On the Predictive Quality of BSP-like Cost Functions for NOWs*

(Extended Abstract)

Mauro Bianco and Geppino Pucci

Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy.
{bianco1,geppo}@dei.unipd.it

Abstract. The Bulk-Synchronous Parallel (BSP) model [16] provides a simple and portable programming discipline that is particularly suitable for coarse-grained parallel systems such as Networks of Workstations (NOWs). In this work we examine the issue of predictability of the BSP cost function for a NOW consisting of SUN workstations connected through a 10Mbps Ethernet network. In particular, we compare the original BSP cost function with a number of newly proposed variants, with the intent of improving predictability by having the cost function encompass those parameters of the hardware/software system which have the largest impact on performance.

1 Introduction

It is widely recognized [15,5] that the quest for a desirable model of parallel programming is made particularly hard by the objective of achieving the following three properties simultaneously: usability, portability and predictability. *Usability* refers to the ease of designing, analyzing, and coding algorithms in the framework provided by the model. *Portability* denotes the ability of compiling and running programs written according to the model over a wide class of target platforms, achieving good performance on each platform. Finally, *Predictability* implies the ability of the model of forecasting performance of a piece of software via an associated cost function. In this paper, we investigate this latter issue for the *Bulk Synchronous Parallel* (BSP) programming model proposed in [16] in the context of low-end parallel systems made of Networks of Workstations (NOWs).

The BSP model provides an abstract machine made of P processors with local memory, connected by a router which implements batch communication via message passing. Computation is divided into phases, named *supersteps*, each terminated by a barrier synchronization. During a superstep, the processors may execute local computation on data held locally at the beginning of the superstep, and/or exchange messages with other processors. The messages sent during a superstep are made available by the router to their destinations only at the beginning of the next superstep.

The running time of a BSP program is obtained by summing the running times of its constituent supersteps. The execution time of a superstep can be expressed as linear cost function which has the following form [14]:

$$T_{ss}(w,h) = w + gh + l \ , \tag{1}$$

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

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 638–646, 2000.

where w is the local computation time and h is the degree of the *relation* realized by the router, that is, the maximum number of bytes sent or received by any processor. Parameters g and l are meant to capture, respectively, the bandwidth and latency characteristics of the underlying architecture.

The simple programming paradigm offered by BSP implies a good level of usability. Also, the inherently machine-independent nature of its communication mechanism, based on batch communication, allows optimized implementations on a large spectrum of parallel architectures, hence fostering efficient portability. However, it has often been noted that the BSP cost function offers only a coarse level of predictability [12,11]. In fact, such observation has motivated further research into defining more descriptive (hence, less usable) models which embody additional aspects of a machine that impact performance (e.g., message injection overhead [6], or clustering [9]). In this paper, we take a different approach. Rather than changing the BSP programming model, we seek to improve its predictability by striving for a tighter coupling between its associated cost function and those features of the hardware/software system under consideration which have the greatest impact on performance. By intervening on the cost function only, we aim at enhancing predictability while preserving usability and portability of the programming model as much as possible.

The programming environment used in this work is based on the message-passing primitives provided by the *BSPlib* library developed by the *Parallel Applications Centre* of Oxford University [10]. *BSPlib* has been installed on a NOW of 10 SUN SPARCstations available at our department, connected by a 10Mbps Ethernet under the UDP/IP protocol [8]. Under *BSPlib*, interprocessor communication occurs when barrier synchronization is called at the end of each superstep, and is realized through a kind of randomized, time-division multiplexing technique [7]. More specifically, time is divided into *time-slices*, which are in turn divided into as many *time-slots* as the number of sending processors. At each time-slice, the sending processors randomly choose a time-slot for sending their messages over the Ethernet. Randomization helps the system pick a transmission schedule that makes a good usage of the available bandwidth of the communication medium. The time-slot duration depends on the maximum Ethernet frame size supported, and packet fragmentation is done at the library level accordingly. As a consequence, when using *BSPlib* there is a limited payoff in orchestrating communication at the program level so to send one (very) long message rather than many (relatively) short ones, hence we can safely refer only to the total amount of bytes sent from one processor to another.

1.1 Our Contribution

The main purpose of this work is to estimate the relative accuracy and the ease of use of a set of cost functions alternative to the classical BSP function of Equation (1) for the hardware/software system under consideration. Although our quantitative results are system-specific, the proposed methodology is rather general and applicable to a wide range of parallel platforms.

