Programming Research Group

NETWORK PERFORMANCE ASSESSMENT UNDER THE BSP MODEL

Alexandros V. Gerbessiotis and Fabrizio Petrini

PRG-TR-03-98

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Network Performance Assessment under the BSP Model*

Alexandros V. Gerbessiotis and Fabrizio Petrini

Computing Laboratory,
Oxford University,
Wolfson Building,
Oxford, OX1 3QD,
UK.

April 1998

Abstract

The BSP model by L.G. Valiant has been proposed as a unifying and bridging model for the design, analysis and programming of general purpose parallel computing systems. A number of libraries have been implemented that allow programming following the BSP paradigm with one of them being the Oxford BSP Toolset. Algorithm designers and software engineers are able to study the performance of their implementations under the BSP model on a particular hardware platform and predict its behavior in others if the parameters of the BSP model are available for these machines. The assessment of the communication and synchronization requirements poses a serious challenge. As most machines were not designed and built as BSP computers, the study of their behavior as BSP computers is an interesting exercise and the experimental measurement of the BSP parameters p, L and g raises challenging questions.

In this work we study the network performance of three parallel hardware platforms (a Cray T3D, an IBM SP2 and an SGI Power Challenge) under the BSP model when *BSPlib* is used to realize the BSP programming paradigm. This study is, to the best of our knowledge, the first exhaustive experimental work that tries to model realistic parallel machines as BSP computers. We show that simplifications in the original BSP cost model may lead to inconsistencies and errors in performance estimation. We provide evidence that more exhaustive tests are needed than the ones currently used, to obtain reliable and consistent estimates of L and g.

*The work of the first author was supported in part by EPSRC(UK) under grant GR/K16999. The second author is sponsored by a Marie Curie Fellowship Contract No ERBFMBICT972076

1 Introduction

Since the introduction and adaptation of the von-Neumann model for sequential computing, the effects of computer revolution have been pretty significant. A general purpose computer performs well on computer programs written on a variety of standard programming languages like C, Fortran, or PASCAL, where the same program can be easily ported on other platforms. It has always been realized that parallel computers will eventually supersede sequential ones. This has yet to be realized despite advances in computer technology and the fact that chip technology seems to have reached physical limitations. Nowadays, fast machines are not much faster than the slowest ones, which, on their turn, are as fast as a supercomputer of twenty years ago. Despite these shortcomings of sequential computing, there has been no significant spread of use of parallel computers and few companies have realized that their future may rely on parallel platforms. The main reason for this has been that the majority of parallel machines built so far do not allow programmers to write programs that are efficient, portable and scalable; such objectives are only feasible by fine-tuning parallel code and taking into account minute details of the underlying architecture. Parallel algorithm design, on the other hand, ignores communication and/or synchronization issues (the PRAM model [10] is a prime example) and works only under unlimited parallelism assumptions. Many algorithms are designed having in mind factors not present in any reasonable machine, such as zero communication delay or infinite bandwidth.

Recently, the introduction of realistic parallel computer models such as the Bulk-Synchronous Parallel (BSP) model of computation by L.G. Valiant [35, 36], the Postal Model [3], the Tau Model [33] and LogP [7] come to address these limitations of parallel computing. The *Bulk-Synchronous Parallel* (BSP) model of computation has been proposed as a unified framework for the design, analysis and programming of general purpose parallel computing systems. It allows the design of algorithms that are both scalable and portable. The BSP model, as described in [35], consists of three parts:

(1) a collection of *processor-memory components*,

(2) a *communication network* that can deliver messages point-to-point among the components, and

(3) a *facility for globally synchronizing*, in barrier style, all or a subset of the components.

Computation on the BSP model proceeds in a succession of *supersteps*. A superstep may be thought of as a segment of computation during which each processor performs a given task using data already available locally before the start of the superstep. Such a task may include (i) local computations, (ii) message transmissions, and (iii) message receipts. It should be noted that, although the model stresses global barrier-style synchronization, pairs of processing units may synchronize pairwise by sending messages to and from an agreed memory location. However, such message exchanges should respect the superstep rules [12].

The tuple (p, L, g) characterizes the behavior and performance of a BSP computer. Parameter p is the number of components available, L is the *synchronization periodicity*, the minimum time between successive synchronization operations, and g is the ratio of the total throughput of the whole system in terms of basic computational operations, to the throughput of the communication network in terms of words of information delivered.

1.1 Parameter L

The interconnection network, the router and the synchronizer set lower bounds on the minimal value that L can attain.

(1) A lower bound on the value of L is the time for a remote memory operation/message dispatch to become effective and is thus dependent on the properties of the router and the interconnection network.

(2) A lower bound on the effective value of L is also the time for barrier synchronization. It is erroneous and misleading to think that L is just the cost of barrier synchronization.

(3) Parameter L is also made large enough for the theoretical bound on g to be realizable, as it is explained in more detail in the following section.

1.2 Parameter g

The value of g is measured while the network is in a steady state of continuous message usage. In the BSP model, when we refer to time we use the abstract notion of a *time-step*, as opposed to a CPU instruction or cycle. As current microprocessors may execute more than one instructions at a time, we use the term time-step to refer to the time needed to perform an elementary local computational operation (such as a fetch from memory and a floating-point operation followed by a store operation, or a comparison and exchange). The definition of g in the BSP model relates to the routing of an h-relation, that is, the situation where each processor sends or receives at most h messages; in practice this is equivalent to having each processor send or receive at most h words of information. Therefore, in the BSP model, g is the cost of communication so that an h-relation is realized within gh time steps, for any h such that $h \geq h_0$, where h_0 is a machine dependent parameter. Otherwise, the cost of communication is L. In conclusion the cost of routing an h-relation is $\max\{L, gh\}$. L is thus not just the cost of barrier synchronization as latency issues associated with routing a small number of messages (or small-size messages) are taken into consideration in the value of parameter L.

As we have already mentioned, the h of an h-relation relates to the total size of communicated data and g is thus expressed in terms of basic local computational operations (sometimes, floating-point operations) or absolute time (seconds) per data-unit (a byte or word of information). If g is expressed in terms of basic computational operations it is naturally $g \geq 1$.

