Estimating Application Hierarchical Bandwidth Requirements using BSP Family Models

Adrian Soviani and Jaswinder Pal Singh
Department of Computer Science, Princeton University
{asoviani, jps}@cs.princeton.edu

Abstract

A vast amount of work has been done on developing programming models that provide good performance across machine architectures, are easy to use, and have predictable performance. Similarly, designing and optimizing architectures to achieve the best performance for an application class is a challenging task. Accurate cost modeling is essential to both application development and system design.

Many scientific computing codes are developed using libraries that provide custom-built collective communication primitives. The family of Bulk Synchronous Parallel (BSP) machine models are suitable tools for analyzing such problems. However, modeling the effect of bandwidth limitations for globally unbalanced communication and estimating the hierarchical bandwidth used by applications remain key challenges.

We present a hierarchical bandwidth machine model (αDBSP) that naturally extends the Decomposable BSP (DBSP) model by associating a bandwidth growth factor α to each message pattern. Algorithms executed on αDBSP have a runtime at least as good as DBSP. There are globally unbalanced problems for which αDBSP analysis is simpler or more accurate. E.g., the one element nearest-neighbor message exchange running on a pruned butterfly takes $O(log^3(p))$ on αDBSP and $O(\sqrt{p})$ on DBSP, while optimally modeling the one-to-all broadcast requires a single communication step on αDBSP vs. $O(log(p))$ steps on DBSP. We present three scientific computing kernels that illustrate differences between αDBSP and DBSP analysis.

Similar to BSP family models, αDBSP predicts collective communication execution time for a given machine. Additionally, αDBSP estimates the hierarchical bandwidth required by a given application. System architects may use this estimation to design machines that avoid bandwidth bottlenecks for their target application class.

Keywords: BSP, DBSP, Performance Modeling, Collective Communication, Interconnect Topology

1. Introduction

Building scalable parallel applications remains a difficult task due to implementation complexity, inability to easily understand costs and bottlenecks, and unpredictable communication performance. The machine architecture further influences application efficiency and performance transparency. An accurate and general cost model can become a bridge between application development and machine design; it allows application writers to optimize algorithms and datastructure decompositions, while it helps system designers to build interconnects and tailor bandwidth capacity to application needs.

Cost Model Implementing applications relying on structured collective communication steps is a common trend in the scientific community [1]. This programming paradigm uses custom built data redistribution libraries, making large scale codes easier to write and maintain. At the same time, this programming model allows analyzing application performance using the Bulk Synchronous Parallel (BSP) machine model introduced by Valiant [18], or the Decomposable BSP (DBSP) machine extension that models submachine locality [9].

On the BSP machine a globally synchronized superstep including w work and a h-relation runs in $w + hg + l$ time; a h-relation is a message exchange where each processor sends and receives at most h packets, and l, g are machine parameters corresponding to global latency and bandwidth. Upper bounds on these parameters are computed for theoretical topologies by solving a routing problem using store and forward switches [18]. Experimental results can approximate the same parameters for real machine configurations [6, 14]. The DBSP model extends BSP by executing synchronized supersteps on predefined partitions of processors [3, 8, 9]. This model exploits locality by computing its parameters for each machine size i; running an i-superstep takes $w + hg(i) + l(i)$ time. Similar to BSP, DBSP is an effective but more accurate tool for cost analysis.

While DBSP overestimates the runtime of unbalanced h-relations, results are improved by routing each h-relation as $O(log(p))$ supersteps. A routing algorithm presented in [12] achieves optimal results for (k_1, k_2)-routing problems[1] running on DBSP$(n, \mathbf{g}^{(\alpha)}, \mathbf{l}^{(\beta)})$ machines such as nD meshes. As a corollary (h, m)-relations[2] can be routed in $O(\lceil m/n \rceil^{\alpha} h^{1-\alpha} n^{\alpha} + n^{\beta})$ time on such machines, matching the results of the Extended BSP [15] without adding a new machine primitive [4, 11]. If this bound is optimal when considering all routing problems that are (h, m)-relations, there are still problems for which DBSP overestimates their runtime. E.g., the 2D nearest neighbor one packet exchange executes in constant time on a 2D mesh and $O(log(p))$ time on a pruned butterfly, yet DBSP estimates its runtime to $O(\sqrt{p})$ for both topologies.

This paper presents a hierarchical bandwidth machine model (αDBSP) that naturally extends the DBSP model. A new exponential factor α is augmented to each h-relation describing its growth in terms of required hierarchical bandwidth. On αDBSP a superstep including w work and a (h, α)-relation runs in $w + hg(i, \alpha) + l(i, \alpha)$ time, where $l(i, \alpha)$ and $g(i, \alpha)$ are the new machine parameters. We present two routing algorithms based on (k_1, k_2)-routing [12] that improve the (h, α)-relation execution upper bound compared to DBSP h-relations. The modified algorithms redistribute fewer packets and compute the execution time using an upper bound on the number of packets traveling up to each level in the machine hierarchy.

The αDBSP model is at least as powerful as DBSP; any DBSP algorithm gives identical results on αDBSP when h-relations are replaced with $(h, 0)$-relations. Additionally, there are unbalanced h-relations for which αDBSP gives better results. E.g., for the 2D nearest neighbor exchange αDBSP gives $\Omega(log(p))$ and $O(log^3(p))$ on the pruned butterfly, improving the DBSP (k_1, k_2)-routing upper bound of $O(\sqrt{p})$. Similarly, for the North-South 1D FFT problem (Section 4.2) αDBSP takes $O(log^4(p))$ while DBSP takes $O(\sqrt{p})$. Section 4 presents the cost analysis for three more general computation kernels commonly included by scientific codes.