We describe the message routing instance associated to a BSP superstep by means of a *communication pattern*, which can be envisioned as a $P \times P$ array containing, for each processor, the number of bytes that the processor sends to any other processor (including itself). The BSP cost function yields the same prediction for all communication patterns which realize an h-relation. However, h might be too drastic a summary for the characteristics of a communication pattern, hence unsuitable to differentiate among those that have the same value of h but feature very different execution times.

To achieve a more effective (yet simple) categorization, we follow a classic approach in routing theory [13,12] and summarize a communication pattern as an (h_i, h_o, M)-*relation*, where h_i (resp., h_o) is the maximum number of bytes received (resp., sent) by any processor and M is the total number of bytes exchanged by the processors.

The candidate cost functions that we consider are the following linear combinations of the parameters h_i, h_o, M and $h = \max\{h_i, h_o\}$:

1. $\mathcal{F}_h(h) = g \cdot h + l$
2. $\mathcal{F}_{io}(h_i, h_o) = g_i \cdot h_i + g_o \cdot h_o + l$
3. $\mathcal{F}_{ioM}(h_i, h_o, M) = g_i \cdot h_i + g_o \cdot h_o + g_M \cdot M + l$
4. $\mathcal{F}_{hM}(h, M) = g \cdot h + g_M \cdot M + l$
5. $\mathcal{F}_M(M) = g_M \cdot M + l$
6. $\mathcal{F}_{oM}(h_o, M) = g_o \cdot h_o + g_M \cdot M + l$
7. $\mathcal{F}_{iM}(h_i, M) = g_i \cdot h_i + g_M \cdot M + l$
8. $\mathcal{F}_o(h_o) = g_o \cdot h_o + l$
9. $\mathcal{F}_i(h_i) = g_i \cdot h_i + l$

In order to obtain the coefficients for the above cost functions, we execute an extensive set of carefully designed communication patterns, whose objective is to exercise a large number of feasible combinations of the three parameters h_i, h_o and M. The running times collected for such patterns are then used to infer the cost functions through least-square fitting. Finally, the predictive quality of the functions is validated on a suite of additional, synthetic access patterns and on a small set of sorting applications.

2 Fitting the Cost Functions

Consider a P-processor BSP machine, where the i-th processor is denoted by P_i, with $0 \leq i \leq P-1$. Let also $\mathcal{H}_1 = \{h : h = 10000 + 30000 \cdot i, i = 0, \ldots, 3\}$, $\mathcal{H}_2 = \{h : h = 150000 + 75000 \cdot i, i = 0, \ldots, 11\}$ and $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_2$. Finally, let x be an integer parameter.

For each value of $h \in \mathcal{H}$ and $1 \leq x \leq P$, we define the following *synthetic* communication patterns, that will be used for fitting and validating the cost functions:

- (h, x)-`scatter` ($h_o = h$, $h_i = h \cdot x/P$, $M = h \cdot x$). For $0 \leq i \leq x-1$ and $0 \leq j \leq P-1$, P_i sends h/P bytes to P_j.
- (h, x)-`gather` ($h_i = h$, $h_o = h \cdot x/P$, $M = h \cdot x$). For $0 \leq i \leq P-1$ and $0 \leq j \leq x-1$, P_i sends h/P bytes to P_j.
- (h, x)-`square` ($h_i = h$, $h_o = h$, $M = h \cdot x$). For $0 \leq i \leq x-1$ and $P-x \leq j \leq P-1$, P_i sends h/x bytes to P_j, with $j = P-x, \ldots, P-1$.
- `random-`(h, x)`-scatter`. A random communication pattern uniformly generated among all those with $h_o = h$, $h_i = h \cdot x/P$ and $M = h \cdot x$.
- `random-`(h, x)`-gather`. A random communication pattern uniformly generated among all those with $h_i = h$, $h_o = h \cdot x/P$ and $M = h \cdot x$.
- `random-`(h, x)`-square`. A random communication pattern uniformly generated among all those with $h_i = h$, $h_o = h$ and $M = h \cdot x$.