In [4] the BSP model is augmented to take into consideration message size by introducing block parameter B; in the resulting BSP* model, if an h-relation is routed with each message of maximum message size s then a cost of $h \lceil s/B \rceil g$ is assigned to this communication or L, whichever is largest.

In practice, the parameter L of the BSP model not only hides the cost of communication when each processor sends/receives a small amount of information to/from few processors but also the cost of communication where each processor sends to/receives from a larger number of processors a small amount of information, or sends to/receives from few processors a larger amount of data.

In general, for any BSP computer, the values of L and g are likely to be non-decreasing functions of p.

1.3 Charging for a superstep

A cost of $\max\{L, x + gh\}$ time-steps is thus assigned to a superstep S, where x is the maximum number of basic computational operations executed by any processor during S, and h is the maximum number of messages sent or received by any processor. If h_s is the maximum number of messages sent and h_r is the maximum number of messages received by any processor during S, then $h = \max\{h_s, h_r\}$. One could also have taken $h = h_s + h_r$. There are other alternative ways to charge for superstep S and each one differs from any other one by a small constant factor. A list of such alternative costs is given below.

(1) $\max\{L, x, gh\}$ (original [35]),

(2) $\max\{L, x + gh\}$ (another alternative, widely used [12]),

(3) $\max\{L, x + g(h_r + h_s)\}$,

(4) $L + x + gh$ (another widely used alternative [21, 31]).

The cost model utilized in this study is the second one. One could have also used the fourth one if L were modeled the way we propose in the following sections and not the way it is being currently used.

1.4 BSP Program Design

Under the BSP programming paradigm the objective is to minimize parallel computation and communication time, minimize the number of supersteps and maximize their size, and increase processor utilization.

The ideal parallel machine would be one with small L and g parameters, that can maintain these values when the number of processors scales up. A PRAM machine follows this description but so far no scalable PRAM machine with these attributes has ever been built. Computational intensive problems, however, with few supersteps and relatively small communication (say, matrix multiplication) exhibit good behavior when executed in machines with relatively high L and g (for fixed matrix size). It would make sense to invest on a cheaper machine with higher L and g than buy the ideal machine if the intended purpose is to run such applications. On the other hand problems with significant synchronization and communication requirements (graph problems, sparse matrix computations) would benefit from machines with small L and g.

The behavior of a particular parallel program depends on both its computational, synchronization and communication requirements. It is not enough, for example, to try to maintain a small constant ratio L/g if the actual values of L and g are high (ethernet is such an example), and the size of the test problem is small enough that computation time is comparable to communication and synchronization time.

2 This Work

The BSP model is not just an architectural-oriented theoretical model. It can also serve as a paradigm for programming parallel computers. The fundamental concept introduced by the BSP is the notion of the superstep, and that all remote memory accesses occur between supersteps as part of a global operation among the processors. The results of these accesses become effective at the end of the current superstep. The BSP model has been realized as a library of functions for process creation and destruction, remote memory access and message passing, and global, barrier-style, synchronization [20, 26, 19]. The abstraction offered by the BSP is such that any library offering such facilities can be used for programming according to the BSP programming paradigm. The Oxford BSP Library [26], the Green BSP library [20] and the the Oxford BSP Toolset [19, 21] are some of the libraries that specifically allow programming according to the BSP paradigm.

In this work we assess the communication and synchronization performance under the BSP model and *BSPlib*, Version 1.3, of three parallel hardware platforms: a shared-memory machine, a Silicon Graphics Power Challenge and two distributed-memory machines, an IBM SP2 system and a Cray T3D. The interconnection network of the Cray T3D is a 3-dimensional torus. Because of its limited processor size the IBM SP2 allows for direct communication of every pair of processors.

For the combination of each of these machines and *BSPlib* we study how well each one of the machines behaves as a BSP computer. We study the performance of each of these machines for various communication patterns; some of the patterns are used in benchmark tests to stress the interconnection network, while others simulate patterns of communication that appear in practical applications. We also study how efficiently synchronization in barrier style can be performed by these machines within the BSP framework. Finally some representative values of the BSP parameters L and g are reported for various machine configurations and the invoked version of *BSPlib*.

Candidate values for L and g for various machines that include the ones of this study have been proposed before [22, 31]. We believe these values do not reflect reality for various reasons: (a) the experiments run to obtain the advertised values for L and g underestimate their true values, and (b) the suggested values of the BSP parameters may lead to inconsistencies when they are used in conjunction with the BSP cost model (eg. underestimate the cost of performing an h-relation). We provide more details in the following discussion.

The source code of our experiments will be made available in the first author's Web page. Our test programs are written in standard ANSI C; only recompilation of the source code is required to execute it on the hardware platforms utilized. The manufacturersupplied C compiler (`cc`) is used, and the source-code is compiled with the `-O3` compiler option set. The results depicted in the following tables and figures are in general averages over 100 experiments and discussed in later sections of this work. Timing is obtained through the use of the real-time clock function `bsp_time()` (it provides wall-clock time) of *BSPlib* [19].

3 Estimating BSP parameter L

The estimation of the value of L on any parallel machine is quite complicated as the value of parameter L is determined by *the cost of barrier synchronization*, *latency related costs* in communication and finally, and most importantly, is being made large enough for the value of the parameter g to make sense (so that one can claim a cost of gh for an h-relation).

Sometimes, parameter L (written also as l) is only associated with the cost of barrier synchronization. Under such an assumption the cost of routing an h-relation in a superstep is $L + gh$, where the second term gives the cost of actual communication and L the cost of barrier synchronization. The advertised values of L for various machines as reported in [22, 31] reflect this assumption. Tables 1 and 2 provide for the machines of our experiments the corresponding BSP parameters as they were reported in [22, 31]; all timings are in μsec. The value of g for the Cray T3D in Tables 1 and 2 is normalized to refer to 64-bit words (the word size on a Cray T3D); in [22, 31] it refers to 32bit words (normalized). The reported values in [22, 31] were obtained by running a program (`bsp_probe.lc` of *BSPlib*) which executes a piece of code like the one depicted in Figure 1. Line 3 in BARRIER_TIME invokes the synchronization primitive of *BSPlib*.

