A Quantitative Measure of Portability with Application to Bandwidth-Latency Models for Parallel Computing*

(Extended Abstract)

Gianfranco Bilardi, Andrea Pietracaprina, and Geppino Pucci

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

Abstract. We introduce a novel methodology for the quantitative assessment of the effectiveness and portability of models of parallel computation. Specifically, we relate the effectiveness of a model M, adopted for algorithm design, with respect to a platform M', where algorithms developed for M are ultimately executed, to the product of cross-simulation slowdowns between M and M'. The portability of M with respect to a class of platforms can be estimated by its minimum effectiveness over the platforms in the class. We apply our methodology to assess the portability of enhanced variants of the BSP model with respect to processor networks, with particular emphasis on multidimensional arrays.

1 Introduction

It is widely recognized that the choice of a model for parallel computation is made particularly hard by the desire of simultaneously achieving usability, effectiveness, and portability. *Usability* refers to the ease of algorithm design and analysis. *Effectiveness* means that efficiency of algorithms in the model translates into efficiency of execution on some given platform. *Portability* 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., data layout, task assignment, scheduling), at the expense of portability and usability.

The quest for a suitable model of parallel computation has been ongoing for over two decades, as witnessed by the proliferation of models in the literature (see [MMT95] for a survey). One elusive aspect of this quest is the lack of a systematic framework to compare and evaluate candidate models at a quantitative level. Typically, the usability of a model is illustrated by exhibiting efficient algorithms for key computational problems, while effectiveness and portability are argued on the basis of qualitative considerations on the essential features to be expected of present and future machines. Simulations are sometimes advocated as a tool to compare models. For example, there are many results of the form: "model M_1 is more powerful than model M_2", meaning that M_1 can simulate M_2 with constant slowdown, but not vice versa. However, such characterization of power does not directly relate to effectiveness: in fact, if M_1 derives its greater power by capabilities that are available on the target platforms, then M_1 is likely to be more effective than M_2, but when M_1 derives its greater power by assuming lower cost for

* This research was supported, in part, by MURST of Italy under Project MOSAICO, and by CNR of Italy under grant CB97.05085.CT12.

primitives that are hard to implement on the target machines, M_2 might well be more effective than M_1.

In this paper, we propose and begin to explore a methodology for studying effectiveness and portability in a quantitative way. Namely, in Section 2 we argue that, if a model M and a platform M' can simulate each other with respective slowdowns $S(M, M')$ and $S(M', M)$, then the quantity $\delta(M, M') = S(M, M')S(M', M)$ provides an upper measure of effectiveness for M with respect to M'. 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. Next, we apply our methodology to compare the effectiveness and portability of the *Bulk-Synchronous Parallel (BSP)* model [Val90] and one of its variants, the *Decomposable BSP (D-BSP)* [DK96], with respect to the class of *processor networks*.

A BSP(p, g, ℓ) machine consists of a set of p processor/memory pairs communicating through a router, whose computation is organized in supersteps. Each superstep embodies a phase where processors compute on locally available data, a phase where processors exchange messages, and a phase of global synchronization. The 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 that superstep, and g and ℓ are parameters capturing bandwidth-latency characteristics of the router. D-BSP is defined similarly, with an additional provision that allows the computation to proceed independently within subsets of processors, each subset being characterized by its on bandwidth and latency parameters. This mechanism aims at capturing the locality of communication and synchronization. (See Section 3 for a formal definition of D-BSP.) Both BSP and D-BSP attempt to strike a balance between usability, effectiveness and portability by allowing programs to be written in terms of the bandwidth and latency parameters. Specific values for these parameters are selected at run-time to adapt to the characteristics of the target platform. Conceivably, parameterization affords higher portability, while it introduces only a moderate burden on the algorithm developer.

It is straightforward to show that suitable choices of the BSP or D-BSP parameters allow these models to be simulated on most networks of interest with only constant slowdown (with the above notation, $S(\text{BSP}, M'), S(\text{D-BSP}, M') = O(1)$, with M' a processor network). Hence, to study the effectiveness of such models, in Section 4 we concentrate on the simulation of an arbitrary processor network M' on BSP and D-BSP, determining a lower bound to $S(M', \text{BSP})$ and an upper bound on $S(M', \text{D-BSP})$. When M' is a multidimensional array, $S(M', \text{D-BSP})$ turns out to be considerably smaller than the optimal achievable slowdown $S(M', \text{BSP})$. Based on these results, we conclude that, under the δ metric, D-BSP is more effective than BSP for such networks.