In order to filter out noise, each pattern is executed 20 times and the running time is considered to be the median of the 20 executions. Together, the first three families of patterns (obtained by varying $h \in \mathcal{H}$ and $1 \leq x \leq P$) make up *Suite 1*, which contains *deterministic patterns*, while the last three make up *Suite 2* which is made of *random patterns* sharing the same summary parameters of their deterministic counterparts. Note

	$\mathcal{F}_h$	$\mathcal{F}_{io}$	$\mathcal{F}_{ioM}$	$\mathcal{F}_{hM}$	$\mathcal{F}_M$	$\mathcal{F}_{oM}$	$\mathcal{F}_{iM}$	$\mathcal{F}_o$	$\mathcal{F}_i$
$g \cdot 10^6$	3042			2207					
$g_i \cdot 10^6$		587.2	552.6				1433		2850
$g_o \cdot 10^6$		2932	2898			3066		3386	
$g_M \cdot 10^6$			22.94	334.1	891.5	119.8	530.8		
l	41.57	25.21	26.59	41.47	396.1	67.79	242.6	76.30	280.3

(a) $P = 4$

	$\mathcal{F}_h$	$\mathcal{F}_{io}$	$\mathcal{F}_{ioM}$	$\mathcal{F}_{hM}$	$\mathcal{F}_M$	$\mathcal{F}_{oM}$	$\mathcal{F}_{iM}$	$\mathcal{F}_o$	$\mathcal{F}_i$
$g \cdot 10^6$	6520			4742					
$g_i \cdot 10^6$		644.3	427.1				2336		5699
$g_o \cdot 10^6$		7070	6853			6972		7531	
$g_M \cdot 10^6$			77.61	395.0	994.9	116.4	700.5		
l	104.9	74.51	84.06	104.9	995.1	122.6	702.7	142.8	824.5

(b) $P = 8$

Fig. 1. Cost function coefficients (in $m$secs).

that the Suites exercise a vast spectrum of feasible values of the 3-tuple (h_i, h_o, M), which is crucial to achieve reliable fits. In particular, by varying x, we obtain patterns characterized by a varying amount and distribution of the global communication traffic, but featuring the same value h of maximum outbound/inbound traffic from/to the same processor. Note that scatter-like (resp., gather-like) patterns are likely to incur in higher overhead during message injection (resp., receipt) since $h = h_o \geq h_i$ (resp., $h = h_i \geq h_0$).

We use the two Suites of patterns to fit (over the patterns in Suite 2) and validate (over the deterministic patterns in Suite 1) the BSP-like cost functions defined in the previous sections for two submachines of 4 and 8 processors, respectively. The coefficients of the cost functions obtained for $P = 4$ and $P = 8$ are shown in Fig. 1.

3 Validation Results

The results of the validations of the cost functions on Suite 1 are shown in Fig. 2, where for each submachine and each cost function we report, respectively, the maximum and the average relative errors incurred by approximating the running time of a pattern with the value returned by the function. Note that for $P = 4$, the two-parameter function $\mathcal{F}_{io}$ behaves better, on average, than the three-parameter function $\mathcal{F}_{ioM}$. This counter-intuitive phenomenon can be explained if we consider that the least-square function obtained from the fitting minimizes the $\| \cdot \|_2$-error, while we have chosen to check the quality of our functions against the (more intuitive) metric of relative error between predicted and measured running time. However, when $P = 8$, $\mathcal{F}_{ioM}$ becomes slightly more predictive than $\mathcal{F}_{io}$, which provides evidence that the impact of parameter M becomes more important as P grows.

Also, note that the classical $\mathcal{F}_h$ function is consistently much worse than functions $\mathcal{F}_o$, $\mathcal{F}_{io}$ and $\mathcal{F}_{ioM}$, and that all functions including h_o as a parameter behave decidedly better than those not including it. In retrospect, this behaviour can be explained by the message-scheduling strategy implemented by *BSPlib*[7], where the number of time-

		$\mathcal{F}_h$	$\mathcal{F}_{io}$	$\mathcal{F}_{ioM}$	$\mathcal{F}_{hM}$	$\mathcal{F}_M$	$\mathcal{F}_{oM}$	$\mathcal{F}_{iM}$	$\mathcal{F}_o$	$\mathcal{F}_i$
$P = 4$	Max. Err.(%)	168	17.2	16.4	124	810	99.3	489	120	594
	Ave. Err.(%)	24.7	9.4	9.6	25.7	95.6	18.0	68.1	19.2	78.1
$P = 8$	Max. Err.(%)	425	65.4	62.5	316	559	51.0	379	44.9	461
	Ave. Err.(%)	34.8	7.31	6.80	31.5	86.4	7.54	70.6	7.86	87.7

Fig. 2. Maximum and average validation errors on Suite 1.

slots and time-slices (hence, the duration of the routing) mainly depends on h_o and is independent of h_i.