αDBSP slightly increases analysis effort by adding the α factor to each h-relation in order to improve analysis accuracy. This tradeoff can lead to a faster runtime as mentioned earlier, or to a simpler cost analysis. For the generalized broadcast problem αDBSP takes $O(np)$ while DBSP takes $O(np\sqrt{p})$ on the pruned butterfly; the (k_1, k_2)-routing DBSP algorithm runs this problem as fast as αDBSP, but it executes $O(log(p))$ supersteps making analysis more complex on arbitrary machines. For the examples from Section 4 αDBSP balances well cost

[1] a (k_1, k_2)-routing is a routing instance where each processor sends at most k_1 and receives at most k_2 packets

[2] a (h, m)-relation is a routing instance where each processor sends and receives at most h packets, and the total number of packets is m

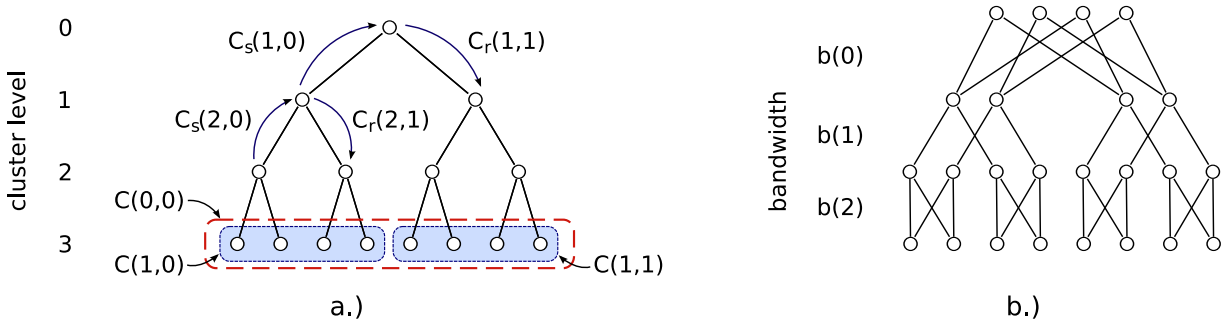

Figure 1: a.) DBSP cluster hierarchy for a 16 processor fat-tree, and top level $C_{s/r}(i,j)$ packet sets; b.) Pruned butterfly implementing a fat-tree with link bandwidths $b(i) = 2^{\lceil k-i \rceil \alpha_M}$ for $\alpha_M = 1/2$.

accuracy and analysis effort, improving DBSP results without making analysis any harder.

System Design Architectural advances over the years have led to a relatively standardized modern parallel computer design. High bandwidth and easy to build, fat-tree and mesh interconnects are commonly used with Myrinet, Infiniband, NumaLink, and SeaStar switches [16]. If the advent of wormhole routing has made in-transit time insignificant compared to end-point overhead, bandwidth capacity planning remains an important issue [17].

Unfortunately, bandwidth availability can vary dramatically at various levels of the memory-network hierarchy [11]. E.g., inter-core bandwidth is higher than intra-socket bandwidth, which is higher than the networking chip bandwidth Moreover, large scale machines cannot provide full bisection width bandwidth[3] at the top of their router hierarchy[4]. This unbalance becomes more prevalent as more cores are packed into a box, and system designers are forced to make a tradeoff between the number of cores, per-core bandwidth, and switch size. When thousand-core chips become available we expect the bandwidth gap to grow even larger.

It is possible to build parallel machines that avoid inherent bandwidth bottlenecks for a class of applications without relying on full bisection width bandwidth. An αDBSP application exposes communication as (h,α)-relations, and its α bandwidth growth factor allows computing a lower bound on required hierarchical bandwidth and link capacity. E.g., a 2D PDE solver that employs FFT filtering at the top and bottom of its grid has $\alpha = 1/2$; a large scale machine organized as a fat-tree with $\alpha_M = 1/2$ or a 2D mesh can execute this problem without inherent bandwidth bottlenecks (Section 4).

The paper is organized as follows. Section 2 gives a more precise definition of DBSP and introduces the αDBSP model. In Section 3 we present two routing algorithms which are later used to prove upper bounds on executing (h,α)-relations on various topologies. Section 4 analyzes three computation kernels chosen to illustrate differences between αDBSP and DBSP in terms of time complexity and analysis effort. Section 5 summarizes related work, and Section 6 presents the concluding remarks.

2. Definitions

Applications that rely on structured collective communication and computation steps may be analyzed using the Bulk Synchronous Parallel (BSP) family of machine models. The Decomposable BSP (DBSP) model is an extension of BSP that exploits locality by relying on submachine decomposition [8]. We will consider only recursive DBSP machines that allow recursive binary machine decompositions (Fig. 1) [9].

Let $p = 2^k$ be the total number of processors of a DBSP machine. For each $0 \le i \le k$ the machine is partitioned into 2^i disjoint i-clusters $C(i,j)$ containing processors $j2^{k-i}$ to $(j+1)2^{k-i}-1$. Note that each $C(i,j)$ cluster has 2^{k-i} processors, and clusters form a binary decomposition hierarchy i.e. $C(i,j) = C(i+1,2j) \cup C(i+1,2j+1)$.

A DBSP machine is parametrized by two vectors $g(i)$ and $l(i)$, where $0 \le i \le k$. These parameters describe the bandwidth and latency characteristics of submachines i.e. a h-relation is executed on a $C(i,j)$ cluster in

[3] minimum aggregate bandwidth linking two equal partitions of the interconnect

[4] the LRZ SGI Altix 4700 machine has nodes organized as a mesh of fat-trees; the RedStorm Cray XT4 system is a 3D mesh

$hg(i) + l(i)$ time; a h-relation is a routing problem where each processor sends and receives at most h packets.

A DBSP program is executed as a sequence of labeled supersteps. An i-superstep is executed in parallel on $C(i, j)$ clusters and takes $w + hg(i) + l(i)$ time, where w is the maximum number of operations or work performed by each processor during this superstep [4].

These definitions of p, k, $g(i)$, $l(i)$, and $C(i, u)$ will be used throughout the rest of the paper.

Definition 2.1 *We call* $\mathbf{R}$*-routing a routing problem where processor* x *sends* $R(x, y)$ *packets to processor* y*.*

Definition 2.2 *For a* $\mathbf{R}$*-routing problem we define* $C_s(i, u)$ *to be the set of packets sent by the processors of cluster* $C(i, u)$ *to all other processors, and* $C_r(i, u)$ *to be the set of packets received by the processors of cluster* $C(i, u)$ *from all other processors. We have:*

$$|C_s(i,u)| = \sum_{x \in C(i,u),\ y \notin C(i,u)} R(x,y), \quad |C_r(i,u)| = \sum_{x \notin C(i,u),\ y \in C(i,u)} R(x,y)$$

A DBSP h-relation can be generalized by computing a value $H(i)$ corresponding to each level i of the machine hierarchy. $H(i)$ represents the maximum number of packets sent to or from any i-cluster divided by the number of processors in that cluster. The vector $\mathbf{H}$ will be used as a tool for cost analysis rather than as a machine primitive.

Definition 2.3 *A* $\mathbf{R}$*-routing problem is a* $\mathbf{H}$*-relation if*

$$H(i) = \frac{1}{2^{k-i-1}}\ max_{0 \le u < 2^{i+1}}\ \{|C_s(i+1,u)|,\ |C_r(i+1,u)|\} \quad \textit{for}\ 0 \le i < k$$

We can define an exponential factor α to capture the growth of $\mathbf{H}$. The $log(p)$ vector values can be compressed to a single (h, α) pair that enforces an upper bound on all $H(i)$.

