness requires modeling a number of platform-specific aspects that affect performance (e.g., interconnection topology) at the expense of portability and usability. The formulation of a *bridging model* that balances among these conflicting requirements has proved a difficult task, as demonstrated by the proliferation of models in the literature over the years.

In the last decade, a number of bridging models have been proposed, which abstract a parallel platform as a set of processors and a set of either local or shared memory banks (or both) communicating through some interconnection. In order to ensure usability and portability over a large class of platforms, these models do not provide detailed characteristics of the interconnection but, rather, summarize its communication capabilities by a few parameters that broadly capture bandwidth and latency properties.

* This research was supported, in part, by the Italian CNR, and by the Italian MURST under Project *Algorithms for Large Data Sets: Science and Engineering.*

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 579–588, 2001.

Perhaps the most popular example in this arena is Valiant's *BSP* (*Bulk Synchronous Parallel*) model [Val90]. A BSP machine is a set of n processors with local memory, communicating through a router, whose computations are sequences of *supersteps*. In a superstep, each processor (i) reads the messages received in the previous superstep; (ii) performs computation on locally available data; (iii) sends messages to other processors; and (iv) takes part in a global barrier synchronization. A superstep is charged a cost of $w + gh + \ell$, where w (resp., h) is the maximum number of operations performed (resp., messages sent/received) by any processor in the superstep, and g and ℓ are parameters with g inversely related to the router's bandwidth and ℓ capturing latency and synchronization delays.

Similar to BSP is the LogP model proposed by Culler et al. [CKP$^+$96] which, however, lacks explicit synchronization and imposes a more constrained message-passing discipline aimed at keeping the load of the underlying communication network below a specified capacity limit. A quantitative comparison developed in [BHP$^+$96,BHPP00] establishes a substantial equivalence between LogP and BSP as computational models for algorithm design guided by asymptotic analysis.

In recent years, a number of BSP variants have been formulated in the literature, whose definitions incorporate additional provisions aimed at improving the model's effectiveness relative to actual platforms without affecting its usability and portability significantly (see e.g., [BGMZ95,BDM95,JW96b,DK96]). Among these variants, the *E-BSP* (*Extended BSP*) by [JW96b] and the *D-BSP* (*Decomposable BSP*) by [DK96] are particularly relevant for this paper. E-BSP aims at predicting more accurately the cost of supersteps with *unbalanced* communication patterns, where the average number h_{ave} of messages sent/received by a processor is lower than the corresponding maximum number, h. Indeed, on many interconnections, routing time increases with h_{ave}, for fixed h, a phenomenon modeled in E-BSP is by adding a term depending upon h_{ave} to the cost of a superstep. However, the functional shape of this term varies with the topology of the intended target platform, making the model somewhat awkward.

D-BSP extends BSP by incorporating some aspects of network proximity into the model. Specifically, the set of n processor/memory pairs is viewed as partitionable as a collection of clusters, where each cluster is able to perform its own sequence of supersteps independently of the other ones and is characterized by its own g and ℓ parameters, typically increasing with the size of the cluster. The partition into clusters can change dynamically within a pre-specified set of legal partitions. The key advantage is that communication patterns where messages are confined within small clusters have small cost, like in realistic platforms and unlike in standard BSP. In fact, it can be shown quantitatively that this advantage translates into higher effectiveness and portability of D-BSP over BSP. Clustering also enables efficient routing of unbalanced communication patterns in D-BSP, making it unnecessary to further extend the cost model in the direction followed by E-BSP. Thus, D-BSP is an attractive candidate among BSP variants and, in general, among bandwidth-latency models, to strike a fair balance among the conflicting features sought in a bridging model of parallel computation.