In addition, we note that when $P = 8$, function $\mathcal{F}_o$ is roughly as predictive as functions $\mathcal{F}_{oM}$, $\mathcal{F}_{io}$ and $\mathcal{F}_{ioM}$, the other parameters embodied by the latter functions having only a second-order effect on improving predictability. Therefore, the simple $\mathcal{F}_o$ function (in fact, even simpler than the classical BSP $\mathcal{F}_h$ function) represents the best compromise between accuracy and simplicity of prediction for a moderately-sized machine. Since the impact of the overall traffic volume (as measured by M) on predictive quality seems to increase with P, it is reasonable to assume that for larger systems, function $\mathcal{F}_{oM}$ would be a better choice.

From our analysis it follows that parameter h_i may be disregarded on our system, since communication time does not seem to depend crucially on the number of messages received by a processor. On the other hand, communication time exhibits a strong linear dependence on parameter h_o. Finally, the synthesis between this two parameters used by the classical BSP function $\mathcal{F}_h$, does not seem to yield good predictions.

In order to fully appreciate the crucial impact of h_o on performance, in Fig. 3 we plot the execution times of some patterns (for varying values of h) in Suite 1, together with all the cost functions under examination, for $P = 8$. Note that when $x = 8$, (h, x)-`scatter`, (h, x)-`gather` and (h, x)-`square` patterns all become *total exchange* patterns, with all processors sending/receiving h/P bytes to/from one another, hence Fig. 3(b) ($(h, 8)$-`gather`) also represents an $(h, 8)$-`scatter` or an $(h, 8)$-`square`. By comparing Figg. 3(a) and 3(b) we note that the running time of an (h, x)-`gather` heavily depends on x, while a comparison of Fig. 3(b) with Figg. 3(c) and 3(d) reveals that there is no such dependency for (h, x)-`scatter` and (h, x)-`square`. Finally, it is very clear from the plots that all functions including h_o as a parameter are much better predictors than the remaining functions, which give rather poor predictions especially for unbalanced patterns (small values of x).

In summary, our experiments imply that one can obtain reliable performance predictions on the hardware/software system under study by adopting a simple variant of the classical BSP cost function, where the contribution of parameter h_o is made explicit. More importantly, we want to point out that *BSPlib* attains such level of predictability while making good use of the hardware, since the peak transmission bandwidth observed during our experiments (8.8Mbps for total exchange patterns) comes close to 90% of the maximum available bandwidth of the communication medium (10Mbps).

4 Predicting the Communication Time of Sorting Algorithms

To test the quality of the above cost functions in real scenarios, we have exercised them on predicting the communication time of *BSPlib* implementations of three classical sorting algorithms, namely, Batcher's *Bitonic Sort* [2]; a simple parallelization of the

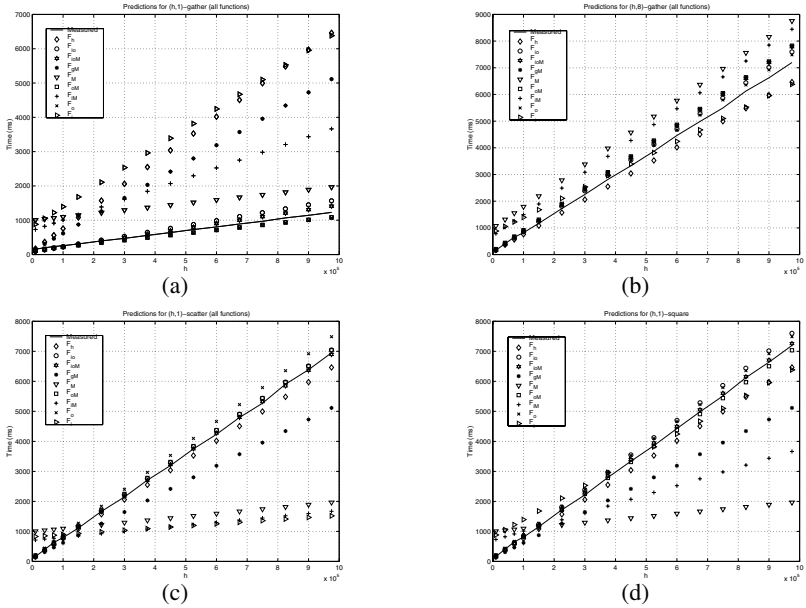

Fig. 3. Running times and predictions for (h, x)-`gather`, (h, x)-`scatter` and (h, x)-`square`, for $x = 1{,}8$. (Fig. 3(b) is also a plot for $(h, 8)$-`scatter` and $(h, 8)$-`square`.)