Finally, in Section 5, we show that although no specific provisions are made in D-BSP to deal with unbalanced communication patterns, the model, unlike BSP, can efficiently cope with such patterns, which is an indication that further complications to the model, as introduced in other BSP variants (e.g., Y-BSP and E-BSP [DK96,JW96]), may not be needed to achieve higher effectiveness.

2 A Quantitative Approach to Effectiveness and Portability

Let us consider a model M where designers develop and analyze programs, which we call M*-programs*, and a platform M' onto which M-programs are translated and exe-

cuted. We call M'-*programs* the programs that are ultimately executed on M'. During the design process, choices between different programs (possibly coding alternative algorithms for the same problem) will be clearly guided by model M, by comparing their running times as predicted by the model's cost function. Intuitively, we consider M to be effective with respect to M' if the choices based on M turn out to be the right ones in relation to program performance on M'. In other words, we hope that the relative performance of any two M-programs reflects the relative performance of their "counterparts" on M'. The difficulty with this approach is that of clearly defining the association between M-programs and M'-programs, which we do as follows.

We postulate the existence of an equivalence relation ρ among both M-programs and M'-programs and restrict our attention to ρ-*optimal* programs, where a ρ-optimal M-program (resp., M'-program) is the fastest among all M-programs (resp., M'-programs) ρ-equivalent to it. Let us define Π and Π' to be the sets of ρ-optimal M-programs and M'-programs, respectively. We also define a *translation function* σ that associates a program $\pi \in \Pi$ with its ρ-equivalent counterpart $\sigma(\pi) \in \Pi'$. In other words, for every M-program, σ identifies a translation into an M'-program which is consistent with the choice of ρ. Several choices for ρ are possible. For instance ρ-equivalent programs could be those that have the same input-output map (which, however, makes σ rather difficult to obtain in practice), or those that implement the same high-level algorithm (which makes σ more realistic). For simplicity, we will not formally specify ρ but, when applying our methodology, we will only give an informal description of its properties.

For $\pi \in \Pi$ and $\pi' \in \Pi'$ we let $T(\pi)$ and $T'(\pi')$ denote their running times on M and M', respectively. We propose the following *inverse* measure of effectiveness

$$\eta(M, M') = \max_{\pi_1, \pi_2 \in \Pi} \frac{T(\pi_1)}{T(\pi_2)} \cdot \frac{T'(\sigma(\pi_2))}{T'(\sigma(\pi_1))} \ . \tag{1}$$

Note that $\eta(M, M') \geq 1$, where a value close to 1 implies a high effectiveness of M with respect to M', while a large value indicates that the relative performance of programs on M may not be preserved on M'. If, rather than a single platform M', we consider a class $\mathcal{C}$ of target platforms, then the quantity $\eta_{\mathcal{C}}(M) = \max_{M' \in \mathcal{C}} \eta(M, M')$ provides an *inverse* measure of portability of model M.

Next, we show that an upper estimate of $\eta(M, M')$ can be obtained based on the ability of M and M' to simulate each other. Consider an algorithm that takes any M-program π and simulates it on M' as an M'-program π' ρ-equivalent to π. (Note that in neither π nor π' needs be ρ-optimal.) We define the *slowdown* $S(M, M')$ of the simulation as the ratio $T'(\pi')/T(\pi)$ maximized over all possible M-programs π. In practice, $S(M, M')$ represents the (worst-case) cost for supporting model M on M'. We define $S(M', M)$ in a similar fashion, by interchanging the roles of M and M'. Then, we define the *distance* between M and M' to be the quantity

$$\delta(M, M') = S(M, M')S(M', M) \ , \tag{2}$$

and show that the following key inequality is satisfied:

$$\eta(M, M') \leq \delta(M, M') \ . \tag{3}$$

Indeed, since the simulation algorithms considered in the definition of $S(M, M')$ and $S(M', M)$ preserve ρ-equivalence, it is easy to see that for any $\pi_1, \pi_2 \in \Pi$,

$T'(\sigma(\pi_2)) \leq S(M, M')T(\pi_2)$ and $T(\pi_1) \leq S(M', M)T'(\sigma(\pi_1))$. Thus, we have that $(T(\pi_1)/T(\pi_2)) \cdot (T'(\sigma(\pi_2))/T'(\sigma(\pi_1))) \leq \delta(M, M')$, which implies that $\delta(M, M')$ is an upper bound to $\eta(M, M')$.