Definition 2.4 *A* $\mathbf{H}$*-relation executed during an* i*-superstep is a* (h, α)*-relation if*

$$H(j) \le \frac{1}{2^{(k-j-1)\alpha}}\ h \quad \textit{for all}\ i \le j < k$$

Note that for $\alpha = 0$ we have $H(i) = H(k-1)$ meaning all packets are routed to the top of the hierarchy, while for $\alpha = 1$ we have $H(i-1) = H(i)/2$ meaning half of the packets reaching level i will be routed one level up.

The α factor gives an upper bound on the number of packets sent to each level of the hierarchy, and it helps computing an execution upper bound. E.g., it allows determining the λ load factor [16] and its corresponding upper bound for a fat-tree with known bandwidths [2]. For some patterns α gives a lower bound on packet count, too. When this is the case, a lower bound on execution time can be proved making a bandwidth argument.

Definition 2.5 *An algorithm executing an* i*-superstep* $\mathbf{H}$*-relation is said to execute a tight* (h, α)*-relation if* $\exists c > 0$ *s.t. for any problem size and machine size we have*

$$c\ \frac{1}{2^{(k-j-1)\alpha}}\ h \le H(j) \le \frac{1}{2^{(k-j-1)\alpha}}\ h \quad \textit{for all}\ i \le j < k$$

We define the αDBSP machine as an extension of the DBSP machine that executes (h, α)-relations during each i-superstep. The bandwidth $g(i, \alpha)$ and latency $l(i, \alpha)$ parameters are defined function of both machine size i, and bandwidth growth factor α.

Definition 2.6 *An* α*DBSP machine executes an* i*-superstep* (h, α)*-relation and* w *work in time*

$$T_{\alpha BSP}(i, w, h, \alpha) = w + hg(i, \alpha) + l(i, \alpha)$$

where $g(i, \alpha)$*,* $l(i, \alpha)$ *are the* α*DBSP machine parameters.*

DBSP machines and algorithms are particular cases of αDBSP for $\alpha = 0$. Any DBSP machine is an αDBSP machine with $g(i, \alpha) = g(i)$ and $l(i, \alpha) = l(i)$ since any (h, α)-relation is a h-relation; any DBSP algorithm becomes an αDBSP algorithm when h-relations are replaced with $(h, 0)$-relations.

The next section analyzes communication cost and presents the αDBSP machine parameters for most common topologies.

Table 1: Bounds on i-superstep (h,α)-relation execution time and αBSP parameters for common machine topologies with p processors. The relation is executed on a partition of 2^i submachines with $p' = p/2^i$ processors each; the lower bound applies only to tight relations.

Machine	α_M	(h,α)-relation	DBSP	αDBSP		$g(i,\alpha)$	$l(i,\alpha)$
Butterfly	0	$\alpha = 0$	$O(h\,log(p'))$	$O(h\,log(p'))$	$\Omega(h + log(p'))$	$O(log(p'))$	$O(log(p'))$
		$\alpha > 0$		$O(h + log^3(p'))$	$\Omega(h + log(p'))$	$O(1)$	$O(log^3(p'))$
Pruned nD Butterfly	$1/2$	$\alpha < \alpha_M$	$O(h\,p'^{\alpha_M})$	$O(h\,p'^{\alpha_M-\alpha})$	$\Omega(h\,p'^{\alpha_M-\alpha})$	$O(p'^{\alpha_M-\alpha})$	$O(log(p'))$
	$1/n$	$\alpha = \alpha_M$		$O(h\,log(p') + log^3(p'))$	$\Omega(h + log(p'))$	$O(log(p'))$	$O(log^3(p'))$
		$\alpha > \alpha_M$		$O(h + log^3(p'))$	$\Omega(h + log(p'))$	$O(1)$	$O(log^3(p'))$
nD Mesh	$1/n$	$\alpha < \alpha_M$	$O(h\,p'^{\alpha_M})$	$O(h\,p'^{\alpha_M-\alpha} + p'^{\alpha_M})$	$\Omega(h\,p'^{\alpha_M-\alpha})$	$O(p'^{\alpha_M-\alpha})$	$O(p'^{\alpha_M})$
		$\alpha = \alpha_M$		$O(h\,log(p') + p'^{\alpha_M})$	$\Omega(h)$	$O(log(p'))$	$O(p'^{\alpha_M})$
		$\alpha > \alpha_M$		$O(h + p'^{\alpha_M})$	$\Omega(h)$	$O(1)$	$O(p'^{\alpha_M})$

3. Bounds

An upper-bound on (h,α)-relation execution time can be computed by routing the relation as a sequence of redistribution steps. This method is similar to the algorithm for (k_1,k_2)-routing [10] that redistributes all packets evenly among all processors during a bottom up phase, then routes each packet to its destination during a top down phase.

The (h,α)-relation routing algorithm improves the (k_1,k_2)-routing by redistributing only the packets with destinations outside a cluster during the bottom up phase i.e. $C_s(i,j)$ for machines of size i. The number of packets redistributed at each level is upper-bounded by $\frac{h}{2^{(k-i)\alpha}}$ (Definition 2.4) giving a bandwidth cost of $O\left(\lceil H(i)\rceil\, g(i)\right)$.

Theorem 3.1 *An s-superstep* $\mathbf{H}$*-relation is executed by a DBSP machine in time*

$$O\left(\sum_{i=s}^{k-1}\lceil H(i)\rceil g(i) + \sum_{i=s}^{k-1}\left(l(i) + Pf(i)\right)\right)$$

where $g(i)$, $l(i)$ are the DBSP machine parameters; $Pf(i)$ is the parallel prefix execution time on an i-level machine.

Proof. Let $C_x(i,j)$ be the set of packets sent from $C(i+1,2j)$ processors to $C(i+1,2j+1)$ processors, and from $C(i+1,2j+1)$ processors to $C(i+1,2j)$ processors. $C_x(i,j)$ are packets exchanged between the children of cluster $C(i,j)$ and travel up to level i in the hierarchy. We have $C_s(i+1,2j) \cup C_s(i+1,2j+1) = C_x(i,j) \cup C_s(i,j)$ and $C_r(i+1,2j) \cup C_r(i+1,2j+1) = C_x(i,j) \cup C_r(i,j)$.

We say that n packets are evenly distributed among 2^a processors if each processor holds at most $\lceil \frac{n}{2^a} \rceil$ packets. Let's consider the following algorithm that executes a $\mathbf{H}$-relation:

1. For each $i = k-1$ downto s execute step 1.a

1.a For each processor cluster $C(i,j)$ with $0 \le j < 2^i$ execute step 1.b

Assert $C_s(i+1,x)$ and $C_x(i+1,x)$ have packets evenly distributed among $C(i+1,x)$ processors for all x.