Machine	*Parameter*	*Processor Configurations*					
		4	8	16	32	64	128
T3D	L(μsec)	13.9	14.4	14.9	16.6	12.3	24.9
	g(μsec/word)	0.14	0.14	0.16	0.24	0.28	0.30

Table 1: Reported values of BSP parameters for the Cray T3D [22, 31]

Machine	*Parameter*	*Processors*
		4
SGI	L(μsec)	25.7
	g(μsec/word)	0.13

Machine	*Parameter*	*Processors*
		4
IBM SP2	L(μsec)	137.8
	g(μsec/word)	0.31

Table 2: Reported values of BSP parameters on SGI Power Challenge and IBM SP2 [22, 31]

The values of BSP parameters reported in Tables 1 and 2 lead to inconsistencies when used in BSP cost modeling as we shall describe below. They affect problems or programs that are mainly synchronization sensitive. We suggest a set of tests that support these claims and provide an alternative and more reliable way to derive the value of L if a

```
begin   BARRIER_TIME (runs)
/* Estimate time for barrier synchronization */
1.    for(i = 0; i < runs; i++)
2.        begin
3.            bsp_sync();
4.        end
end BARRIER_TIME
```

Figure 1: Primitive Barrier-synchronization time estimation test ([22])

particular machine is viewed as a BSP computer. We propose five tests in order to estimate the value of L. These tests are described in more detail below.

(1) The EMPTY test. This is the test shown in program BARRIER_TIME; it should report timings similar (but for changes in *BSPlib*) to those depicted in Tables 1 and 2 for the same machines.

(2) The COMP test. This is BARRIER_TIME with an elementary computation (like incrementing the contents of a memory location) inserted between steps 2 and 3 of BARRIER_TIME.

(3) The FULL test. In BARRIER_TIME, between steps 2 and 3, a total exchange is performed (each processor sends to every other processor an integer).

(4) The SIMPLE test. In BARRIER_TIME, between steps 2 and 3, processor 0 send processor $p-1$ an integer.

(5) The SCATTER test. In BARRIER_TIME, between steps 2 and 3, processor 0 sends an integer to every other processor.

In Table 3 we report the timing results (in μsec) obtained for these five tests on the three platforms. The reported timings are averages over a number of runs (100) and in each run the maximum time recorded by any of the p processors is used. The results of test EMPTY are comparable to those depicted in Tables 1 and 2.

Machine	*Processors*	*Empty*	*Comp*	*Simple*	*Scatter*	*Full*
T3D	4	17	18	33	46	47
T3D	8	25	27	42	73	78
T3D	16	40	43	53	120	130
T3D	32	51	51	60	147	175
T3D	64	93	93	107	312	364
T3D	128	182	183	204	651	762
SGI	4	26	27	43	66	109
IBM SP2	4	199	207	282	414	680

Table 3: Experimental Results to estimate bounds on L

We explain below the inconsistencies related to the values of the BSP parameters as reported in Tables 1 and 2. If the time for barrier synchronization (and thus the time of test EMPTY) were an accurate estimator of the BSP parameter L, then the BSP cost of SIMPLE would have been bounded above by $L + g$, for the BSP parameters to make sense. Similarly, the BSP cost of SCATTER and FULL would have been bounded above by $L + gp$. It is clear that this is not the case as it can be concluded from Tables 1, 2 and 3 (the choice of g from these tables or from the graphs of the following section does not make any difference). Actual timings are much larger than the predicted ones by a factor as large as five.

It is thus evident that the choice of the time of test EMPTY for the value of L leads to inconsistencies. The time of test SIMPLE, provided that the time for test FULL makes sense for the chosen value of g, or the time of test FULL seems to be a more reliable alternative for a first approximation of L.

The value of L we suggest also depends on the value of g that will be reported for each machine configuration. After taking into consideration the results related to the estimation of g we propose values for the tuple (p, L, g) for the various machine configurations tested.

Few observations need to be addressed. Library *BSPlib* supports two modes of communication: *buffered* for Direct Remote Memory Access (DRMA) [19] and a *high performance unbuffered* for DRMA. The reported values of L and g reflect high performance unbuffered communication. L is expressed in μsec and g in μsec per word; a word is 32-bits for the IBM SP2 and the SGI Power Challenge and 64-bits for the Cray T3D (in accordance to the layout of Tables 1 and 2). The proposed value for L is much larger than the cost of barrier synchronization, for the value of g to make sense.

If a higher value of g is suggested than the asymptotic one observed when large-size communication is performed (we shall refer to such a g as g_∞) then the proposed L can be kept relatively small. If one suggests a smaller value for g (closer to g_∞), the proposed L is larger, for the chosen g to make sense.

In Tables 4 and 5 below proposed values for the tuple (p, L, g) for each machine configuration are depicted. In addition, the value of g_∞ is also shown. As the BSP model only claims *upper-bound* estimates on communication, the proposed value of g may overestimate large-size communication (communication intensive parallel programs). For such communication, the g_∞ might have been more suitable. The value of g_∞ should not be used, however, as the estimator of g for any configuration as it underestimates small-size communication. This leads to various observations. One should not expect, in general, accurate time prediction from the BSP cost model, at least for the current machines in use. This is because BSP prediction is quite reliable for balanced communication patterns (total-exchange) whereas it provides upper bounds for unbalanced communication patterns (as it is also shown in the following section of our study). In practice, however, the BSP cost model is being used for this purpose; our experiments show that this may be difficult to achieve reliably. In order to achieve more reliable time prediction for a particular machine configuration one may need to carefully switch between g and g_∞ depending on the properties of his program or/and augment the BSP cost model to distinguish between the major communication patterns that appear in applications (eg total-exchange, scatter/gather communication).

Tables 4 and 5 depict our estimates for the BSP parameters for high performance unbuffered communication. The values of L and g preceded by the word "proposed" are the ones we suggest as the BSP values for the particular machine configuration. The asymptotic value g_∞ of g is also shown for one to compare it withe the proposed one.