In Section 2, we define a restricted version of D-BSP where clusters are defined according to a regular recursive structure, which greatly simplifies the use of the model without diminishing its power significantly. In Section 3, we employ the methodology based on cross-simulations proposed in [BPP99] to quantitatively assess the higher

effectiveness of D-BSP with respect to BSP, relatively to the wide class of processor networks. Then, in Subsection 3.1 we show that, for certain relevant computations and prominent topologies, D-BSP exhibits a considerably higher effectiveness than the one guaranteed by the general result. In such cases, the effectiveness of D-BSP becomes close to optimal. Furthermore, we present a general strategy to exploit communication locality: one of the corollaries is a proof that D-BSP can be as effective as E-BSP in dealing with unbalanced communication patterns. Finally, in Section 4 we show how D-BSP can efficiently support a shared memory abstraction, a valuable provision for algorithm development in a distributed-memory environment. The results presented in the section clearly indicate that the network proximity modeled by D-BSP can be exploited to reduce network congestion and bank contention when implementing a shared address space both by randomized and by deterministic strategies.

2 The D-BSP Model

The D-BSP (Decomposable BSP) model was introduced in [DK96] as an extension of Valiant's BSP [Val90] aimed at capturing, in part, the proximity structure of the network. In its most general definition, the D-BSP is regarded as a set of n processor/memory pairs communicating through a router, which can be aggregated according to a predefined collection of submachines, each able to operate independently. For concreteness, we focus our attention on a restricted version of the model (referred to as *recursive D-BSP* in [DK96]) where the collection of submachines has the following regular structure. Let n be a power of two. For $0 \leq i \leq \log n$, the n processors are partitioned into 2^i fixed, disjoint *i-clusters* $C_0^{(i)}, C_1^{(i)}, \cdots, C_{2^i-1}^{(i)}$ of $n/2^i$ processors each, where the processors of a cluster are able to communicate among themselves independently of the other clusters. The clusters form a hierarchical, binary decomposition tree of the D-BSP machine: specifically, $C_j^{\log n}$ contains only processor P_j, for $0 \leq j < n$, and $C_j^{(i)} = C_{2j}^{(i+1)} \cup C_{2j+1}^{(i+1)}$, for $0 \leq i < \log n$ and $0 \leq j < 2^i$.

A D-BSP computation consists of a sequence of labelled supersteps. In an *i-superstep*, $0 \leq i \leq \log n$, each processor executes internal computation on locally held data and sends messages exclusively to processors within its i-cluster. The superstep is terminated by a barrier, which synchronizes processors within each i-cluster independently. It is assumed that messages sent in one superstep are available at the destinations only at the beginning of the subsequent superstep. Let $\boldsymbol{g} = (g_0, g_1, \ldots, g_{\log n})$ and $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{\log n})$. If each processor performs at most w local operations, and the messages sent in the superstep form an *h-relation* (i.e., each processor is source or destination of at most h messages), then, the cost of the i-superstep is upper bounded by $w + hg_i + \ell_i$, for $0 \leq i \leq \log n$. Parameters g_i and ℓ_i are related to the bandwidth and latency guaranteed by the router when communication occurs within i-clusters. We refer to such a model as a D-BSP $(n, \boldsymbol{g}, \boldsymbol{\ell})$.

Note that the standard BSP(n, g, ℓ) defined by Valiant can be regarded as a D-BSP $(n, \boldsymbol{g}, \boldsymbol{\ell})$ with $g_i = g$ and $\ell_i = \ell$ for every i, $0 \leq i \leq \log n$. In other words, D-BSP introduces the notion of proximity in BSP through clustering, and groups h-relations into specialized classes associated with different costs. This ensures full compatibility

between the two models, which allows programs written according to one model to run on any machine supporting the other, the only difference being their estimated performance.

In this paper, we will often exemplify our considerations by focusing on a class of parameter values for D-BSP of particular significance. Namely, let α and β be two arbitrary constants, with $0 < \alpha, \beta < 1$. We will consider D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ machines with

$$\begin{cases} g_i^{(\alpha)} = G \cdot (n/2^i)^\alpha, \\ \ell_i^{(\beta)} = L \cdot (n/2^i)^\beta, \end{cases} \quad 0 \leq i \leq \log n,$$

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, such as, for example, multidimensional arrays.

