Submachine Locality in the Bulk Synchronous Setting[1]

EXTENDED ABSTRACT

Pilar de la Torre[2] and Clyde P. Kruskal[3]

[2] Department of Computer Science, University of New Hampshire, Durham, NH 03824, U.S.A.
[3] Department of Computer Science, University of Maryland, College Park, MD 20742, U.S.A.

Abstract

The BSP (bulk synchronous parallel) model has been gaining adherents as a standard model for programming parallel computers. When programmed in *direct-mode*, the BSP is supposed to predict runtimes for specific machines based just on three parameters. Although in many situations it does a very good job, sometimes it predicts unnecessarily high levels of slackness. We present two refinements of the BSP model aimed at enhancing its predictive potential: one to account for submachine locality and the other to more accurately reflect router load. We illustrate how the refined models allow one to obtain better estimates of algorithm performance with manageable accounting of costs. In particular, we look at parallel prefix, FFT, and matrix multiplication. The refined models more effectively capture the amount of slackness these problems need to execute efficiently.

1 Introduction

The BSP (bulk synchronous parallel) model [10] has been gaining adherents as a standard model for programming parallel computers. It is supposed to predict runtimes for specific machines just knowing three parameters: machine size, bandwidth, and latency. In many situations it does a very good job. We submit, however, that it sometimes predicts unnecessarily high levels of slackness. We observe:

1. Although the BSP model does capture *strict* (processor) locality, it does not attempt to capture *submachine locality*. Many machines have some submachine locality, which can be exploited to improve runtimes. We will see this effect explicitly for a processor-network model with low bandwidth and low latency.
2. In some algorithms, processors communicate different amounts of data. When calculating runtime, the BSP model charges as if all processors communicate exactly the same amount of data as the busiest processor. For some algorithms, this exaggerates the total amount of data sent and grossly over-predicts runtime. We will see this effect explicitly for the problem of summing on a for processor-network model with low bandwidth and low latency.

Based on these observations, we present two refinements of the BSP model aimed at enhancing its predictive potential. Both of these enhancements can be viewed as the result of transferring to the setting of the BSP a generalization of the principles behind our Y-PRAM model [4]. The first introduces submachine locality; we call this model the *D-BSP*, or *Decomposable BSP*. In contrast with the Y-PRAM, the D-BSP allows arbitrary (irregular) decompositions and can account for costs. The second not only captures submachine locality, but also more accurately accounts for the amount of data being sent through the router; we call this model the *Y-BSP*.

We present D-BSP algorithms for a few of basic problems: parallel prefix, FFT, and matrix multiplication to illustrate its predictive potential. Comparing their performances to

[1] **Contact Author:** Pilar de la Torre; Department of Computer Science; University of New Hampshire; Durham, NH 03824; **E-mail:** dltrr@cs.unh.edu; **Phone:** 603-862-2682; **Fax:** 603-862-3493.

[2] The research of this author was supported in part by the National Science Foundation under Grant CCR-9410592.

algorithms for the standard BSP model points to the D-BSP as potentially capable of more accurate estimates for runtime and slackness requirements. We also show how the parallel prefix computation time estimate can be dramatically improved using the Y-BSP model. The intent of this paper is not to obtain the best possible algorithm for a problem, but rather to point to the potential advantages of using a decomposable model.

RELATED WORK. One of Valiant's goals in inventing the BSP model was to provide a parallel distributed memory computer model, with parameters such as latency and bandwidth, for developing parallel algorithms that reflect their running times on actual parallel machines. Valiant also considered the same issues for shared memory machines, and designed the X-PRAM [11], which is a formalization of the BSP in the shared memory setting. At around the same time, Aggarwal, Chandra, and Snir investigated latency in shared memory machines with their LPRAM and BPRAM models [1, 2] .

We designed the Y-PRAM model [4] to allow more refined analyses than that of Valiant's X-PRAM model. While the ideas behind the Y-PRAM were described specifically to account for communication costs in the shared memory setting, as pointed out in [4], they naturally extend to other settings. In this paper we transfer them to the bulk synchronous parallel setting. An alternative model of submachine locality, the H-PRAM, has been independently proposed by Heywood and Ranka[7]. For a comparative study of the Y-PRAM [4], H-PRAM [7], BSP [10], and LogP [3] models see Ranka et al. [6].