Cray T3D							
Comment	*Parameter*	*Processor Configurations*					
		4	8	16	32	64	128
proposed	L	72	90	130	250	364	762
proposed	g	0.30	0.43	0.55	0.50	0.85	0.92
	g_∞	0.12	0.17	0.21	0.26	0.28	0.34

Table 4: Proposed values of BSP parameters for the Cray T3D

SGI Power Challenge		
Comment	*Parameter*	*Processors*
		4
proposed	L	90
proposed	g	0.55
	g_∞	0.12

IBM SP2		
Comment	*Parameter*	*Processors*
		4
proposed	L	700
proposed	g	1.5
	g_∞	0.25

Table 5: Proposed values of BSP parameters for SGI Power Challenge and IBM SP2

As it is shown in Tables 4 and 5, the ratio g/g_∞ is low (2-3) for a Cray T3D. It grows larger for the SGI and IBM SP2. We note that the small machine configurations in the case of the SGI and IBM SP2 do not allow any general conclusions to be drawn.

4 Estimating BSP parameter g

In this section we analyze the permeability of the three parallel machines. The study is conducted using different types of communication benchmarks, that range form very simple patterns that distribute or collect data inside the network to more dense and irregular collective patterns.

The IBM SP2 is a distributed memory machine interconnected by a high performance switch. The SGI is a cache-coherent shared memory multiprocessor that uses a shared bus as interconnection fabric. The Cray T3D is a scalable parallel machine that features high performance wormhole routing switches [8] interconnected as a three-dimensional torus. The Cray T3D uses a standard deterministic routing algorithm that avoids deadlocks on the wrap around connections mapping two sets of virtual channels on each physical link [30].

4.1 Traffic patterns

We will consider the following traffic patterns: scatter and gather, total exchange, perturbed h-relation and four permutations that are usually generated by numerical applications. For each pattern we compute the effective value of g by increasing the grain size of the communication.

In the *scatter* a single processor distributes to all processors a distinct token of information of the same size. Symmetrically, the *gather* collects information from all processors. These two patterns often occur in the initial and final part of a parallel computation and share the important characteristic that they do not generate conflicts inside the network, so they represent the simplest case of collective communication. In fact, in both patterns the packets are routed through distribution trees determined by the deterministic routing algorithm.

The all-to-all personalized communication, or simply *total-exchange*, is an important communication pattern that is at the heart of many applications, such as matrix transposition and the Fast Fourier transform (FFT). The total-exchange is a collective communication pattern where every processor sends a distinct message to every other processor. The efficient implementation of the total-exchange has been extensively studied in a variety of networks [24] [6] [23] [32] [34] [29] [28]. These studies are motivated by several applications in the field of scientific computing [25]. In contrast with the scatter and the gather, the total exchange is a dense communication pattern and stresses the bisection bandwidth of the network.

The *perturbed h-relation* is a pattern of communication similar to the total exchange. Each processor sends a message to every other processor such that the total size of information sent by any processor is fixed to h words. The size of each message is a random variable that takes values between $h/(2p)$ and $3h/(2p)$. This way, each processor may receive up to $3h/2$ words. This means that an $3h/2$-relation may be communicated when a *perturbed h-relation* is realized. This pattern of communication appears in deterministic and randomized BSP sorting [13, 17].

The experimental evaluation is completed by four permutation patterns, where each processor sends a fixed amount of information to a single destination. These patterns are often difficult to route because they stress the routing algorithm creating hot spots inside the network. To describe these traffic patterns, let the binary representation of a processors be $a_0 a_1 \ldots a_{n-1}$. Also, let $\overline{0} = 1$ and $\overline{1} = 0$. The traffic patterns can be defined as follows.

- *Complement traffic.* Each processor sends a message to the destination given by $\overline{a_0 a_1 \ldots a_{n-1}}$.
- *Bit reversal.* Each processor sends a message to the destination given by $a_{n-1} \ldots a_1 a_0$.
- *Shuffle.* Processor $a_0 a_1 \ldots a_{n-1}$ communicates with processor $a_1 \ldots a_{n-1} a_0$ (rotate left one bit).
- *Butterfly.* Processor $a_0 a_1 \ldots a_{n-2} a_{n-1}$ communicates with processor $a_{n-1} a_1 \ldots a_{n-2} a_0$ (exchange the most and least significant bits).

Most graphs shown in the following sections study the value of g for a given set of traffic patterns and report the maximum amount of information exchanged, expressed in words, on the x axis and the value of the permeability g, expressed in μsec/word, on the y axis. Both axes use a logarithmic scale. In order to obtain a more stable sample, each single value in the plots is the average of 100 distinct runs. To minimize the buffering problems, we used the high performance, unbuffered communication primitives of *BSPlib*.

4.2 IBM SP2

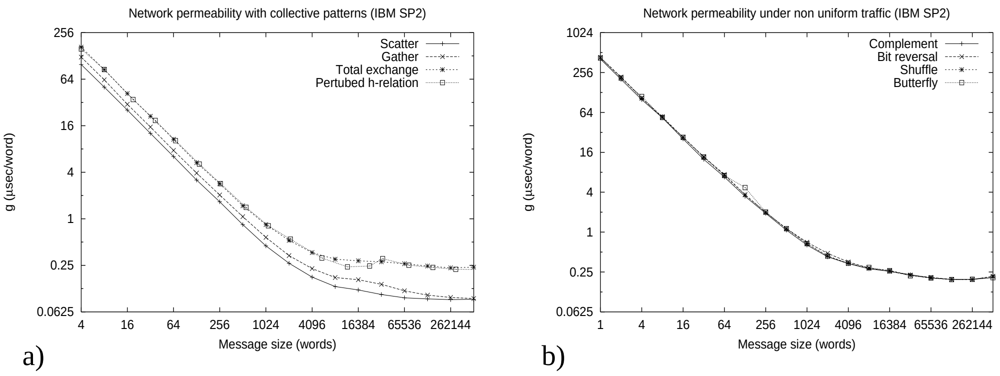


Figure 2: Experimental results for the IBM SP2.