1.b Redistribute evenly the packets from $C_s(i,j)$ among the processors of $C(i,j)$. Repeat the process for the packets from $C_x(i,j)$.

2. For each $i = s$ to $k-1$ execute step 2.a

2.a For each processor cluster $C(i,j)$ with $0 \le j < 2^i$ execute step 2.b

Assert $C_r(i,x)$ have packets evenly distributed among $C(i,x)$ for all x.

2.b. Redistribute evenly the packets from $C_r(i+1,2j)$ among the processors of $C(i+1,2j)$, and the packets from $C_r(i+1,2j+1)$ among the processors of $C(i+1,2j+1)$

The assertions after step 1.a and 1.b can be easily checked by induction. During the bottom up phase the packets from the $C_x(i,j)$ sets are evenly distributed among $C(i,j)$ processors, level by level. Step 1.b can be executed as a parallel prefix and two $\lceil H(i) \rceil$-relations. Note that for $H(i) < 1$ execution time is still $g(i)$ even when $H(i)g(i)$ could be significantly smaller.

Similarly, during the top down phase the packets from the $C_r(i,j)$ sets are distributed evenly, and step 2.b is executed as a parallel prefix followed by two $\lceil H(i) \rceil$-relations. The theorem follows after adding up the execution time for steps 1.b and 2.b. □

The results from Theorem 3.1 can be improved on the pruned butterfly if we observe that redistributions of $n < p$ packets run faster than a 1-relation. Such redistributions executed as $\left\lceil \frac{n}{\sqrt{p}} \right\rceil$ waves [2] take $O\left(\left\lceil \frac{n}{\sqrt{p}} \right\rceil + log(p)\right)$ time, improving the $O\left(\sqrt{p}\right)$ bound for a 1-relation.

Theorem 3.2 *An s-superstep* **H***-relation is executed by a pruned butterfly in time*

$$O\left(\sum_{i=s}^{k-1} \lceil H(i)g(i) \rceil + \sum_{i=s}^{k-1} (l(i) + Pf(i))\right)$$

where $l(i) = k-i$, $g(i) = 2^{(k-i)\alpha_M}$, $\alpha_M = 1/2$, and $Pf(i)$ is the parallel prefix execution time on an i-level machine.

Proof. The algorithm presented by Theorem 3.1 will be modified to accommodate this result. We say n packets are evenly distributed among 2^a processors if: i) for $n \geq 2^a$ each processor holds at most $\lceil \frac{n}{2^a} \rceil$ packets; ii) for $n < 2^a$ processors $x = 0 \, (mod \ 2^{a-m})$ hold at most one packet and all other processors hold no packets, where $2^{m-1} < n \leq 2^m$.

Steps 1.b and 2.b are executed as $H(i)$-relations for $H(i) \geq 1$, and $\lceil H(i)/g(i) \rceil$ waves for $H(i) < 1$ [2].

Let's consider redistributing $C_s(i,j)$ during step 1.b for $H(i) < 1$. The $C_s(i,j)$ packets are evenly distributed i) first by computing a parallel prefix for the placement of $C_s(i+1,2j) \cup C_s(i+1,2j+1)$ also in $C_s(i,j)$; ii) then by routing these packets as waves to their $C(i,j)$ destinations.

Let's consider routing packets $C_s(i+1,2j) \cap C_s(i,j)$ to processors $C(i,j)$. Cluster $C(i+1,2j)$ has 2^{k-i-1} processors; let $s = k-i-1$, and let's choose $2^{m-1} < H(i)2^s \leq 2^m$. Processors $x = 0 \, (mod \ 2^{s-m})$ from $C(i+1,2j)$ will hold at most one packet since $C_s(i+1,2j)$ has at most $H(i)2^s \leq 2^m$ packets distributed among 2^s processors. Similarly, processors $x = 0 \, (mod \ 2^{s-m})$ from $C(i,j)$ will hold at most one packet. When $m \geq \lceil (k-i)/2 \rceil$ the routing is executed as $2^{m-\lceil (k-i)/2 \rceil}$ waves, each wave j involving processors $x = j \, (mod \ 2^{s-m-\lceil (k-i)/2 \rceil})$; otherwise it is executed as a single wave. This gives a routing time of $O(\lceil 2^{m-\lceil (k-i)/2 \rceil} \rceil + (k-i)) = O(\lceil H(i)g(i) \rceil + l(i))$.

The $C_s(i+1,2j+1) \cap C_s(i,j)$ packets are routed to $C(i,j)$ using the same algorithm. Overall, redistributing $C_s(i,j)$ during step 1.b takes $O(\lceil H(i)g(i) \rceil + l(i) + Pf(i))$ time. The same argument can be repeated for redistributing $C_x(i,j)$ during step 1.b, and redistributing $C_r(i+1,2j)$, $C_r(i+1,2j+1)$ during step 2.b. The theorem follows after adding up these terms. □

We can compute an upper bound on i-superstep (h,α)-relation execution time by plugging in the machine parameters into Theorem 3.1 for the butterfly and mesh topologies, and into Theorem 3.2 for the pruned butterfly topologies. Lower bounds can be deduced by making a bandwidth argument about the links connecting $i+1$ level machines and enforcing the minimum packet travel time within i level machines. We will briefly calculate the results shown in Table 1.

On a butterfly, for $\alpha = 0$ the h-relation BSP routing algorithm gives an execution time of $O(h(k-i))$. For $\alpha > 0$ Theorem 3.1 gives an execution time of $O\left(\sum_{j=i}^{k-1} \left(\frac{1}{2^{(k-j)\alpha}} h + 1\right)(k-j) + \sum_{j=i}^{k-1} (k-j)^2\right) = O\left(h + (k-i)^3\right)$.