3 D-BSP and Processor Networks

Intuitively, a computational model M, where designers develop and analyze algorithms, is *effective* with respect to a platform M', on which algorithms are eventually implemented and executed, if the algorithmic choices based on M turn out to be the right choices in relation to algorithm performance on M'. In other words, one hopes that the relative performance of any two algorithms developed on M reflects the relative performance of their implementations on M'. In order to attain generality, a computational model abstracts specific architectural details (e.g., network topology), while it incorporates powerful features that simplify its use but may not be exhibited by certain platforms. Therefore, in order to evaluate the effectiveness of a model M with respect to a platform M', we must establish the performance loss incurred by running algorithms developed for M (which do not exploit platform-specific characteristics) on M', and, on the other hand, we must assess how efficiently features offered by M to the algorithm designer, can be supported on M'.

More precisely, let us regard M and M' as two different machines, and define $S(M, M')$ (resp., $S(M', M)$) as the minimum slowdown needed for simulating M on M' (resp., M' on M). In [BPP99] a quantitative measure of effectiveness of M with respect to M' is given, and it is shown that the product $\delta(M, M') = S(M, M')S(M', M)$ provides an upper bound to this measure. Namely, effectiveness decreases with increasing $\delta(M, M')$ and is highest for $\delta(M, M') = 1$. When maximized over all platforms M' in a given class, this quantity provides an upper measure of the portability of M with respect to the class.

We use this approach to evaluate the effectiveness of D-BSP with respect to the class of processor networks. Let G be a connected n-processor network, where in one *step* each processor executes a constant number of local operations and may send/receive one point-to-point message to/from each neighboring processor (multi-port regimen). As is the case for all relevant network topologies, we assume that G has a decomposition tree $\{G_0^{(i)}, G_1^{(i)}, \cdots, G_{2^i-1}^{(i)} \ : \ \forall i,\, 0 \leq i \leq \log n\}$, where each $G_j^{(i)}$ is a connected subnet (i-*subnet*) with $n/2^i$ processors and $G_j^{(i)} = G_{2j}^{(i+1)} \cup G_{2j+1}^{(i+1)}$. By combining the routing results of [LR99,LMR99] one can easily show that for every $0 \leq i \leq \log n$ there exist suitable values g_i and ℓ_i related, respectively, to the bandwidth and diameter characteristics of the i-subnets, such that an h-relation followed by a barrier

synchronization within an i-subnet can be implemented in $O(hg_i + \ell_i)$ time. Let M be a D-BSP $(n, \boldsymbol{g}, \boldsymbol{\ell})$ with these particular g_i and ℓ_i values, for $0 \leq i \leq \log n$. Clearly, we have that

$$S(M, G) = O(1) \quad . \tag{1}$$

Vice-versa, an upper bound to the slowdown incurred in simulating G on M is provided by the following theorem proved in [BPP99, Theorem 3].

Theorem 1. *Suppose that at most b_i links connect every i-subnet to the rest of the network, for $1 \leq i \leq \log n$. Then, one step of G can be simulated on the D-BSP $(n, \boldsymbol{g}, \boldsymbol{\ell})$ in time*

$$S(G, M) = O\left(\min_{n' \leq n}\left\{\frac{n}{n'} + \sum_{i=\log(n/n')}^{\log n - 1}(g_i \max\{h_i, h_{i+1}\} + \ell_i)\right\}\right) \quad , \tag{2}$$

where $h_i = \lceil b_{i-\log(n/n')}/(n/2^i)\rceil$, for $0 \leq i \leq \log n$.