We first tested an IBM SP2 with 4 processors. In Figure 2 we can see that the communication overhead is the main factor when the maximum amount of information transmitted is less than 4096 words. The asymptotic g for the scatter and gather is about 0.09 μsec/word, while for all other patterns is 0.25 μsec/word.

4.3 SGI Power Challenge

The same set of results is shown in Figure 3 for a SGI Power Challenge with 4 processors. As in the previous example, the effective g of scatter and gather is small, while all the other patterns converge to the same asymptotic value of 0.125 μsec/word. The slight increase of the value of g for the total exchange and the perturbed h-relation in the final part of the plot is determined by caching problems. In fact the data set tends to overflow the L2 cache when the value of h exceeds 65536 words.

4.4 Cray T3D

The results for the Cray T3D are shown in Figure 4 for 4, 8 and 16 processors, and the results for 32, 64 and 128 processors are shown in Figure 5. The network topology of the Cray T3D is a three-dimensional torus, as is the topology of any subset of the network allocated to run a parallel task. Smaller subsets, however, cannot use the wrap-around

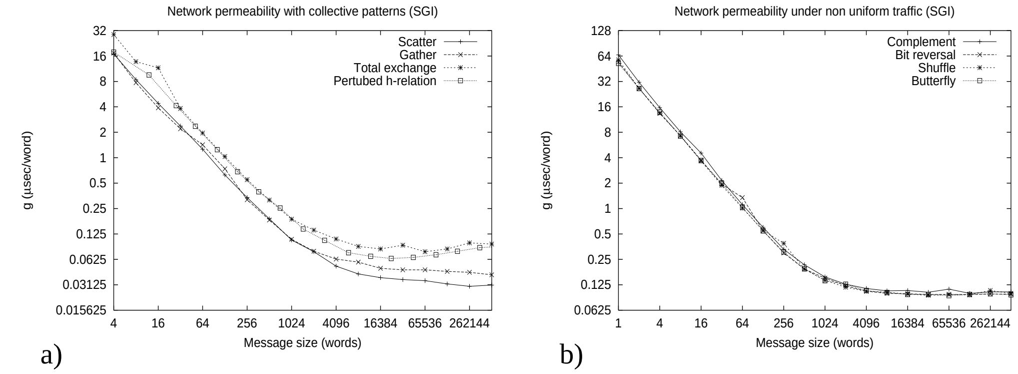


Figure 3: Experimental results for the SGI Power Challenge.

connections so, in the general case, we are going to consider a three-dimensional grid rather than a torus.

In all the machine configurations the value of g is stable when the message size is larger than 4096 words. The total exchange and the perturbed h-relation have a similar behavior with the same asymptotic values of g, as scatter and gather.

In most cases the network performance is largely insensitive to the type of permutation. The only exception, reported in Figure 4 b), is the complement traffic with 4 processors. This can be explained by noting that in the bit reversal, shuffle and complement traffic only two of the four processors generate remote communication (for example in the bit reversal two processors have palindrome bit strings as identifiers), while in the complement all processors send remote messages. It is worth noting, looking at Figure 2 b) and 3 b), that in the IBM SP2 and in the SGI with same number of processors the network performance is the same with all permutations. This means that, for this configuration and traffic pattern, the Cray T3D is limited by the network performance while the dominating factor in the other two machines is the local overhead.

As in the previous cases, we can note a clear separation between the scatter and gather and the other communication patterns, the total exchange, the perturbed h-relation and the permutations.

More insight on the network performance is provided in Figure 6, where we can see the asymptotic values of g for all the machine configurations and traffic patterns. We can highlight the following aspects.

1. The asymptotic values for scatter and gather are not sensitive to the number of processors and are, in practice, constants. The current implementation of *BSPlib* slightly favors gather over scatter.

2. An important characteristic of *BSPlib* is that it evens out all the remaining traffic patterns that are dense and congestion-prone. We can see that the asymptotic g is approximately the same for any machine configuration with more than 4 processors. The results for 16, 32 and 128 processors are all clustered in a very short interval.

Also, the value of g approximately scales as the network bisection bandwidth.

3. The communication time of a perturbed h-relation is less than the communication time of the total-exchange with the same h. Skewed communication usually experiences less congestion.

4. The determining factor in communication time is the maximum amount of information sent or received rather than the number of messages. For example, in the permutation patterns each processor sends a single message while in the total-exchange the same amount of information is sent in p different chunks, where p is the number of processors. The communication time of both patterns is similar.

BSPlib achieves these results using different techniques and optimizations. First of all, the randomized allocation of processes to processors smoothes the characteristics of both interconnection network and communication pattern. It is well known that non uniform traffic patterns, as the ones analyzed in this study, can severely degrade the network performance in the presence of deterministic routing [25] [9]. Many current machines are still limited by the injection and reception overhead. *BSPlib* carefully schedules the message transmission and optimizes the buffer allocation by properly combining messages.

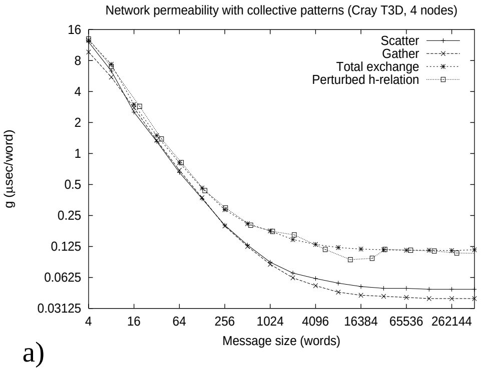
In Figure 7 we can see the asymptotic values obtained using a stripped version of *BSPlib* that directly interfaces to the shared memory library of the Cray T3D. It does not randomize processor allocation, does not combine messages and does not resolve contention at the receiving ends. We can observe that we get widely different results. Some permutations, as the butterfly, perform better but others get worse, as the total exchange and the perturbed h-relation. All in all, we lose the performance predictability without getting a real performance advantage.