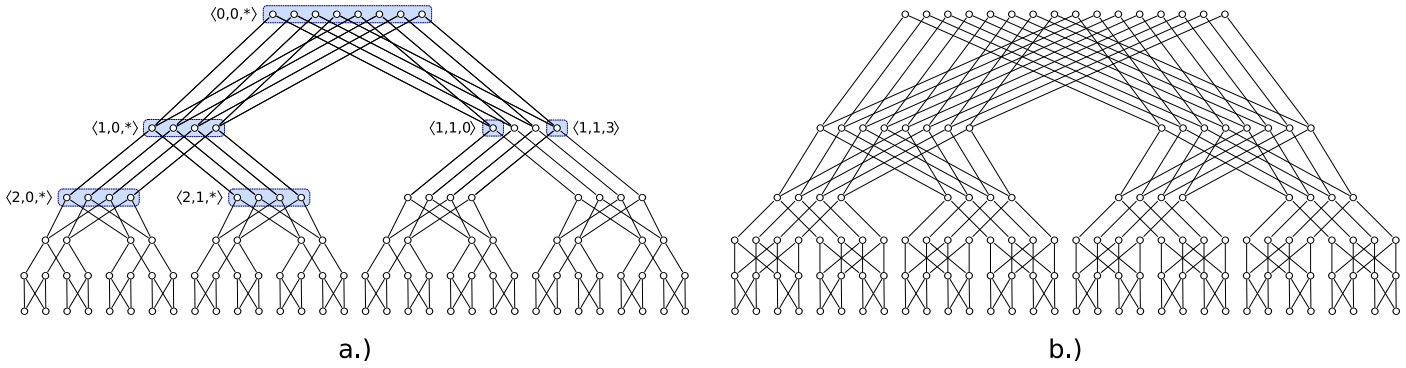

Figure 2: a.) Pruned Butterfly or 2D Butterfly for $p = 64$; b.) 3D Butterfly for $p = 64$

On a nD mesh with $\alpha_M = 1/n$ the execution time is $O\left(\sum_{j=i}^{k-1}\left(\frac{1}{2^{(k-j)\alpha}}h+1\right)2^{(k-j)\alpha_M}+\sum_{j=i}^{k-1}2^{(k-j)\alpha_M}\right) = O\left(\sum_{j=i}^{k-1}\frac{1}{2^{(k-j)\alpha}}\,h\,2^{(k-j)\alpha_M}+2^{(k-i)\alpha_M}\right) = O\left(h\sum_{j=i}^{k-1}2^{(k-j)(\alpha_M-\alpha)}+2^{(k-i)\alpha_M}\right)$. This expression evaluates to $O\left(h\,2^{(k-i)(\alpha_M-\alpha)}+2^{(k-i)\alpha_M}\right)$ for $\alpha < \alpha_M$; $O\left(h(k-i)+2^{(k-i)\alpha_M}\right)$ for $\alpha = \alpha_M$; and $O\left(h+2^{(k-i)\alpha_M}\right)$ for $\alpha > \alpha_M$.

On a pruned butterfly with $\alpha_M = 1/2$ the execution time is $O\left(\sum_{j=i}^{k-1}\left(\frac{1}{2^{(k-j)\alpha}}\,h\,2^{(k-j)\alpha_M}+1\right)+(k-i)^3\right) = O\left(h\sum_{j=i}^{k-1}2^{(k-j)(\alpha_M-\alpha)}+(k-i)^3\right)$. This evaluates to $O\left(h2^{(k-i)(\alpha_M-\alpha)}\right)$ for $\alpha < \alpha_M$; $O\left(h(k-i)+(k-i)^3\right)$ for $\alpha = \alpha_M$; and $O\left(h+(k-i)^3\right)$ for $\alpha > \alpha_M$.

The pruned butterfly topology presented in [2] can be generalized by excluding one set of nodes from a regular butterfly every n levels. The resulting topology has a bisection width of $p^{(n-1)/n}$. While both an nD mesh and an nD butterfly have the same bisection width, the butterfly has $log(p)$ maximum latency vs. $\sqrt[n]{p}$ for the mesh. Note that for $n=1$ this topology is a binary tree, for $n=2$ it is a pruned butterfly, and for $n=\infty$ it is a butterfly.

Definition 3.3 *We call an nD butterfly a graph $G(V,E)$ that is a subgraph of a butterfly with $p = 2^k$ processors. More precisely*

$$\begin{aligned} V &= \left\{\langle i,j,u\rangle \mid 0 \le i \le k,\ 0 \le j < 2^i,\ 0 \le u < 2^m\right\} \\ E &= \left\{\left(\langle i,j,u\rangle, \langle i+1, 2j, u'\rangle\right),\ \left(\langle i,j,u\rangle, \langle i+1, 2j+1, u'\rangle\right) \mid \right. \\ &\qquad \left. 0 \le i < k,\ 0 \le j < 2^i,\ 0 \le u < 2^m\right\} \end{aligned}$$

where $m = \left\lceil (k-i)\frac{n-1}{n}\right\rceil$, and $u' = u$ for $k-i = 0\,(mod\ n)$; $u' = u - 2^{m-1}\left\lfloor \frac{u}{2^{m-1}}\right\rfloor$ for $k-i \neq 0\,(mod\ n)$

An upper bound on executing an i-superstep (h,α)-relation on the nD butterfly can be computed relying on a generalization of Theorem 3.2 where $\alpha_M = 1/n$; the results from Table 1 follow as corollaries. Proving the generalization of Theorem 3.2 relies on the ability of nD butterflies to route m waves of $2^{\lceil k(n-1)/n\rceil}$ evenly distributed packets in $O(m + log(p))$ time. This property can be proved by building a routing algorithm similar to the pruned butterfly wave routing algorithm presented in [2].

4. Examples

This section compares the αDBSP model presented in this paper with DBSP using three simple kernels common in scientific computing applications. Message patterns are executed on DBSP using two routing algorithms: i) DBSP : a h-relation is executed as a single i-superstep; ii) DBSP+ : a (k_1, k_2) K_{12}-routing problem is executed as $2(log(p) - i)$ supersteps as described in [4, 10]; a (h, m) hm-routing problem is executed as a $(h, \lceil \frac{m}{p} \rceil)$ K_{12}-routing followed by another $(\lceil \frac{m}{p} \rceil, h)$ K_{12}-routing. To simplify analysis we use the following notation to represent K_{12}-routing time:

$$K_{12}route(k_1, k_2, i) = O\left(\sum_{j=0}^{log(p)-1-i} \left((min\{k_1, k_2 2^j\} + min\{k_1 2^j, k_2\})g(j+i) + l(j+i) + Pf(j+i)\right)\right)$$

where $Pf(i)$ is the time taken to execute a parallel prefix on an i-level machine.

Our examples are chosen to illustrate the difference between the BSP family models in terms of time complexity and analysis effort (Table 2). We choose to evaluate these results on the pruned butterfly since our focus is estimating the required hierarchical bandwidth; on full bisection width topologies the results are straight-forward, while on meshes the bandwidth cost may be dominated by the latency cost.