We can apply Equations 1 and 2 to quantitatively estimate the effectiveness of D-BSP with respect to specific network topologies. Consider, for instance, the case of an n-node d-dimensional array. Fix $g_i = \ell_i = (n/2^i)^{1/d}$, for $0 \leq i \leq \log n$. Such D-BSP $(n, \boldsymbol{g}, \boldsymbol{\ell})$ can be simulated on G with constant slowdown. Since G has a decomposition tree with subnets $G_j^{(i)}$ that have $b_i = O\left((n/2^i)^{(d-1)/d}\right)$, the D-BSP simulation quoted in Theorem 1 yields a slowdown of $S(G, M) = O\left(n^{1/(d+1)}\right)$ per step. In conclusion, letting M be the D-BSP with the above choice of parameters, we have that $\delta(M, G) = O\left(n^{1/(d+1)}\right)$. The upper bound on $S(G, M)$ can be made exponentially smaller when each array processor has constant-size memory. In this case, by employing a more sophisticated simulation strategy, we can get $S(G, M) = O\left(2^{\Theta(\sqrt{\log n})}\right)$ [BPP99], thus significantly improving D-BSP's effectiveness.

It is important to remark that the D-BSP clustered structure provides a crucial contribution to the model's effectiveness. Indeed, it can be shown that, if M' is a BSP(n, g, ℓ) and G is a d-dimensional array, then $\delta(M', G) = \Omega\left(n^{1/d}\right)$ independently of g, l and the size of the memory at each processor [BPP99]. This implies that, under the δ metric, D-BSP is asymptotically more effective than BSP with respect to multidimensional arrays.

3.1 Effectiveness of D-BSP with Respect to Specific Computations

We note that non-constant slowdown for simulating an *arbitrary* computation of a processor network on a D-BSP is to be expected since the D-BSP disregards the fine structure of the network topology, and, consequently, it is unable to fully exploit topological locality. However, for several prominent topologies and several relevant computational problems arising in practical applications, the impact of such a loss of locality is much less than what the above simulation results may suggest, and, in many cases, it is negligible.

Consider, for example, the class of processor networks G whose topology has a recursive structure with bisection bandwidth $O\left(n^{1-\alpha}\right)$ and diameter $O\left(n^{\beta}\right)$, for arbitrary constants $0 < \alpha, \beta < 1$ (note that multidimensional arrays belong to this class).

In this case, the results of [LR99,LMR99] imply that a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ M can be efficiently supported on G, so that $S(M, G) = O(1)$. Hence, algorithms devised on M can be implemented on G with at most constant loss of efficiency. Although the reverse slowdown $S(G, M)$ is in general non-constant, we now show that M allows us to develop algorithms for a number of relevant computational problems, which exhibit optimal performance when run on G. Hence, for these problems, the loss of locality of M with respect to G is negligible.

Let $N \geq n$ items be evenly distributed among the n processors. A parallel prefix on these items (N-*prefix*) requires time $\Omega\left(N/n + n^{\beta}\right)$ to be performed on G [Lei92]. On the D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ there is a parallel prefix algorithm that runs in time $O\left(N/n + n^{\alpha} + n^{\beta}\right)$ [DK96]. Clearly, when $\alpha \leq \beta$ the implementation of the D-BSP algorithm on G exhibits optimal performance. We remark that optimality cannot be obtained using the standard BSP model. Indeed, results in [Goo96] imply that the direct implementation on G of any BSP algorithm for N-prefix, runs in time $\Omega\left(n^{\beta} \log n / \log(1 + \lceil n^{\beta-\alpha} \rceil)\right)$, which is not optimal, for instance, when $\alpha = \beta$.

We call k-*sorting* a sorting problem in which k keys are initially assigned to each one of n processors and are to be redistributed so that the k smallest keys will be held by processor P_0, the next k smallest ones by processor P_1, and so on. It is easy to see that k-sorting requires time $\Omega\left(kn^{\alpha} + n^{\beta}\right)$ to be performed on G because of the bandwidth and diameter of the network. On the D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ there is an algorithm that performs k-sorting in time $O\left(kn^{\alpha} + n^{\beta}\right)$ [FPP01], which is clearly optimal when ported to G. Again, a similar result is not possible with standard BSP: the direct implementation on G of any BSP algorithm for k-sorting runs in time $\Omega\left((\log n / \log k)(kn^{\alpha} + n^{\beta})\right)$, which is not optimal for small k.