3 D-BSP: Capturing Network Proximity in BSP

The *Decomposable Bulk Synchronous Parallel* model (D-BSP) was originally introduced in [DK96] as an extension of BSP. In this paper we refer to a weaker variant[1] of the model which can be defined as follows. Let $\boldsymbol{g} = (g_0, g_1, \ldots, g_{\log p})$ and $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{\log p})$. A D-BSP $(p, \boldsymbol{g}, \boldsymbol{\ell})$ is a collection of p processor-memory pairs communicating through a router. For $0 \leq i \leq \log p$, the p processors are partitioned into 2^i fixed, disjoint *i-clusters* $C_0^{(i)}, C_1^{(i)}, \cdots, C_{2^i-1}^{(i)}$ of $p/2^i$ processors each, where the processors of a cluster are able to communicate and synchronize among themselves independently of the other clusters. The clusters form a binary decomposition tree for the D-BSP: specifically, $C_j^{(\log p)}$ contains only Processor j, for $0 \leq j < p$, and $C_j^{(i)} = C_{2j}^{(i+1)} \cup C_{2j+1}^{(i+1)}$, for every $0 \leq i < \log p$ and $0 \leq j < 2^i$. Parameters g_i and ℓ_i are related to the bandwidth and latency guaranteed by the router when communication occurs within i-clusters, for $0 \leq i \leq \log p$.

Analogously to BSP, a p-processor D-BSP computation consists of a sequence of *labeled supersteps*, where labels range in the set $\{0, 1, \ldots \log p\}$. In a superstep of label i, communication and synchronization occurs exclusively within i-clusters. If, in an i-superstep, each processor performs at most w local operations, and the messages exchanged form an h-relation, then the cost of the superstep is $w + hg_i + \ell_i$.

Note that the standard BSP(p, g, ℓ) can be regarded as a D-BSP $(p, \boldsymbol{g}, \boldsymbol{\ell})$ with $g_i = g$ and $\ell_i = \ell$ for every i, $0 \leq i \leq \log p$, hence a valid D-BSP algorithm is also a valid BSP algorithm. Vice versa, any valid BSP(p, g, ℓ) algorithm is also valid for any D-BSP $(p, \boldsymbol{g}, \boldsymbol{\ell})$ with $g_0 = g$ and $\ell_0 = \ell$. This ensures full compatibility between the two models, which differ only with respect to their cost functions. In particular, D-BSP introduces the notion of proximity in BSP through clustering, and groups h-relations into specialized classes associated with different costs.

For suitable choices of the parameters, both BSP and D-BSP can be supported (i.e., simulated) with no loss of efficiency on processor networks. For example, by setting $g = \ell = p^{1/d}$ for BSP and $g_i = \ell_i = (p/2^i)^{1/d}$, $0 \leq i \leq \log p$, for D-BSP, both models can be simulated with constant slowdown on a d-dimensional array. (See [DK96] for other examples.)

4 Effectiveness of Bulk Synchronous Models for Processor Networks

In this section, we consider the issue of simulating processor networks on BSP and D-BSP. In the light of our previous discussion, our goal is to provide quantitative evidence of how the richer structure exposed in D-BSP improves its effectiveness with respect to the "flat" BSP.

[1] Our variant is weaker in the sense that the algorithms for the variant immediately translate into algorithms for the original model of [DK96] with no loss in performance.

BSP 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). We can prove the following general lower bound for the simulation of G by BSP.

Theorem 1. *For any connected N-node network G of bounded degree and every integer $T \geq 1$ there exists a T-step computation of G that requires time*

$$\Omega\left(T\left(\frac{N}{p} + \min\{g, T, N\}\right) + \ell\right)$$

to be simulated by BSP(p, g, ℓ). The lower bound holds even if at the beginning of the simulation each BSP processor knows the initial state of every node of G, and if the operations performed by a node of G may be simulated by more than one processor.