4.1. Generalized Broadcast

The generalized broadcast algorithm sends n data elements from one processor to all other processors in a single step. This problem is an example of unbalanced communication for which αDBSP gives a result similar to K_{12} routing on DBSP+ and better than DBSP. The routing problem is a np-relation for DBSP, a tight $(np, 1)$-relation for αDBSP, and a K_{12} (np, n)-routing for DBSP+. For αDBSP we have:

$$T_{\alpha BSP}(n) = np\, g(0,1) + l(0,1)$$

and for DBSP we have:

$$\begin{aligned} T_{DBSP}(n) &= np\, g(0) + l(0) \\ T_{DBSP+}(n) &= K_{12}route(np, n, 0) = O\left(\sum_{i=0}^{log(p)-1} \left(n(2^i + 1)g(i) + l(i) + Pf(i)\right)\right) \end{aligned}$$

On the pruned butterfly we have $g(i) = \sqrt{p}/2^{i/2}$, $l(i) = log(p) - i$, $Pf(i) = (log(p) - i)^2$, and $g(i, 1) = 1$, $l(i, 1) = (log(p) - i)^2$ (Table 1). By plugging in these parameters we evaluate the execution time for the three machine models:

$$\begin{aligned} T_{\alpha BSP}(n) &= O(np + log^2(p)) = O(np) \\ T_{DBSP}(n) &= O(np\sqrt{p} + log(p)) = O(np\sqrt{p}) \\ T_{DBSP+}(n) &= O\left(\sum_{i=0}^{log(p)-1} \left(n2^i\sqrt{p}\, 2^{-i/2} + (log(p) - i)^2\right)\right) \\ &= O\left(n\sqrt{p} \sum_{i=0}^{log(p)-1} 2^{i/2} + log^3(p)\right) = O(np + log^3(p)) = O(np) \end{aligned}$$

As we hoped, the αDBSP model presented in this paper gives a cost analysis that is both simple and accurate for the generalized broadcast problem.

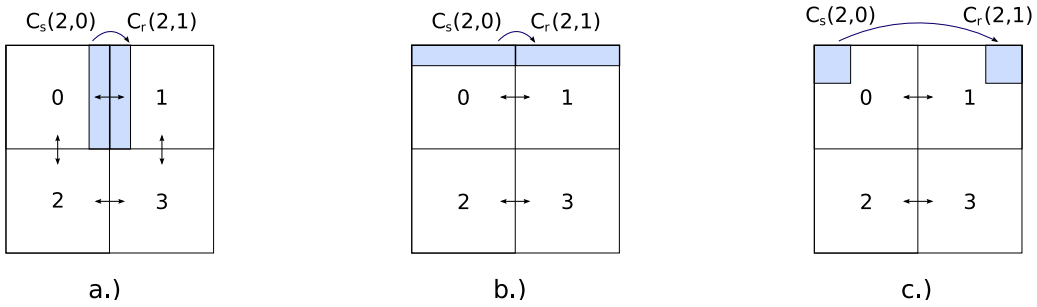

Figure 3: Message patterns corresponding to a $(n, \frac{\sqrt{p}}{2}n)$ hm-routing: a.) nearest neighbor exchange between level 2 clusters; b.) North-South FFT for $s = log(\sqrt{p}) - 1$; c.) $\frac{\sqrt[4]{p}}{\sqrt{2}} \times \frac{\sqrt[4]{p}}{\sqrt{2}}$ corner area exchange.

4.2. North-South FFT

Many scientific codes discretize the space using grids that have singularity points. Problems such as atmospheric and ocean simulations use bipolar or tripolar grids that are mapped to the surface of the Earth. Most numerical methods become unstable in the neighborhood of grid poles and use FFTs to filter undesired frequencies.

The North-South FFT problem computes an $n\sqrt{p}$ element FFT on the top and bottom rows of a 2D matrix. We are summarizing the algorithm for the top row FFT in terms of computation and communication steps:

1. At the beginning each top row processor is assigned n data elements
2. Each processor computes a local FFT that takes $n\,log(n)$ computation time
3. For each $s = 0..log(\sqrt{p}) - 1$ execute steps 4. and 5.
4. Processor x gets n data elements from processor y, where x, y are top row processors and their binary representation is identical except for bit $2s$
5. Each top row processor recomputes its local n data elements taking n computation time
6. At the end the n data elements assigned to the top row processors contain the Fourier transform

The 2D processor grid can be recursively divided vertically, then horizontally to create a binary tree that is mapped to the hierarchical DBSP machine. Each communication step 4. is executed up to level $i = log(p)-1-2s$. For every $0 \leq s' < s$ and intermediate level $i' = log(p) - 1 - 2s'$ there are $2^{s'}$ packets sent from a size $2^{2s'}$ submachine to the higher levels. Step 4. has a growth factor of $\alpha = 1/2$ and represents a n-relation for DBSP, a $(n, 1/2)$-relation for αDBSP, and a $(n, n\sqrt{p})$ hm-routing for DBSP+. For αDBSP we have:

$$T_{\alpha BSP}(n) = n\,log(n\sqrt{p}) + \sum_{s=0}^{log(p)/2-1} (ng(2s{+}1, 1/2) + l(2s{+}1, 1/2))$$

and for DBSP we have:

$$T_{DBSP}(n) = n\,log(n\sqrt{p}) + \sum_{s=0}^{log(p)/2-1} (ng(2s{+}1) + l(2s{+}1))$$

$$T_{DBSP+}(n) = n\,log(n\sqrt{p}) + \sum_{s=0}^{log(p)/2-1} 2\,K_{12} route\left(n, \left\lceil \frac{2^s n}{2^{2s}} \right\rceil, log(p){-}1{-}2s\right)$$

$$= O\left(n\,log(n\sqrt{p}) + \sum_{s=0}^{log(p)/2-1} \sum_{j=0,\, i=log(p)-1-2s+j}^{2s} \left(min\{n, \left\lceil \frac{n}{2^s} \right\rceil 2^j\} g(i) + l(i) + Pf(i)\right)\right)$$

On the pruned butterfly we have $g(i) = \sqrt{p}/2^{i/2}$, $l(i) = log(p)–i$, $Pf(i) = (log(p)–i)^2$, and $g(i, 1/2) = log(p)–i$, $l(i, 1/2) = (log(p)-i)^2$ (Table 1). We can evaluate:

$$\begin{aligned} T_{\alpha BSP}(n) &= O\left(n\,log(np) + n\sum_{i=0}^{log(p)/2}(2i + (2i)^3)\right) = O\left(n\,log(n) + n\,log^2(p) + log^4(p)\right) \\ T_{DBSP}(n) &= O\left(n\,log(np) + n\sum_{i=0}^{log(p)/2}(2^i + 2i)\right) = O\left(n\,log(n) + n\sqrt{p}\right) \end{aligned}$$