As a final important example, consider a (k_1, k_2)-*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 a greedy routing strategy where all packets are delivered in one superstep requires $\Theta\left(\max\{k_1, k_2\} \cdot n^{\alpha} + n^{\beta}\right)$ time on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$, which is the best one could do on a BSP$(n, n^{\alpha}, n^{\beta})$. However, a careful exploitation of the submachine locality exhibited by the D-BSP yields a better algorithm for (k_1, k_2)-routing, which runs in time $O\left(k_{\min}^{\alpha} k_{\max}^{1-\alpha} n^{\alpha} + n^{\beta}\right)$, where $k_{\min} = \min\{k_1, k_2\}$ and $k_{\max} = \max\{k_1, k_2\}$ [FPP01]. Indeed, standard lower bound arguments show that such a routing time is optimal for G [SK94].

As a corollary of the above routing result, we can show that, unlike the standard BSP model, D-BSP is also able to handle unbalanced communication patterns efficiently, which was the main objective that motivated the introduction of a BSP variant, called E-BSP, by [JW96a]. Let an (h, m)-relation be a routing instance where each processor sends/receives at most h messages, and a total of m messages are exchanged. Although a greedy routing strategy for an (h, m)-relation requires time $\Theta\left(hn^{\alpha} + n^{\beta}\right)$ on both D-BSP and BSP, the exploitation of submachine locality in D-BSP allows us to route any (h, m)-relation in time $O\left(\lceil m/n \rceil^{\alpha} h^{1-\alpha} n^{\alpha} + n^{\beta}\right)$, which is equal or smaller than the greedy routing time and it is optimal for G. Consequently, D-BSP can be as effective as E-BSP in dealing with unbalanced communication as E-BSP, where the treatment of unbalanced communication is a primitive of the model.

4 Providing Shared Memory on D-BSP

A very desirable feature of a distributed-memory model is the ability to support a shared memory abstraction efficiently. Among the other benefits, this feature allows porting the vast body of PRAM algorithms [JáJ92] to the model at the cost of a small time penalty. In this section we present a number of results that demonstrate that D-BSP can be endowed with an efficient shared memory abstraction.

Implementing shared memory calls for the development of a *scheme* to represent m shared cells (*variables*) among the n processor/memory pairs of a distributed-memory machine 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 often referred to as the *slowdown* of the scheme.

Numerous randomized and deterministic schemes have been developed in the literature for a number of specific processor networks. Randomized schemes (see e.g., [CMS95,Ran91]) usually distribute the variables randomly among the memory modules local to the processors. As a consequence of such a scattering, a simple routing strategy is sufficient to access any n-tuple of variables efficiently, with high probability. Following this line, we can give a simple, randomized scheme for shared memory access on D-BSP. Assume, for simplicity, that the variables be spread among the local memory modules by means of a totally random function. In fact, a polynomial hash function drawn from a $\log n$-universal class [CW79], suffices to achieve the same results [MV84], but it takes only poly$(\log n)$ rather than $O(n \log n)$ random bits to be generated and stored at the nodes. We have:

Theorem 2. *Any n-tuple of memory accesses on a D-BSP $(n, \boldsymbol{g}, \boldsymbol{\ell})$ can be performed in time*

$$O\left(\sum_{i=0}^{\lfloor \log(n/\log n)\rfloor - 1} T_{\mathrm{pr}}(i) + g_{\lfloor \log(n/\log n)\rfloor} \frac{\log n}{\log\log n} + \ell_{\lfloor \log(n/\log n)\rfloor}\right) \qquad (3)$$

with high probability, where $T_{\mathrm{pr}}(i)$ denotes the time of a prefix-sum operation within an i-cluster.

Proof (Sketch). Consider the case of write accesses. The algorithm consists of $\lfloor \log(n/\log n)\rfloor + 1$ steps. More specifically, in Step i, for $1 \leq i \leq \lfloor \log(n/\log n)\rfloor$, we send the messages containing the access requests to their destination i-clusters, so that each node in the cluster receives roughly the same number of messages. A standard occupancy argument [MR95] suffices to show that, with high probability, there will be no more than $\lambda n/2^i$ messages destined to the same i-cluster, for a given small constant $\lambda > 1$, hence each step requires a simple prefix and the routing of an $O(1)$-relation in i-clusters. In the last step, we simply send the messages to their final destinations, where the memory access is performed. Again, the same probabilistic argument implies that the degree of the relation in this case is $O(\log n/\log\log n)$, with high probability.