Radix Sort algorithm for integer sorting; and finally *Sample Sort* with oversampling [4]. We measured the communication times of each constituent superstep by subtracting the time required for local computation from the overall running time of the superstep. (More details on the algorithms will be provided in the full version of this extended abstract.)

Let $\mathcal{N}_1 = \{N : N = 2500 + 7500 \cdot i, i = 0, \ldots, 3\}$, $\mathcal{N}_2 = \{N : N = 37500 + 18750 \cdot i, i = 1, \ldots, 11\}$ and $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$. The sorting algorithms have been executed with random inputs of size $N \cdot P$, for each $N \in \mathcal{N}$, with measured communication times chosen as the median time out of five executions.

The table in Fig. 4 compares the maximum and average prediction errors incurred, for each sorting algorithm, by the BSP function $\mathcal{F}_h$ and the functions that turned out to be better predictors on the synthetic patterns, namely $\mathcal{F}_{io}$, $\mathcal{F}_{ioM}$, $\mathcal{F}_{io}$ and $\mathcal{F}_o$. Also, Fig. 5 plots the measured communication times against the predictions of $\mathcal{F}_h$ (the worst function) and $\mathcal{F}_{ioM}$ (the best function) as a function of N. As before, it is clear that functions including parameter h_o yield much better predictions than $\mathcal{F}_h$, although the difference in quality is not so dramatic as the one observed on the patterns in Suite 1. This relatively better behaviour of the $\mathcal{F}_h$ function is mainly due to the fact that the most expensive communication patterns (at least for radix and sample sort) generated by the sorting applications tend to be total exchanges of N/P^2-length messages, which are those on which $\mathcal{F}_h$ incurs into the least prediction errors, since such patterns have $h = h_i = h_0$, hence coalescing the indegree and the outdegree of the relation into the "summary" parameter h does not imply a large loss of information. Consequently, the improvement of over 50% on the quality of the predictions provided by the more com-

plex $\mathcal{F}_{oM}$ and $\mathcal{F}_{ioM}$ functions can be explained mainly with the presence of parameter M, which captures the impact of the overall traffic volume generated by the pattern.

	Bitonic		**Radix**		**Sample**	
	$P=4$	$P=8$	$P=4$	$P=8$	$P=4$	$P=8$
$\mathcal{F}_h$	43.51	26.52%	17.90	16.43%	22.77	11.10%
$\mathcal{F}_{io}$	35.74	16.15%	13.31	9.93%	17.35	5.01%
$\mathcal{F}_{ioM}$	35.45	13.92%	12.26	6.42%	18.72	4.01%
$\mathcal{F}_{oM}$	36.98	13.62%	11.78	3.17%	42.18	7.37%
$\mathcal{F}_o$	39.69	16.91%	15.24	6.72%	42.91	10.25%

Fig. 4. Average prediction errors for bitonic sort, radix sort and sample sort for $P = 4, 8$ and $n \in \mathcal{N}$

The data collected for bitonic sorting are definitively the most puzzling. Although the communication patterns generated by the algorithm are extremely regular (namely, permutations of N/P-length messages) *all* the cost functions tend to severely underestimate the associated running time. We conjecture that this phenomenon is due to a suboptimal management of this important class of communication patterns by the scheduling algorithm provided by the *BSPlib* library. Note, however, that even in this case, functions embodying the h_o parameter are much better predictors than the BSP function $\mathcal{F}_h$.

5 Future Work

Further investigation is needed to determine to which extent the newly proposed cost functions can be effectively used in practice as an alternative to the classic BSP function to enhance predictability. We believe that the value of separating the contributions of h_i and h_o and adding a parameter of global congestion of the communication medium such as M will prove to be even more substantial for applications characterized by more irregular communication patterns than sorting. In order to substantiate this intuition, we are thinking of exercising our cost functions over a bulk-synchronous version of the *NAS* benchmarks [1].

An orthogonal line of investigation concerns devising cost functions for other network architectures, such as 100Mbps or Gigabit Ethernet, Myrinet or ATM, or comparing the performance/predictability levels achieved by *BSPlib* against those attained by other communication libraries, such as the BSP *PUB* library developed at Paderborn University [3].