ORGANIZATION. Section 2 reviews the BSP model and introduces the D-BSP, and a restricted variant of it that we call the *recursive D-BSP*. Section 3 discusses the D-BSP time cost of algorithms as a predictor of time costs in specific processor-network models. Section 4 gives examples of algorithms for the BSP and D-BSP models, and compares their estimates of time-costs: parallel prefix, FFT, and matrix multiplication. Section 5 introduces the Y-BSP model, and uses it for a better estimate of parallel prefix time requirements.

2 BSP models

2.1 The BSP model

Roughly speaking, the BSP model [10] can be viewed as the combination of parts:

1. a set of sequential processors with local memory,
2. a *router* that delivers messages point-to-point between pairs of processors,
3. facilities for global synchronization.

The cost of a BSP computation depends on three parameters: machine size, latency, and bandwidth. Loosely, the latency of a machine is how much time it takes to start the router and to synchronize the processors. The bandwidth, which is how much data it can be routed per time unit, was abstracted by Valiant into the model's ability to route h-relations.

COST ASSUMPTION. The BSP assumes that the combined time of delivering any h-relation by the router followed by a global processor synchronization is $gh + l$. Thus g captures the bandwidth inefficiency and l the latency of the machine. The two parameters g and l are normalized by dividing by the time for a local simple operation; otherwise there would be a fourth explicit parameter.

BSP COMPUTATION AND COST. A BSP COMPUTATION consists of a sequence of supersteps. At each superstep, each processor computes only with values held locally within the processor at the start of the superstep, and sends and receives messages. At the end of the superstep, the processors synchronize. The time charge for the superstep is $w + gh + l$, where w is the maximum number of local basic computation steps by any processor during the superstep, and h is the maximum number of messages sent or received by any one processor during the superstep. The time charge for a BSP computation is the sum of the charges for its supersteps.

2.2 The D-BSP model

We extend the BSP model to be able to take advantage of *submachine locality* within a parallel machine. A *decomposable BSP (D-BSP)* model m has four parts:

1. a set P of sequential processors with local memory,
2. a *router* that delivers messages point-to-point between pairs of processors,
3. a collection $\mathcal{S}$ of subsets of processors, including P itself, where each of the subsets can act as a *submachine*, i.e. its processors can exchange messages and synchronize among themselves as an autonomous D-BSP.
4. facilities for *synchronization* within each submachine in $\mathcal{S}$.

The cost of a D-BSP computation depends on two function-parameters: a bandwidth-inefficiency function $\mathcal{G}_y(x)$ and latency function $\mathcal{L}_y(x)$, where variable y sweeps over the space of D-BSP machines and, for each fixed y, x sweeps over the collection a submachines of the machine represented by y.

COST ASSUMPTION. A D-BSP machine m assumes that the router realizes h-relations within each of m's submachines s, and that the combined time of delivering any h-relation followed by a processor synchronization within s is $h\mathcal{G}_m(s) + \mathcal{L}_m(s)$. Thus $\mathcal{G}_m(s)$ captures the bandwidth inefficiency and $\mathcal{L}_m(s)$ captures the latency of submachine s of m.

D-BSP COMPUTATION AND COST. A computation of the D-BSP model m is a sequence of structural-metasteps and terminal-metasteps of submachines of m. These metasteps and their time charges are inductively defined as follows. A *metastep* of submachine s of m is:

- ▷ a *1-metastep (or terminal-metastep)* τ: at which each processor in s computes only with values held locally within the processor at the start of the metastep, and sends and receives messages within submachine s. At the end of the metastep, all processors s synchronize. The time charge is $t(\tau) = w + h\mathcal{G}_m(s) + \mathcal{L}_m(s)$ where w is the maximum number of local computation basic steps executed by any one processor in s, and h is the maximum number of messages sent or received by any processor in s during the metastep.
- ▷ a *r-metastep*, $r > 1$, (or *structural-metastep*) γ: at which, for a given collection $\{\mu_1, \ldots, \mu_r\}$ where each μ_i is a metastep of an independent submachine s_i of s, the processors in each s_i behave according to metastep μ_i. At the end of the metastep all processors in s synchronize. The time of for γ is $t(\gamma) = max_{1 \le i \le r}\{t(\mu_i) + \mathcal{L}_m(s)\}$.
- ▷ a *sequence-metastep* σ: which consists of a sequence of metasteps $\mu_1, \mu_2, \ldots, \mu_k$ of s. Its time charge is $t(\sigma) = \sum_{1 \le i \le k} t(\mu_i)$.