For read accesses, the return journey of the messages containing the accessed values can be performed by reversing the algorithm for writes, thus remaining within the same time bound.

By plugging in the time for prefix in Eq. (3) we obtain:

Corollary 1. *Any n-tuple of memory accesses can be performed in optimal time $O\left(n^{\alpha}+n^{\beta}\right)$, with high probability, on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$.*

Observe that under a uniform random distribution of the variables among the memory modules, $\Theta\left(\log n/\log\log n\right)$ out of *any* set of n variables will be stored in the same memory module, with high probability, hence any randomized access strategy without replication would require at least $\Omega\left(n^{\alpha}\log n/\log\log n+n^{\beta}\right)$ time on a BSP$(n, n^{\alpha}, n^{\beta})$.

Let us now switch to deterministic schemes. In this case, achieving efficiency is much harder, since, in order to avoid the trivial worst-case where a few memory modules contain all of the requested data, we are forced to replicate each variable and manage replicated copies so to enforce consistency. A typical deterministic scheme replicates every variable into ρ copies, which are then distributed among the memory modules through a map exhibiting suitable expansion properties. Expansion is needed to guarantee that the copies relative to any n-tuple of variables be never confined within few nodes. The parameter ρ is referred to as the *redundancy* of the scheme. In order to achieve efficiency, the main idea, originally introduced in [UW87] 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 maximize communication bandwidth while ensuring consistency (i.e., a read access must always return the most updated value of the variable).

A general deterministic scheme to implement a shared memory abstraction on a D-BSP is presented in [FPP01]. The scheme builds upon the one in [PPS00] for a number of processor networks, whose design exploits the recursive decomposition of the underlying topology to provide a hierarchical, redundant representation of the shared memory based on $k+1$ levels of logical modules. Such an organization fits well with the structure of a D-BSP, which is hierarchical in nature. More specifically, each variable is replicated into $r=O\left(1\right)$ copies, and the copies are assigned to r logical modules of level 0. In general, the logical modules at the i-th level, $0\leq i<k$ are replicated into three copies, which are assigned to three modules of level $i+1$. This process eventually creates $r3^{k}=\Theta\left(3^{k}\right)$ copies of each variable, and 3^{k-i} replicas of each module at level i. The number (resp., size) of the logical modules decreases (resp., increases) with the level number, and their replicas are mapped to the D-BSP by assigning each distinct block to a distinct cluster of appropriate size, so that each of the sub-blocks contained within the block is recursively assigned to a sub-cluster.

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 [Hal86], and can always be explicitly constructed.

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 on the D-BSP through a protocol similar to the one in [PPS00], which can be implemented through a combination of prefix, sorting and (k_1, k_2)- routing primitives. By employing

the efficient D-BSP implementations for these primitives discussed in Section 3, the following result is achieved on a D-BSP $(n, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ [FPP01, Corollary 1].

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

An interesting consequence of the above theorem is that it shows 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. When this is not the case, it is shown in [FPP01] that the proposed scheme is not too far-off from being optimal.

Perhaps, the most important feature of the above scheme is that, unlike the other classical deterministic schemes in the literature, it solely relies on expander graphs of mild expansion, hence it can be made fully constructive for a significant range of the parameters involved. Such mild expanders, however, are only able to guarantee that the copies of an arbitrary n-tuple of variables be spread among $O\left(n^{1-\epsilon}\right)$ memory modules, for some constant $\epsilon < 1$. Hence the congestion at a single memory module can be as high as $O\left(n^{\epsilon}\right)$ and the clusterized structure of D-BSP is essential in order to achieve good slowdown. In fact, any deterministic strategy employing these graphs on a BSP(n, g, ℓ) could not achieve better than $\Theta\left(gn^{\epsilon}\right)$ slowdown.