Acknowledgments We are grateful to Nancy Amato and Andrea Pietracaprina for helping set the ground for this research. We also wish to thank the EUROPAR referees for their valuable comments and suggestions.

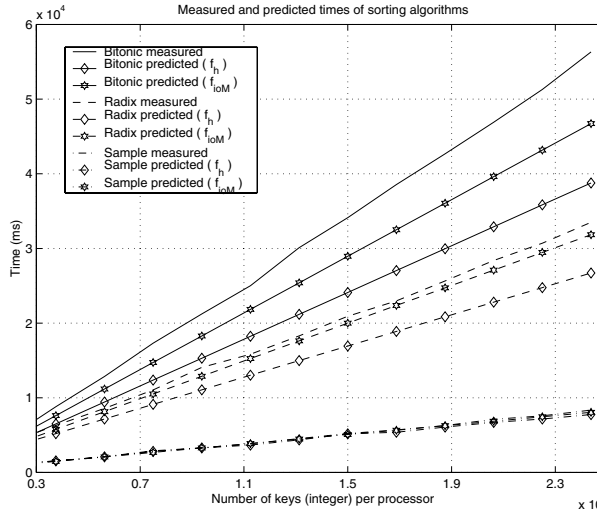

Fig. 5. Measured versus predicted (functions $\mathcal{F}_h$ and $\mathcal{F}_{ioM}$) communication times for Bitonic, Radix and Sample sort.

References

1. D. Bailey, E. Barszcz, J.T. Barton, *et al.* The NAS parallel benchmarks. *Int. J. Supercomputer Appl.*, 5(3):63–73, 1991.
2. K.E. Batcher. Sorting networks and their applications. In *Proc. of the AFIPS Spring Joint Computer Conf.*, pages 307–314, 1968.
3. O. Bonorden, B. Juurlink, I. von Otte, and I Rieping. The Paderborn university BSP (PUB) Library – Design, Implementation and Performance In *Proc. of the 2nd merged IPPS/SPDP Symp.*, pages 99-104, San Juan, Puerto Rico, April 1999.
4. G. Blelloch, C.E. Leiserson, B.M. Maggs, *et al.* A comparison of sorting algorithms for the connection machine CM-2. In *Proc. of the 3rd ACM Symp. on Parallel Algorithms and Architectures*, pages 3–16, Hilton Head SC, USA, 1991.
5. G. Bilardi, A. Pietracaprina, and G. Pucci. A quantitative measure of portability with application to bandwidth-latency models for parallel computing. In *Proc. EURO-PAR'99 – Parallel Processing*, pages 543–551, Toulouse, F, Aug./Sep 1999.
6. D.E. Culler, R. Karp, D. Patterson, *et al*. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, November 1996.
7. S.R. Donaldson, J.M.D. Hill, and D.B. Skillicorn. Predictable communication on unpredictable networks: Implementing BSP over TCP/IP. In *Proc. EURO-PAR'98 – Parallel Processing*, pages 970–980, Southampton, UK, September 1998.
8. S.R. Donaldson, J.M.D. Hill, and D.B. Skillicorn. BSP clusters: high performance, reliable and very low cost. Technical report PRG-TR-5-98, Oxford University Computing Laboratory, Oxford, UK, 1998.
9. P. De la Torre and C.P. Kruskal. Submachine locality in the bulk synchronous setting. In In *Proc. EURO-PAR'96 – Parallel Processing*, pages 352–358, Lyon, F, August 1996.
10. M. Goudreau, J.M.D. Hill, W. McColl, S. Rao, D.C. Stefanescu, T. Suel, and T. Tsantilas. A proposal for the BSP worldwide standard library. BSP Worldwide `http://www.bsp-worldwide.org/`, April 1996.
11. M. Goudreau, K. Lang, S. Rao, *et al.* Portable and efficient parallel computing using the BSP model. *IEEE Trans. on Computers*, C-48(7):670–689, July 1999.

12. B.H.H. Juurlink and H.A.G. Wijshoff. A quantitative comparison of paralle computation models. *ACM Trans. on Computer Systems*, 16(3):271–318, 1998.
13. F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.
14. W.F. McColl. BSP programming. *DIMACS Series in Discrete Mathematics*, pages 21–35. American Mathematical Society, 1994.
15. B. Maggs, L.R. Matheson, and R.E. Tarjan. Models of parallel computation: A survey and synthesis. In *Proc. of the 28th Hawaii Int. Conf. on System Sciences (HICSS)*, volume 2, pages 61–70, January 1995.
16. L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.