To compute $T_{DBSP+}(n)$ we can make an argument similar to [10]. We observe that the inner sum is dominated by its largest term that is reached when $n/2^{j/2} = \left\lceil \frac{n}{2^s} \right\rceil 2^{j/2}$. We get:

$$\begin{aligned} T_{DBSP+}(n) &= O\left(n\,log(np) + \sum_{s=0}^{log(p)/2}\left(\left\lceil \frac{n}{2^s} \right\rceil^{1/2} n^{1/2}2^s + s^3\right)\right) \\ &= O\left(n\,log(np) + \sum_{s=0}^{log(p)/2}\left(n2^{s/2} + n^{1/2}2^s\right) + log^4(p)\right) = O\left(n\,log(n) + n\sqrt[4]{p} + n^{1/2}\sqrt{p}\right) \end{aligned}$$

The αDBSP analysis leads to a generic parametrized expression without too much effort, while plugging in the machine parameters is straightforward. The K_{12} routing algorithm used by DBSP+ improves the single superstep DBSP runtime while αDBSP improves the DBSP+ bounds: for $n = 1$ αDBSP gives $O(log^4(p))$ vs. $O(\sqrt{p})$ on DBSP; for $n = \sqrt{p}$ αDBSP gives $O(n\,log^2(p))$ vs. $O(n\sqrt[4]{p})$ on DBSP (Table 2).

4.3. Nearest Neighbor Exchange

Scientific codes commonly use differencing schemes to solve partial differential equations. To approximate the local derivatives during each iteration, these solvers need data computed by the neighbor processors during the previous iteration. The nearest neighbor problem sends n data elements from each processor to its direct neighbors considering that processors are mapped to a 2D grid.

The 2D processor grid is mapped to a binary tree by recursive horizontal and vertical division as in Section 4.2. For a vertical split, a level $2s$ subtree sends up $2^s n$ data; for a horizontal split a level $2s + 1$ subtree sends up $2^s n$ data. This problem is a n-relation for DBSP, a tight $(n, 1/2)$-relation for αDBSP, and a sum of two $(n, \left\lceil \frac{2^s n}{2^{2s}} \right\rceil)$ K_{12}-routings for DBSP+. For αDBSP we have:

$$T_{\alpha BSP}(n) = ng(0, 1/2) + l(0, 1/2)$$

and for DBSP we have:

$$\begin{aligned} T_{DBSP}(n) &= ng(0) + l(0) \\ T_{DBSP+}(n) &= O\left(\sum_{s=0}^{log(p)/2} K_{12}route\left(n, \left\lceil \frac{n}{2^s} \right\rceil, log(p)-1-s\right)\right) \end{aligned}$$

On the pruned butterfly the computations are similar to the analysis from Section 4.2. By substitution we obtain $T_{\alpha BSP}(n) = O(nlog(p) + log^3(p))$ and $T_{DBSP}(n) = O(n\sqrt{p})$. Evaluating the $K_{12}route$ term for DBSP+ we get $T_{DBSP+}(n) = O(n\sqrt[4]{p} + n^{1/2}\sqrt{p})$.

The nearest neighbor and the North-South FFT examples represent a simplification of the steps executed by scientific codes running on 2D grids mapped to the surface of the Earth. By adding up the numbers we compute a

Table 2: Bounds on algorithm execution time for three BSP family machines modeling a pruned butterfly topology. The examples are: a.) Generalized Broadcast; b.) North-South FFT; c.) Nearest Neighbor Exchange.

	DBSP	DBSP+	αDBSP	
a.)	$O(np\sqrt{p})$	$O(np)$	$\Omega(np)$	$O(np)$
b.)	$O(n\,log(n) + n\sqrt{p})$	$O(n\,log(n) + n\sqrt[4]{p} + n^{1/2}\sqrt{p})$	$\Omega(n\,log(np) + log^2(p))$	$O(n\,log(n) + nlog^2(p) + log^4(p))$
c.)	$O(n\sqrt{p})$	$O(n\sqrt[4]{p} + n^{1/2}\sqrt{p})$	$\Omega(n + log(p))$	$O(n\,log(p) + log^3(p))$

communication cost of $O\left(\sum_{i=0}^{log(p)-1}(ng(i,1/2) + l(i,1/2))\right)$ and a required bandwidth growth factor of $\alpha = 1/2$ for the entire application.

The DBSP+ results from Section 4.2 and Section 4.3 use the hm and K_{12} routing algorithms for DBSP presented in [4, 10]. The K_{12} algorithm is not optimal for all problems classified as $\left(n, \left\lceil \frac{n}{\sqrt{p}} \right\rceil\right)$ K_{12}-routings. E.g., if the examples shown in Fig. 3b and Fig. 3c have identical K_{12}-routings, the first example can be executed faster. For the example from Fig. 3b the DBSP algorithm introduced by Theorem 3.1 improves the K_{12} results being as good as αDBSP for $n \geq \sqrt{p}$; for $n < \sqrt{p}$ this is no longer the case. In particular, for $n = 1$ no DBSP algorithm can give a bound better than $O(\sqrt{p})$ since at least one 0-superstep 1-relation is executed. This observation and the results from Theorem 3.2 explain why αDBSP can improve the highly optimized DBSP+ results.

Table 2 summarizes the execution bounds for the three examples analyzed in this section; the αDBSP results include lower bounds since all relevant (h, α)-relations are tight. For these examples αBSP is an improvement of DBSP+ in terms of execution complexity, and observing the detailed analysis we can argue using αBSP is no harder than using DBSP.

5. Related Work

The PRAM model provides a framework for algorithm analysis extending the RAM concept to parallel architectures by assuming constant latency remote memory access. The technical infeasibility of this model has been addressed by PRAM variants that try to adapt the cost model to real machine performance [13].

First in a family of bridging models, Valiant's BSP [18] describes a machine executing synchronized supersteps consisting of local computation, global communication, and synchronization. The BSP machine executes a superstep with w work and a h-relation in $w + gh + l$ time, where the latency and bandwidth machine parameters l, g are computed globally without modelling locality.

The LogP model proposed by Culler et al. relaxes the synchronization requirement and models more precisely the interconnect performance [7]. The fine grain cost model accounts for local overhead o for sending and receiving packets, local bandwidth limitation G and global interconnect capacity L. Efficiency of LogP implementations can be higher than BSP partly due to its asynchronicity but BSP's simplicity may be desirable. In terms of complexity the two models are equivalent [5].

The BSP machine overestimates execution time for globally unbalanced h-relations. The EBSP model addresses this issue by adding a new (M, k_1, k_2)-relation that incorporates the total number of packets, and the maximum number of sent and received packets [15]. While this machine improves BSP results, it increase analysis effort by adding a rather complex primitive.