The recursive D-BSP An important sub-class of D-BSP machines consists of 'very regular' machines where, loosely speaking, the machine m can be partitioned into say r equal sized submachines, each of which is a smaller version of the whole machine, and which can themselves be recursively decomposed into r equal sized submachines. Their bandwidth-inefficiency and latency functions depend not on the machine/submachine itself, but rather they depend only on the machine size, denoted by $|m|$. More formally, a D-BSP m is said to be *recursively decomposable into r parts* model, or a *recursive D-BSP model*, if

1. The collection $\mathcal{S}$ of m's submachines is of the form $\mathcal{S} = \mathcal{S}_0 \cup \ldots \cup \mathcal{S}_{\log_r |m|}$ where each $\mathcal{S}_i$ is a partition of m into (approximately) r submachines of size (approximately) $|m|/r^i$, and each submachine of S_i, $i < \log_r |m|$, partitions into submachines of S_{i+1}.
2. There exist functions $g(x)$ and $l(x)$ such that $\mathcal{G}_m(s) = g(|s|)$ and $\mathcal{L}_m(s) = l(|s|)$ for every machine m and every submachine s of m.

3 Processor-Network Models

In order to make our algorithmic results concrete, we show how they apply to a variety of standard, processor-network models. We assume that it takes constant time to send a message through a wire. We take g to be the bandwidth-inefficiency for a large number of messages, and l to be the network diameter or, equivalently, the time it takes to synchronize with no other traffic on the network.

	machines				
parameter functions	linear array	mesh	3-D mesh	tree	hypercube
$g(\|x\|) = \mathcal{G}_y(x)$	x	$x^{1/2}$	$x^{1/3}$	x	1
$l(\|x\|) = \mathcal{L}_y(x)$	x	$x^{1/2}$	$x^{1/3}$	$\lg x$	$\lg x$

In order to match the assumptions on the BSP model, we would like routing to take time $O(gh + l)$. This holds for the linear array, mesh, 3-D mesh, and tree machines, and for non-adaptive routing on the hypercube. Since all algorithms in this paper are non-adaptive, the BSP provides upper bounds for their runtime on those networks. On the other hand, as described in [5], each of those networks has a natural hierarchical partitioning into sub-machines which matches the assumptions of the D-BSP. In fact, the partitioning is regular, with parameter functions $\mathcal{G}_y(x) = g(|x|)$ and $\mathcal{L}_y(x) = g(|x|)$ (see following table) that satisfy the recursive D-BSP assumptions. Thus, the D-BSP also provides time upper bounds for non-adaptive hypercube model algorithms, and for algorithms on the above network models. The question, however, is this: How good are the BSP and D-BSP runtime costs of these algorithms as estimates of time-cost on each specific processor-network model?

4 D-BSP Algorithms

The detailed description of the algorithms and analyses below are in the full version of the paper [5].

Parallel Prefix. This section considers the problem of computing the parallel prefix of $n \geq p$ numbers using the standard tree algorithm. The analysis are summarized below:

- ▷ BSP cost: $g(p) = \mathcal{G}_p(p)$, $l = \mathcal{L}_p(p)$, $\Theta(n/p + (\lg p)(1 + g + l))$
- ▷ D-BSP cost: $\Theta(n/p + \lg p + \sum_{j=1}^{\lg p}(\mathcal{G}_p(2^j) + \mathcal{L}_p(2^j)))$.
- ▷ BSP/D-BSP as cost predictors of costs on specific processor-network models:

machines	parameters		predicted cost when $n \geq p$		time
	$g(p) = \mathcal{G}_p(p)$	$l(p) = \mathcal{L}_p(p)$	BSP	D-BSP	network
linear array	p	p	$n/p + p\log p$	$n/p + p$	$n/p + p$
tree	p	$\log p$	$n/p + p\log p$	$n/p + p$	$n/p + \log p$
mesh	$p^{1/2}$	$p^{1/2}$	$n/p + p^{1/2}\log p$	$n/p + p^{1/2}$	$n/p + p^{1/2}$
hypercube	1	$\log p$	$n/p + (\log p)^2$	$n/p + (\log p)^2$	$n/p + \log p$

Note that for the tree machine, which has high bandwidth inefficiency but low latency, the predictions are much too conservative. We improve this estimate later with the Y-BSP.