Based on these preliminary results, we can distinguish two classes of communication patterns. Those that generate congestion-free communication and the other patterns that, in some form, generate conflicts in the network or at the network interfaces. We argue that most communication, in real parallel applications, falls in one of the two classes and that performance in each class can be estimated easily with good approximation and little variance.

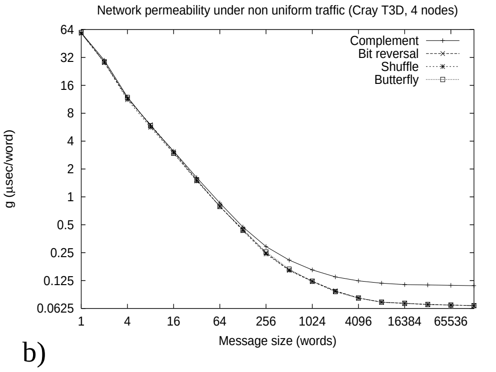
5 Conclusion

We examined the communication and synchronization performance under the BSP model of three parallel machines and studied how well each one of the machines behaved as a BSP computer. In particular, we examined the performance of each of these machines for various communication patterns. We also studied how efficiently synchronization in barrier style can be performed by these machines within the BSP framework. Finally some representative values of the BSP parameters L and g were reported for various machine configurations.

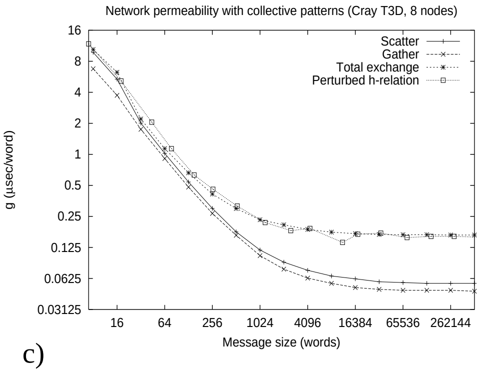
We also showed that simplifications in the original BSP cost model may lead to inconsistencies and errors in performance estimation. We provided evidence that more



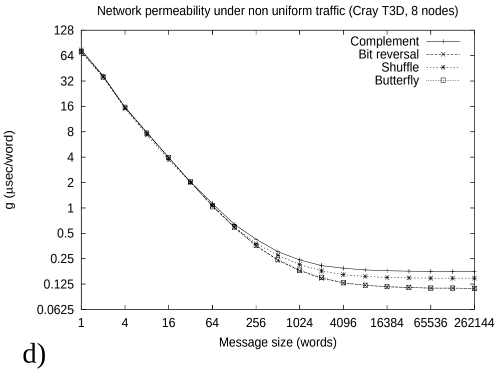
a)



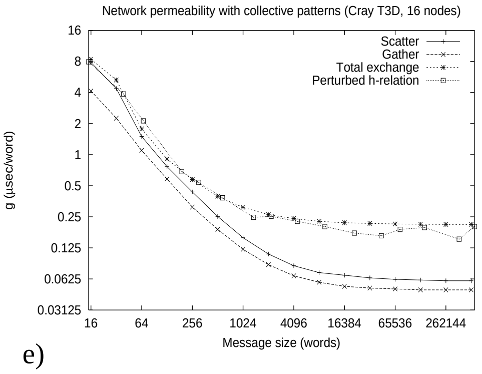
b)



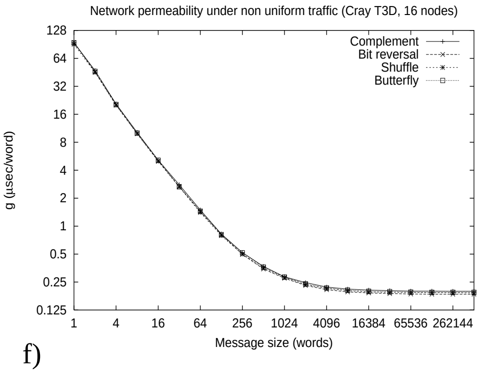
c)



d)



e)



f)

Figure 4: Experimental results for the Cray T3D. Configurations with 4, 8 and 16 processors.