Proof (Sketch). Since G embeds the N-node linear array A_N with constant dilation, it suffices to derive a lower bound for BSP simulations of A_N. The terms $T(N/p)$ and ℓ are obtained immediately. Next, observe that an arbitrary T-step computation of A_N can be modeled as a a dag $\Delta_T(A_N)$ whose nodes (i.e., operations) are all pairs (v, t), with $v \in V$ and $0 \leq t \leq T$, and whose arcs (i.e., state dependencies) connect pairs $(v_1, t), (v_2, t+1)$, where $t < T$ and $|v_1 - v_2| \leq 1$. Simulating a T-step computation of A_N on BSP amounts to *execute* $\Delta_T(A_N)$. During an execution of $\Delta_T(A_N)$, an operation associated to a dag node can be executed by a BSP processor q if and only if q knows the state of all the node's predecessors.

Consider the case $T \leq N$. Then, $\Delta_T(A_N)$ contains the $\lceil T/2 \rceil \times \lceil T/2 \rceil$ *diamond dag* $D_{T/2}$ as a subgraph. The result in [ACS90, Th. 5.1] implies that any BSP execution of $D_{T/2}$ either requires an $\Omega(T)$-relation or an $\Omega\left(T^2\right)$-time sequential computation performed by some processor. Hence, any BSP execution of $D_{T/2}$ requires $\Omega(T \min\{g, T\})$ time. For the case $T \geq N$, it suffices to observe that $\Delta_T(A_N)$ contains a vertical stack of $\Theta(T/N)$ $\lceil N/2 \rceil \times \lceil N/2 \rceil$ disjoint copies of $D_{T/2}$, whose first executions do not overlap in time, thus requiring $\Omega(T \min\{g, N\})$ BSP time.

The next theorem, whose proof is omitted for brevity, provides a lower bound on the reverse simulation.

Theorem 2. *Let G be an N-node network of bisection width B. Then, any simulation of BSP(p, g, ℓ) on G requires slowdown*

$$S\left(\text{BSP}(p, g, \ell), G\right) = \Omega\left(\frac{\min\{N, p\}}{gB}\right) \quad .$$

D-BSP Let us now turn to network simulations on D-BSP. In what follows, we assume w.l.o.g. that G has a decomposition tree $\{G_0^{(i)}, G_1^{(i)}, \cdots, G_{2^i-1}^{(i)} \ : \ \forall i,\ 0 \leq i \leq \log p\}$, where each $G_j^{(i)}$ (*i-subnet*) has $N/2^i$ nodes and is connected to the rest of the network by at most b_i boundary links; moreover, $G_j^{(i)} = G_{2j}^{(i+1)} \cup G_{2j+1}^{(i+1)}$. In order to simulate G on a p-processor D-BSP, we adopt the following strategy. Partition the nodes of G among the D-BSP processors so that the nodes of $G_j^{(i)}$ are assigned to the processors of i-cluster $C_j^{(i)}$, for every i and j. Let $M_{i,j}^{\text{out}}$ (resp., $M_{i,j}^{\text{in}}$) denote the messages that are sent

(resp., received) by nodes of $G_j^{(i)}$ to (resp., from) nodes outside the subnet. Since the number of boundary links of an i-subnet is at most b_i, we have that $|M_{i,j}^{\text{out}}|, |M_{i,j}^{\text{in}}| \leq b_i$. Let also $\bar{M}_{i,j}^{\text{out}} \subseteq M_{i,j}^{\text{out}}$ denote those messages that from $G_j^{(i)}$ go to nodes in its sibling $G_{j'}^{(i)}$, with $j' = j \pm 1$ depending on whether j is even or odd. The idea behind the simulation is to guarantee, for each cluster, that the outgoing messages be balanced among the processors of the cluster before they are sent out, and, similarly, that the incoming messages destined to any pair of sibling clusters be balanced among the processors of the cluster's father before they are acquired.

More precisely, after a first superstep where each D-BSP processor simulates the local computation and communications internal to the subnet assigned to it, the following two cycles are executed.