Fast Fourier Transform (FFT). This section considers the problem of computing an n-point FFT. It presents implementations of FFT algorithms [9, 1, 10] on the BSP and on the D-BSP. The results of the analysis are summarized below:

- ▷ BSP cost: $g = \mathcal{G}_p(p)$, $l = \mathcal{L}_p(p)$, $\Theta\left(\frac{n\lg n}{p} + \frac{gn\lg n}{p\lg(n/p)} + l\frac{\lg n}{\lg(n/p)}\right)$.

  For n large this matches the result in [10].
- ▷ D-BSP cost: $\Theta\left(\frac{n\lg n}{p} + \frac{n}{p}\sum_{i=1}^{\frac{\lg n}{\lg(n/p)}-1}\mathcal{G}_p((\frac{n}{p})^i) + \sum_{i=1}^{\frac{\lg n}{\lg(n/p)}-1}\mathcal{L}_p((\frac{n}{p})^i)\right)$

▷ BSP/D-BSP as cost predictors of costs on specific processor-network models:

machines	parameters		predicted cost, $n \geq 2p$		time
	$g(p) = \mathcal{G}_p(p)$	$l(p) = \mathcal{L}_p(p)$	BSP	D-BSP	network model
linear array	p	p	$\frac{n \lg n}{p} + n \frac{\lg n}{\lg(n/p)}$	$\frac{n \lg n}{p} + n$	$\frac{n \lg n}{p} + n$
mesh	$p^{1/2}$	$p^{1/2}$	$\frac{n \lg n}{p} + \frac{n}{p^{1/2}} \frac{\lg n}{\lg(n/p)}$	$\frac{n \lg n}{p} + \frac{n}{p^{1/2}}$	$\frac{n \lg n}{p} + \frac{n}{p^{1/2}}$
hypercube	1	$\lg p$	$\frac{n \lg n}{p}$	$\frac{n \lg n}{p}$	$\frac{n \lg n}{p}$

Usually, n will be polynomially larger than p, so $(\lg n)/\lg(n/p)$ will be a constant. Then the two models will give the same prediction, which matches the specific processor-network time. But when n is not much larger than p, the D-BSP provides a slightly better prediction of the slackness.

Matrix Multiplication. This section considers the problem of multiplying two dense $n \times n$ matrices.

NAIVE ALGORITHM. Here we describe an implementation of the naive algorithm both on the BSP and on the D-BSP. For the latter we need to explicitly determine how the elements are being sent between processors. The analysis for $n > p^2$ are summarized below:

▷ BSP cost: $g = \mathcal{G}_p(p)$, $l = \mathcal{L}_p(p)$, $\Theta\left(n^3/p + gn^2/p^{1/2} + l\right)$.

▷ naive algorithm's D-BSP cost: $\Theta\left(\frac{n^3}{p} + \frac{n^2}{p^{1/2}} \sum_{j=0}^{(1/2)\lg p - 1} \frac{\mathcal{G}_p(4^j)}{2^j} + \sum_{j=0}^{(1/2)\lg p - 1} \mathcal{L}_p(4^j)\right)$.

▷ BSP/D-BSP as cost predictors of costs on specific processor-network models:

	parameters		predicted cost when $n^2 \geq p$		time
	$\mathcal{G}_p(p)$	$\mathcal{L}_p(p)$	BSP	D-BSP	network model
linear array	p	p	$n^3/p + n^2p^{1/2} + p$ $\sim n^3/p + n^2p^{1/2}$	$n^3/p + n^2 + p$ $\sim n^3/p + n^2$	$n^3/p + n^2$
mesh	$p^{1/2}$	$p^{1/2}$	$n^3/p + n^2 + p^{1/2}$ $\sim n^3/p + n^2$	$n^3/p + n^2(\lg p)/p^{1/2} + p^{1/2}$ $\sim n^3/p + n^2(\lg p)/p^{1/2}$	$n^3/p + n^2/p^{1/2}$
3-d mesh	$p^{1/3}$	$p^{1/3}$	$n^3/p + n^2/p^{1/6} + p^{1/3}$ $\sim n^3/p + n^2/p^{1/6}$	$n^3/p + n^2/p^{1/2} + p^{1/3}$ $\sim n^3/p$	n^3/p
hyper-cube	1	$\lg p$	$n^3/p + n^2/p^{1/2} + \lg p$ $\sim n^3/p$	$n^3/p + n^2/p^{1/2} + (\lg p)^2$ $\sim n^3/p$	n^3/p

Since $n^2 \geq p$ some of the formulas can be simplified as shown. For models with high bandwidth inefficiency the D-BSP gives a better prediction of slackness.