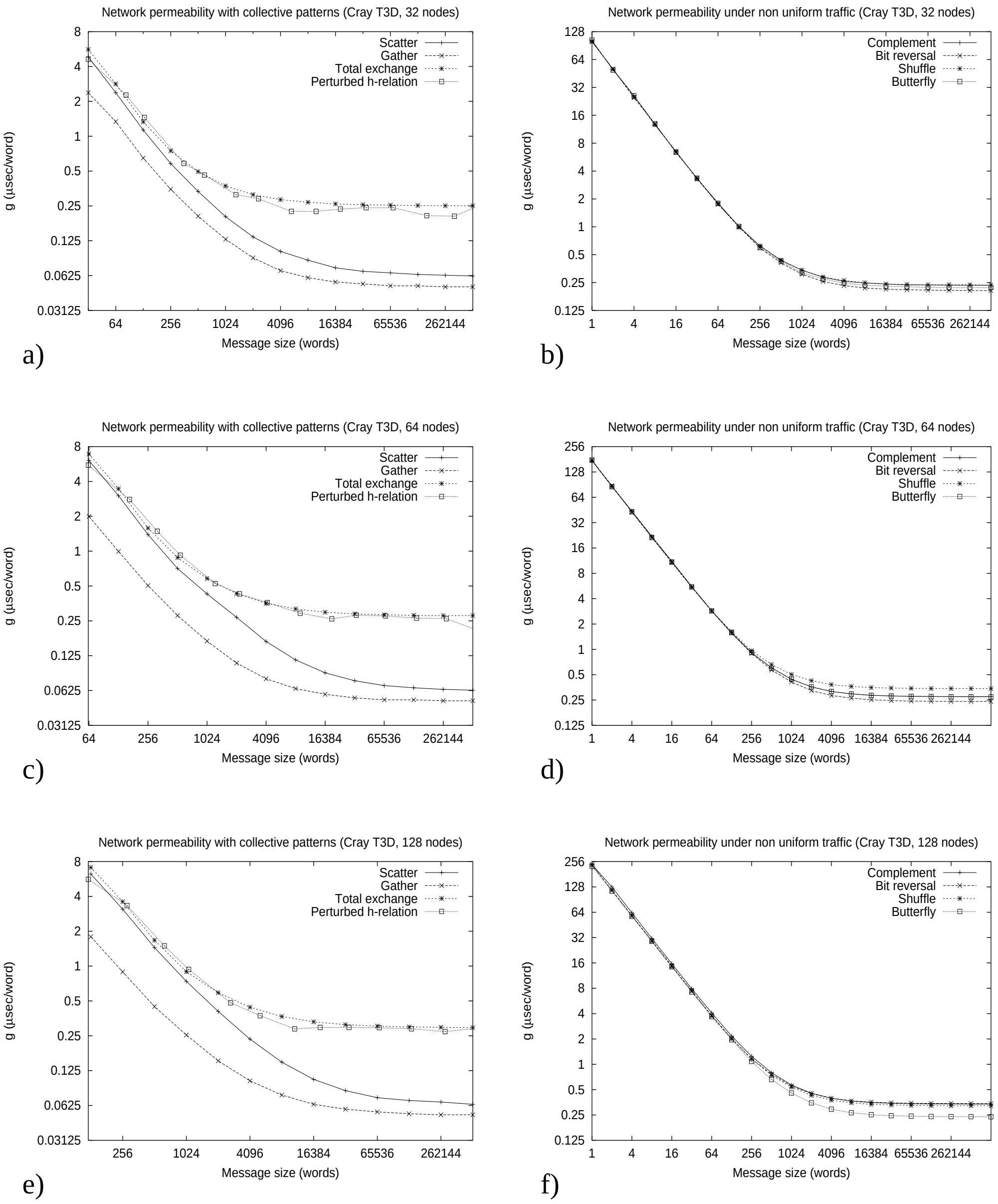


Figure 5: Experimental results for the Cray T3D. Configurations with 32, 64 and 128 processors.

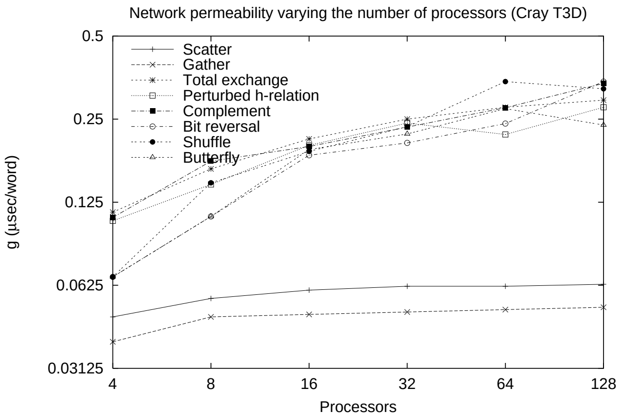


Figure 6: Asymptotic values of g on the Cray T3D, for all machine configurations and traffic patterns.

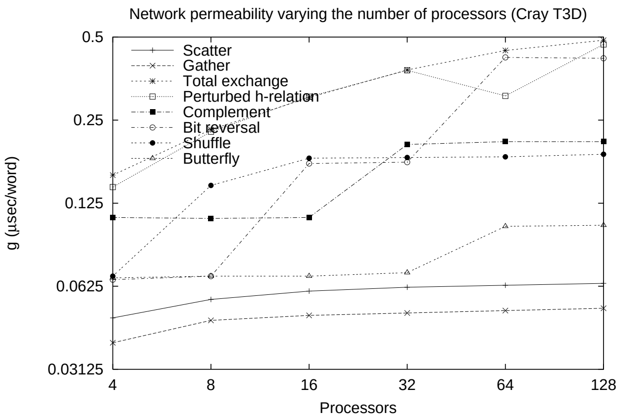


Figure 7: Asymptotic values of g on the Cray T3D using an unoptimized version of *BSPlib*, for all machine configurations and traffic patterns.

exhaustive tests are needed than the ones currently used to obtain reliable and consistent estimates of L and g.

In [11] an experimental study of dense matrix computations on the BSP model was undertaken with the purpose of examining whether good algorithm performance prediction was possible under the BSP model for such problems. Not only this was confirmed but in many instances one could get reliable prediction of the absolute (time) performance of these algorithms on various machines. In particular, the efficiency of the examined algorithms as predicted by the BSP cost model was very close to the observed efficiency during the experiments. The test platforms were the ones used in this study and the values of the BSP parameters utilized in that study are identical to the proposed ones in this work.

One should not expect accurate time prediction from the BSP cost model, at least for the current machines in use. This is because BSP prediction is quite reliable for balanced communication patterns (total-exchange) whereas it provides upper bounds for unbalanced communication patters (as it is also shown in the following section of our study). In practice, however, the BSP cost model is being used for this purpose; our experiments show that this may be difficult to achieve reliably. In order to achieve more reliable time prediction for a particular machine configuration one may need to carefully switch between g and g_∞ depending on the properties of his program or/and augment the BSP cost modeling to distinguish between the major communication patterns that appear in applications (eg total-exchange, scatter/gather communication).

6 Acknowledgements

We would like to acknowledge the support of the Edinburgh Parallel Computing Center for granting us access to the Cray T3D at EPCC.

References

[1] Alok Aggarwal and Ashok K. Chandra and Marc Snir. On Communication Latency in PRAM Computation. Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, pages 11-21, 1989.

[2] Alok Aggarwal and Ashok K. Chandra and Marc Snir. Communication Complexity of PRAMs. Theoretical Computer Science, pages 3-28, 1990.

[3] Amotz Bar-Noy and Shlomo Kipnis. Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems. Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, pages 13-22, June 1992.

[4] A. Baumker, W. Dittrich, and F. Meyer auf der Heide. Truly efficient parallel algorithms: c-optimal multisearch for an extension of the BSP model. In *Proceedings of the Annual European Symposium on Algorithms*, 1995.

[5] R. H. Bisseling and W. F. McColl. Scientific computing on Bulk-Synchronous Parallel architectures. Preprint 836, Department of Mathematics, University of Utrecht, December 1993.