References

[BDM95] A. Bäumker, W. Dittrich, and F. Meyer auf der Heide. Truly efficient parallel algorithms: c-optimal multisearch for and extension of the BSP model. In *Proc. of the 3rd European Symposium on Algorithms*, pages 17–30, 1995.

[BGMZ95] G.E. Blelloch, P.B. Gibbons, Y. Matias, and M. Zagha. Accounting for memory bank contention and delay in high-bandwidth multiprocessors. In *Proc. of the 7th ACM Symp. on Parallel Algorithms and Architectures*, pages 84–94, Santa Barbara, CA, July 1995.

[BHP+96] G. Bilardi, K.T. Herley, A. Pietracaprina, G. Pucci, and P. Spirakis. BSP vs LogP. In *Proc. of the 8th ACM Symp. on Parallel Algorithms and Architectures*, pages 25–32, 1996. To appear in *Algorithmica*, Special Issue on Coarse Grained Parallel Algorithms.

[BHPP00] G. Bilardi, K. Herley, A. Pietracaprina, and G. Pucci. On stalling in LogP. In *Proc. of the Workshop on Advances in Parallel and Distributed Computational Models*, LNCS 1800, pages 109–115, May 2000.

[BPP99] G. Bilardi, A. Pietracaprina, and G. Pucci. A quantitative measure of portability with application to bandwidth-latency models for parallel computing. In *Proc. of EUROPAR 99*, LNCS 1685, pages 543–551, September 1999.

[CKP+96] D.E. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K.E. Schauser, R. Subramonian, and T.V. Eicken. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, November 1996.

[CMS95] A. Czumaj, F. Meyer auf der Heide, and V. Stemann. Shared memory simulations with triple-logarithmic delay. In *Proc. of the 3rd European Symposium on Algorithms*, pages 46–59, 1995.

[CW79] J.L. Carter and M.N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[DK96] P. De la Torre and C.P. Kruskal. Submachine locality in the bulk synchronous setting. In *Proc. of EUROPAR 96*, LNCS 1124, pages 352–358, August 1996.

[FPP01] C. Fantozzi, A. Pietracaprina, and G. Pucci. Implementing shared memory on clustered machines. In *Proc. of 2nd International Parallel and Distributed Processing Symposium*, 2001. To appear.

[Goo96] M.T. Goodrich. Communication-efficient parallel sorting. In *Proc. of the 28th ACM Symp. on Theory of Computing*, pages 247–256, Philadelphia, Pennsylvania USA, May 1996.

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

[JáJ92] J. JáJá. *An Introduction to Parallel Algorithms*. Addison Wesley, Reading MA, 1992.

[JW96a] B.H.H. Juurlink and H.A.G. Wijshoff. The E-BSP model: Incorporating general locality and unbalanced communication into the BSP model. In *Proc. of EUROPAR 96*, LNCS 1124, pages 339–347, August 1996.

[JW96b] B.H.H. Juurlink and H.A.G. Wijshoff. A quantitative comparison of paralle computation models. In *Proc. of the 8th ACM Symp. on Parallel Algorithms and Architectures*, pages 13–24, June 1996.

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

[LMR99] F.T. Leighton, B.M. Maggs, and A.W. Richa. Fast algorithms for finding O(congestion + dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999.

[LR99] F.T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge MA, 1995.

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

[PPS00] A. Pietracaprina, G. Pucci, and J. Sibeyn. Constructive, deterministic implementation of shared memory on meshes. *SIAM Journal on Computing*, 30(2):625–648, 2000.

[Ran91] A.G. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42:307–326, 1991.

[SK94] J.F. Sibeyn and M. Kaufmann. Deterministic 1-k routing on meshes, with application to hot-potato worm-hole routing. In *Proc. of the 11th Symp. on Theoretical Aspects of Computer Science*, LNCS 775, pages 237–248, 1994.

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

[Val90] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.