MORE SOPHISTICATED ALGORITHM. Applying the basic ideas of the naive algorithm we develop a more sophisticated algorithm. Versions of this algorithm were implemented by Aggarwal, Chandra, and Snir [2] for the LPRAM, and by McColl and Valiant [8] for the BSP which has time $\Theta(n^3/p + (n^2/p^{2/3})g + l)$. The analyses result of our D-BSP implementation, for $n^3 \geq p$, are summarized below:

▷ BSP cost: $g = \mathcal{G}_p(p)$, $l = \mathcal{L}_p(p)$, $\Theta\left(n^3/p + (n^2/p^{2/3})g + l\right)$.

▷ D-BSP cost: $\Theta\left(\frac{n^3}{p} + \frac{n^2}{p^{2/3}} \sum_{i=0}^{(1/3)\lg p - 1} \frac{\mathcal{G}_p(8^i)}{2^i} + \sum_{i=0}^{(1/3)\lg p - 1} \mathcal{G}_p(8^i) + \sum_{i=0}^{(1/3)\lg p - 1} \mathcal{L}_p(8^i)\right)$.

▷ BSP/D-BSP as cost predictors of costs on specific processor-network models:

	parameters		predicted cost when $n^3 \geq p$		time
	$\mathcal{G}_p(p)$	$\mathcal{L}_p(p)$	BSP	D-BSP	network model
linear array	p	p	$n^3/p + n^2p^{1/3} + p$ $\sim n^3/p + n^2p^{1/3}$	$n^3/p + n^2 + p$ $\sim n^3/p + n^2)$	$n^3/p + n^2 + p$
mesh	$p^{1/2}$	$p^{1/2}$	$n^3/p + n^2/p^{1/6} + p^{1/2}$ $\sim n^3/p + n^2/p^{1/6}$	$n^3/p + n^2/p^{1/2} + p^{1/2}$ $\sim n^3/p + n^2/p^{1/2} + p^{1/2}$	$n^3/p + n^2p^{1/2} + p^{1/2}$
3-d mesh	$p^{1/3}$	$p^{1/3}$	$n^3/p + n^2/p^{1/3} + p^{1/3}$ $\sim n^3/p + n^2/p^{1/3}$	$n^3/p + n^2(\lg p)/p^{2/3} + p^{1/3}$ $\sim n^3/p + p^{1/3}$	$n^3/p + p^{1/3}$
hyper-cube	1	$\lg p$	$n^3/p + \lg p$ $\sim n^3/p + \lg p$	$n^3/p + (\lg p)^2$ $\sim n^3/p + (\lg p)^2$	$n^3/p + n^2/p^{2/3} + \lg p$

Since, $n^3 \geq p$ some of the formulas can be simplified as shown. As above, when the bandwidth inefficiency is small, one would use the BSP algorithm on the D-BSP.

5 Y-BSP

As discussed earlier the BSP model charges as though all processors were sending the same amount of data as the busiest processor. Although the D-BSP accounts for submachine locality, the runtime prediction for summing on a tree is poor. This is because within a submachine, the D-BSP charges exactly the same for a single message exchange as it does for routing a full 1-relation. More realistic estimates of runtime can be achieved by calculating the total amount of data sent by the processors. This gives a true picture of the load on the router. If we do so, we must keep track separately of the maximum number of messages sent or received by any one processor. There is usually an overhead cost of preparing a message for transmission; we need a new parameter to take this into account. We denote this cost by b and we view it as a function $b(s)$ of the submachine. Although it is often the case that this parameter is independent of submachine size, it can be a very large constant which is worthwhile keeping track of explicitly in the analysis of algorithms. The Y-BSP model is exactly the same as the D-BSP model except that costs are calculated slightly differently.