[6] J. Bruck, C. T. Ho, S. Kipnis, and D. Weathersby. Efficient Algorithms for All-to-All Communications in Multi-Port Message-Passing Systems. In *Proceedings of the 6th ACM Symposium on Parallel Architectures and Algorithms*, pages 298–309, 1994.

[7] D.E.Culler and R. Karp and D. Patterson and A. Sahay and K.E. Schauser and E. Santos and R. Subramonian and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, CA, May 1993.

[8] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[9] José Duato and Pedro López. Performance Evaluation of Adaptive Routing Algorithms for k-ary n-cubes. In Kevin Bolding and Lawrence Snyder, editors, *First International Workshop, PCRCW'94*, volume 853 of *LNCS*, pages 45–59, Seattle, Washington, USA, May 1994.

[10] S. Fortune and J. Willie. Parallelism in Random Access Machines Proceedings of the 10th Annual Symposium on Theory of Computing, pages 114–118, 1978.

[11] A. V. Gerbessiotis. Algorithmic and practical considerations for dense matrix computations on the BSP model. Technical Report PRG-TR-32-97, Computing Laboratory, Oxford University, October 1997 revised in March 1998.

[12] A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing,* 22:251-267, 1994.

[13] A. V. Gerbessiotis and C. J. Siniolakis. Deterministic sorting and randomized median finding on the BSP model. In *Proceedings of the 8-th ACM Symposium on Parallel Algorithms and Architectures*, Padova, Italy, June 1996. An extended version appeared as Efficient Deterministic Sorting on the BSP Model. Technical Report PRG-TR-19-96, Oxford University Computing Laboratory, October 1996.

[14] A. V. Gerbessiotis and C. J. Siniolakis. Primitive operations on the BSP model. Technical Report PRG-TR-23-96, Computing Laboratory, Oxford University, October 1996.

[15] A. V. Gerbessiotis and C. J. Siniolakis. Efficient Deterministic Sorting on the BSP Model. Technical Report PRG-TR-19-96, Oxford University Computing Laboratory, October 1996.

[16] A. V. Gerbessiotis and C. J. Siniolakis. A randomized sorting algorithm on the BSP model. In *Proceedings of the 11-th International Parallel Processing Symposium*, Geneva, Switzerland, April 1997.

[17] A. V. Gerbessiotis and C. J. Siniolakis. An Experimental Study of BSP Sorting Algorithms. In Proceedings of 6th Euromicro Workshop on Parallel and Distributed Processing, Madrid, Spain, January, 1998.

[18] A. V. Gerbessiotis. Web page: `http://www.comlab.ox.ac.uk/oucl/people/alex.gerbessiotis.html`

[19] M.W. Goudreau, J.M.D. Hill, K. Lang, W.F. McColl, S.D. Rao, D.C. Stefanescu, T. Suel, and T. Tsantilas. A proposal for a BSP Worldwide standard. BSP Worldwide, `http://www.bsp-worldwide.org/`, April 1996.

[20] M. Goudreau, K. Lang, S. Rao and T. Tsantilas. The Green BSP Library. Technical Report CR-TR-95-11, University of Central Florida, 1995.

[21] J. M. D. Hill and D. Skillicorn. Lessons learned from implementing BSP. To appear in High Performance Computing and Networking (HPCN'97), Lecture Notes in Computer Science, Springer-Verlag, April 1997.

[22] J. M. D. Hill. In `http://www.bsp-worldwide.org/implmnts/oxtool/`, September 1997.

[23] S. Hinrichs, C. Kosak, D O'Hallaron, T. M. Stricker, and R. Take. An Architecture for All-to-All Personalized Communication. In *Proceedings of the 6th ACM Symposium on Parallel Architectures and Algorithms*, pages 310–319, 1994.

[24] S. L. Johnsson and C. T. Ho. Optimal Broadcasting and Personalized Communication in Hypercubes. *IEEE Transactions on Computers*, 38:1249–1268, 1989.

[25] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, USA, 1992.

[26] R. Miller. A library for Bulk-Synchronous Parallel programming. In *Proceedings of the British Computer Society Parallel Processing Specialist Group Workshop on General Purpose Parallel Computing*, December 1993.

[27] C. H. Papadimitriou and M. Yannakakis. Towards an Architecture Independent Analysis of Parallel Algorithms Proceedings of the 20th Annual Symposium on Theory of Computing, pages 510-513,1988.

[28] Fabrizio Petrini. Total-Exchange on Wormhole k-ary n-cubes with Adaptive Routing. In *Proceedings of the 12th International Parallel Processing Symposium, IPPS'98*, Orlando, Florida, March 1998.

[29] Satish Rao, Torsten Suel, Thanasis Tsantilas, and Mark Goudreau. Efficient Communication Using Total-Exchange. In *Proceedings of the 9th International Parallel Processing Symposium, IPPS'94*, Santa Barbara, CA, April 1995.

[30] Steve Scott and Greg Thorson. Optimized Routing in the Cray T3D. In *First International Workshop, PCRCW'94*, volume 853 of *LNCS*, pages 281–294, Seattle, Washington, USA, May 1994.

[31] D. B. Skillicorn, J. M. D. Hill, and W. F. McColl. Questions and Answers about BSP. *Scientific Programming*, Vol 6, pp 249-274, 1997.

[32] Rajeev Thakur and Alok Choudary. All-to-All Communication on Meshes with Wormhole Routing. In *Proceedings of the 8th International Parallel Processing Symposium, IPPS'94*, pages 561–565, Cancun, Mexico, April 1994.

[33] Y-J Tsai and P.K.McKinley. The τ-Model: A Unified Communication Cost Model. 10th International Conference on Parallel and Distributed Computing Systems (PDCS-97), New Orleans, Louisiana, USA, October 1997.

[34] Yu-Chee Tseng and Sandeep K. S. Gupta. All-to-All Personalized Communication in a Wormhole-Routed Torus. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):498–505, May 1996.

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

[36] L. G. Valiant. General purpose parallel architectures. In *Handbook of Theoretical Computer Science,* (J. van Leeuwen, ed.), North Holland, 1990.