1. For $i = \log p - 1$ down to 0 do in parallel within each $C_j^{(i)}$, for $0 \leq j < 2^i$:
   (a) Send the messages in $\bar{M}_{i+1,2j}^{\text{out}}$ (resp., $\bar{M}_{i+1,2j+1}^{\text{out}}$) from $C_{2j}^{(i+1)}$ (resp., $C_{2j+1}^{(i+1)}$) to $C_{2j+1}^{(i+1)}$ (resp., $C_{2j}^{(i+1)}$), so that each processor receives (roughly) the same number of messages.
   (b) Balance the messages in $M_{i,j}^{\text{out}}$ among the processors of $C_j^{(i)}$. (This step is vacuous for $i = 0$.)
2. For $i = 1$ to $\log p - 1$ do in parallel within each $C_j^{(i)}$, for $0 \leq j < 2^i$:
   (a) Send the messages in $M_{i,j}^{\text{in}} \cap M_{i+1,2j}^{\text{in}}$ (resp., $M_{i,j}^{\text{in}} \cap M_{i+1,2j+1}^{\text{in}}$) to the processors of $C_{2j}^{(i+1)}$ (resp., $C_{2j+1}^{(i+1)}$), so that each processor receives (roughly) the same number of messages.

A formal proof of correctness of the above procedure will be provided in the full version of this abstract. As for its running time, let $h_i = \lceil b_i/(p/2^i) \rceil$, for $0 \leq i \leq \log p$, denote the average number of incoming/outgoing messages for an i-cluster. The balancing operations performed by the algorithm guarantee that iteration i of either cycle entails a $\max\{h_i, h_{i+1}\}$-relation within i-clusters. Finally, we can optimize the above simulation by running it entirely within within a cluster of $p' \leq p$ processors of the p-processor D-BSP, where p' is the value that minimizes the overall slowdown. The following theorem summarizes the above discussion.

Theorem 3. *For any N-node network G with a decomposition tree of parameters $(b_0, b_1, \ldots, b_{\log p})$, one step of G can be simulated on a D-BSP $(p, \boldsymbol{g}, \boldsymbol{\ell})$ in time*

$$O\left(\min_{p' \leq p}\left\{\frac{N}{p'} + \sum_{i=\log(p/p')}^{\log p-1}(g_i \max\{h_i, h_{i+1}\} + \ell_i)\right\}\right) \quad ,$$

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

Portability of BSP vs D-BSP on multidimensional arrays First, we remark that all the simulation results described above deal with simulation algorithms which preserve ρ-equivalence for most realistic definitions of ρ (e.g., same input-output map, or same high-level algorithm). Let now M be a BSP(p, g, ℓ), and G be a p-node d-dimensional array. Then, from Theorems 1 and 2, it follows that $\delta(M, G) = \Omega\left(p^{1/d}\right)$. Consider now a D-BSP $(p, \boldsymbol{g}, \boldsymbol{\ell})$ with $g_i = \ell_i = (p/2^i)^{1/d}$, for $0 \leq i \leq \log p$. Such D-BSP $(p, \boldsymbol{g}, \boldsymbol{\ell})$

can be simulated on G with constant slowdown. Since G has a decomposition tree with subnets $G_j^{(i)}$ that have $b_i = O\left((p/2^i)^{(d-1)/d}\right)$, the D-BSP simulation presented above yields a slowdown of $O\left(p^{1/(d+1)}\right)$ per step, when entirely run on a cluster of $p' = p^{d/(d+1)}$ processors. In conclusion, letting M' be the D-BSP with the above choice of parameters, we have that $\delta(M', G) = O\left(p^{1/(d+1)}\right)$. Therefore, under the δ metric, D-BSP is asymptotically less distant than BSP from a multidimensional array.

One drawback of the D-BSP simulation is that it requires that the memory size of each D-BSP processor be a $\Theta\left(p^{1/(d+1)}\right)$ factor larger than the memory size of an array node, which might be unreasonable in practice. In the full version of this paper, we will exhibit a subtler simulation strategy of multidimensional arrays on D-BSP which reduces the memory blowup to a mere $O(\log\log p)$ factor, at the expense of an extra $O(\log p)$ factor in the simulation slowdown.

Finally, if we allow the simulation of a step of the network to start before the simulation of previous steps is completed, we can set up a more complex recursive strategy for simulating arrays with constant-size memory at each node on D-BSP, whose slowdown is exponentially smaller than the one obtained by applying the step-by-step simulation of Theorem 3. We have:

Theorem 4. *For $T = \Omega\left(p^{1/d}\right)$, any T-step computation of a p-node d-dimensional array G (d constant) with constant-size memory at each node can be simulated by a D-BSP $(p, \boldsymbol{g}, \boldsymbol{\ell})$ with $g_i = \ell_i = (p/2^i)^{1/d}$, for $0 \leq i \leq \log p$, in time $O\left(T 2^{a\sqrt{\log p}}\right)$, with $a = 2d + 6$.*

(The simulation algorithm and the proof of Theorem 4 are omitted.) Recall that the lower bound on network simulations by BSP (Theorem 1) does not depend on the size of the memory modules local to each node of G and, in particular, it holds even if such modules have constant size. Therefore, Theorem 4 implies that the δ metric for D-BSP and multidimensional arrays with constant-size memory at each node is exponentially smaller than the respective metric for BSP.