Y-BSP COST ASSUMPTION. The Y-BSP assumes that the router realizes h-relations within submachines. Within a submachine s, the combined time of realizing any h-relation followed by a processor synchronization is $hb(s) + (k/|s|)\mathcal{G}_m(s) + \mathcal{L}_m(s)$, where k is the sum total of messages sent by all processors. Thus $\mathcal{G}_m(s)$ is the bandwidth inefficiency and $\mathcal{L}_m(s)$ is the latency of submachine s. Notice that if every processor is sending exactly h messages, as is often the case, then $h = \frac{k}{|s|}$ and the Y-BSP time charge formula is the same as the BSP formula, except that $b(s) + \mathcal{G}_m(s)$ is the D-BSP's $\mathcal{G}_m(s)$.

The definitions of metasteps and their charges are analogous to those for the D-BSP with the exception of the way in which the Y-BSP charges for 1-metasteps: The time charge for a 1-metastep is $w + h\,b(s) + (k/|s|)\mathcal{G}_m(s) + \mathcal{L}_m(s)$: where k is the total number of messages sent or received by all of the processors during the superstep; w the maximum number of local computation basic steps executed by any one processor of s's, and h be the maximum number of messages sent or received by any one of s's processors during the metastep. The time for a Y-BSP computation on the whole machine is analogous to the corresponding definition for the D-BSP.

PARALLEL PREFIX REVISITED. Using the same algorithm as earlier, the Y-BSP analysis yields the following total time for a parallel prefix computation of $n > p$ items:

$$\Theta(n/p) + \sum_{j=1}^{\lg p} \Theta(1) + \sum_{j=1}^{\lg p} \Theta(b(2^j)) + \sum_{j=1}^{\lg p} \Theta(\frac{\mathcal{G}_p(2^j)}{2^j}) + \sum_{j=1}^{\lg p} \Theta(\mathcal{L}_p(2^j))$$

If, $\mathcal{L}_p(p) = \Omega(1)$, $\mathcal{G}_p(p) = O(p)$, and $b(p) = O(\mathcal{L}_p(p))$, this simplifies to $\Theta(n/p + \sum_{j=1}^{\lg p} \mathcal{L}_p(2^j))$. For the parameters of the tree machine this is $\Theta(\frac{n}{p} + (\lg p)^2)$, which is a much better estimate than that provided by the BSP or D-BSP models.

ACKNOWLEDGMENTS The authors thank Rob Bisseling for suggesting the name decomposable BSP, Matt Plumlee for editing, and both of them for helpful discussions.

References

[1] A. Aggarwal, A. Chandra, and M. Snir. On communication latency in PRAM computations. In *Proc. ACM 1st Ann. Symp. on Parallel Algorithms and Architectures*, pages 11–21, Sante Fe, New Mexico, 1989.

[2] A. Aggarwal, A. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71:3–28, 1990.

[3] D. Culler, R. Karp, D. Patterson, A. Sahay, K.R. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Log p: Towards a realistic model of parallel computation. In *4th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming* , pages 1–12, May 1993.

[4] P. de la Torre and C.P. Kruskal. Towards a single model of efficient computation in real parallel machines. In *Parallel Architectures and Languages Europe (PARLE'91)*, E.H.L. Aarts, J. van Leeuwen, and M. Rem, editors, pages 6–24. Lecture Notes in Computer Science, Springer-Verlag, vol 505, 1991. Journal version appeared in *Future Generations Computer Systems*, 8:395–408, 1992.

[5] P. de la Torre and C.P. Kruskal. Submachine locality in the bulk synchronous setting. Technical Report 96-03, Department of Computer Science, University of New Hampshire, May 1996.

[6] A. Gramma, V. Kumar, S. Ranka, and V. Singh. On architecture independent design and analysis of parallel programs. Manuscript, 1995.

[7] T. Heywood and S. Ranka. A practical model of parallel computation: I. The model. In *Proc. 3rd IEEE Symp. on Parallel and Distributed Processing*, December 1991. Journal version appeared *Journal of Parallel and Distributed Algorithms*, 16: 212–232.

[8] W. F. McColl. Scalable parallel computing. To appear in *LNCS Volume 1000*, J. van Leeuwen, editor, August, 1995.

[9] C. Papadimitriou, and M. Yannakakis. Towards an Architecture-Independent Analysis of Parallel Algorithms. In *Proc. of the 20th Ann. ACM Symp. on Theory of Computing*, pages 510–513, 1988.

[10] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, pages 103–111, 1990.

[11] L.G. Valiant. General purpose parallel architectures. In *Handbook of Theoretical Computer Science, Vol. A*, J. van Leeuwen, editor, pages 943–971. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.