The Decomposable BSP machine is a BSP extension that captures submachine locality by executing supersteps on predefined partitions of processors [8] [9] [3]. The $\mathbf{l}$ and $\mathbf{g}$ parameters are vectors whose elements correspond to each partition size; these costs get smaller as partition size decreases. If DBSP overestimates the runtime of unbalanced h-relations, the results are improved by routing each relation as $O(log(p))$ supersteps using the DBSP (k_1, k_2)-routing algorithm from [12]. On DBSP$(n, \mathbf{g}^{(\alpha)}, \mathbf{l}^{(\beta)})$ machines (h, m)-relations can be routed in $O(\lceil m/n \rceil^{\alpha} h^{1-\alpha} n^{\alpha} + n^{\beta})$ time, matching the results obtained by EBSP [4, 11]. If this bound is optimal when considering all routing problems within a class, the solution is not optimal for any such problem. There are un-

balanced routing problems for which αDBSP improves the DBSP bounds or allows a simpler analysis (Section 4). Furthermore, the αDBSP machine abstraction allows more than estimating time complexity; for tight relations the α factor gives a lower bound on required hierarchical link bandwidth which is useful for capacity planning.

6. Concluding Remarks

In this paper we have presented αDBSP, an extension of the DBSP model that augments h-relations with an exponential bandwidth growth factor α. This added factor allows better cost analysis by providing an upper bound on the number of packets traveling up to each machine level. We present two routing algorithms for (h, α)-relations that improve DBSP results; Theorem 3.1 describes an algorithm that executes on any DBSP machine but sends fewer packets compared to the (k_1, k_2)-routing algorithm, while Theorem 3.2 further exploits the ability of the pruned butterfly to route sparse relations faster than $O(\sqrt{p})$. Table 1 shows the αDBS machine parameters computed using the above routing algorithms. On the mesh and pruned butterfly topologies the bandwidth parameter $g(i, \alpha)$ improves the DBSP $g(i)$ parameter by p^{α_M}, p^{α} or $p^{\alpha_M}/log(p)$ depending on the α_M machine factor, and the α relation factor. These results are applied by Section 4 to analyze a few computation kernels employed by scientists to solve problems such as the Earth atmospheric simulation. This exercise shows αDBSP improves DBSP for both the FFT and PDE kernels, at the same time keeping analysis complexity virtually unchanged (Table 2).

The lower bound on (h, α)-relation execution time can be easily computed making a bandwidth argument (Table 1). A heuristic routing algorithm running on a real world interconnect will most likely approach this bound if contention is avoided by distributing messages over multiple paths; based on this assumption we notice an (h, α)-relation executes in linear time on a machine with $\alpha_M \leq \alpha$. To avoid bandwidth bottlenecks for a class of applications, machines should be designed to meet at least this criteria. For some machines and problem configurations the routing algorithms presented in this paper allow some slack; investigating alternative deterministic or randomized algorithms for these cases can lead to improved results.

References

[1] V. Balaji and Robert W. Numrich. A Uniform Memory Model for Distributed Data Objects on Parallel Architectures. In Walter Zwieflhofer and George Mozdzynski, editors, *Use of High-Performance Computing in Meteorology*, pages 272–294. World Scientific Publishing Co., 2005.

[2] P. Bay and G. Bilardi. Deterministic on-line routing on area-universal networks. In *SFCS '90: Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 297–306 vol.1, Washington, DC, USA, 1990. IEEE Computer Society.

[3] Martin Beran. Decomposable bulk synchronous parallel computers. In *SOFSEM '99: Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics on Theory and Practice of Informatics*, pages 349–359, London, UK, 1999. Springer-Verlag.

[4] G. Bilardi, C. Fantozzi, A. Pietracaprina, and G. Pucci. On the effectiveness of D-BSP as a bridging model of parallel computation. In *ICCS '01: Proceedings of the International Conference on Computational Science-Part II*, pages 579–588, London, UK, 2001. Springer-Verlag.

[5] G. Bilardi, K.T. Herley, A. Pietracaprina, G. Pucci, and P. Spirakis. BSP vs LogP. In *SPAA '96: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pages 25–32, New York, NY, USA, 1996. ACM.

[6] Olaf Bonorden, Ben Juurlink, Ingo von Otte, and Ingo Rieping. The Paderborn University BSP (PUB) library. *Parallel Computing*, 29(2):187 – 207, 2003.

[7] D.E. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E.Santos, R. Subramonian, and T. Eicken. LogP: towards a realistic model of parallel computation. In *PPOPP '93: Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 1–12, New York, NY, USA, 1993. ACM.

[8] P. de la Torre and C.P. Kruskal. A structural theory of recursively decomposable parallel processor-networks. *Parallel and Distributed Processing, 1995. Proceedings. Seventh IEEE Symposium on*, pages 570–578, Oct 1995.

[9] Pilar de la Torre and C.P. Kruskal. Submachine locality in the bulk synchronous setting. In *Euro-Par '96: Proceedings of the Second International Euro-Par Conference on Parallel Processing-Volume II*, pages 352–358, London, UK, 1996. Springer-Verlag.

[10] C. Fantozzi, A. Pietracaprina, and G. Pucci. A general PRAM simulation scheme for clustered machines. In *International Journal of Foundations of Computer Science*, volume 14, pages 1147–1164, 2003.

[11] Carlo Fantozzi and Andrea Pietracaprina. Translating submachine locality into locality of reference. In *Print. Special issue on 18th IPDPS*, 2006.

[12] Kieran T. Herley, Andrea Pietracaprina, and Geppino Pucci. Implementing shared memory on mesh-connected computers and on the fat-tree. *Information and Computation*, 165(2):123–143, 2001.

[13] T. Heywood and S. Ranka. A practical hierarchical model of parallel computation. *Parallel and Distributed Processing, 1991. Proceedings of the Third IEEE Symposium on*, pages 18–25, 2-5 Dec 1991.

[14] Jonathan M. D. Hill, Bill McColl, Dan C. Stefanescu, Mark W. Goudreau, Kevin Lang, Satish B. Rao, Torsten Suel, Thanasis Tsantilas, and Rob H. Bisseling. BSPlib: The BSP programming library. *Parallel Computing*, 24(14):1947 – 1980, 1998.

[15] Ben H. H. Juurlink and Harry A. G. Wijshoff. The E-BSP model: Incorporating general locality and unbalanced communication into the BSP model. In *Euro-Par, Vol. II*, pages 339–347, 1996.

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

[17] L.M. Ni, Y. Gui, and S. Moore. Performance evaluation of switch-based wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 8(5):462–474, 1997.

[18] L.G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.