5 Exploiting D-BSP Locality for Unbalanced Communication

An (h, m)-relation is a routing instance where each processor sends/receives at most h-messages, and a total of m messages are exchanged [JW96]. On several networks, the time to route an (h, m) relation decreases with m, for fixed h. This does not happen in BSP, where *any* (h, m)-relation requires $gh + \ell$ steps, regardless of m. Motivated by these observation, some authors have proposed the introduction of m as an explicit parameter of the model (see E-BSP [JW96] and Y-BSP [DK96]).

In this section, we argue that D-BSP can route (h, m) relations in ways that take advantage of small values of m. In particular, we show that, when a d-dimensional array simulates the routing of an (h, m)-relation on D-BSP, the resulting network performance is within a constant factor of optimal. Therefore, for such networks, there is no increase of effectiveness by augmenting D-BSP with m-sensitive capabilities. This is an indication that the complication of making a model which already captures network proximity also sensitive to unbalance may not be warranted by the accruing advantages.

The key insight that leads to the exploitation of cluster locality to deal with unbalance is similar to the one employed in Section 4 to simulate one step of a network, and is inspired by the efficient routing protocols known for specific architectures (e.g., see

[PPS94]). Specifically, messages are progressively balanced within clusters of increasing size so that the degree of the relation decreases as the cost of communication within the clusters increases. More specifically, let k be such that $2^k \leq hp/m < 2^{k+1}$ (note that $0 \leq k \leq \log p$). The (h, m)-relation is routed in two phases. In the first phase (*ascend*), messages are iteratively balanced within larger and larger clusters (starting from k-clusters). In the second phase (*descend*) messages are kept balanced and iteratively dispatched to arbitrary processors within their destination k-cluster and from there to their final destinations. We have:

Theorem 5. *An (h, m)-relation can be routed on a D-BSP $(p, \boldsymbol{g}, \boldsymbol{\ell})$ in time*

$$T(h,m) = O\left(\sum_{i=0}^{\lfloor \log(hp/m) \rfloor} \left(T_{\mathrm{pr}}(i) + g_i \left\lceil \frac{m}{(p/2^i)} \right\rceil + \ell_i\right)\right),$$

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

To appreciate the potential of the above strategy, consider again the case of a p-node d-dimensional array, which is characterized by parameters $g_i = \ell_i = p^{1/d}$, for $0 \leq i \leq \log p$. For such values of the parameters, $T_{\mathrm{pr}}(i) = O\left((p/2^i)^{1/d}\right)$ [DK96]. Hence, by Theorem 5 we get

$$T(h,m) = O\left(p^{1/d} + m^{1/d} h^{1-1/d}\right),$$

which is smaller than the BSP time by a factor $(ph/m)^{1/d}$ when $p/h^{d-1} \leq m \leq ph$. More surprisingly, a simple bandwidth argument shows that the above time is optimal for routing an arbitrary (h, m)-relation directly on a d-dimensional array. Hence, for such communication patterns, the effectiveness of D-BSP is maximum, in the sense that there is no loss of efficiency in running the D-BSP algorithm on the network with respect to running the best network-specific algorithm. A corollary of this result is that for multidimensional arrays D-BSP is as effective in treating unbalanced communication as E-BSP [JW96], where the treatment of unbalanced communication is a primitive of the model.

References

[ACS90] A. Aggarwal, A.K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71:3–28, 1990. See also *Proc. of ICALP '88*, 1–17.

[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.

[JW96] 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.

[MMT95] B. Maggs, L.R. Matheson, and R.E. Tarjan. Models of parallel computation: A survey and synthesis. In *Proc. of the 28th Hawaii International Conference on System Sciences (HICSS)*, volume 2, pages 61–70, January 1995.

[PPS94] A. Pietracaprina, G. Pucci, and J. Sibeyn. Constructive deterministic PRAM simulation on a mesh-connected computer. In *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures*, pages 248–256, Cape May, NJ, June 1994.

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