Experimental Validation of Parallel Computation Models on the Intel Paragon[*]

Ben H.H. Juurlink

Heinz Nixdorf Institute, Paderborn University
Fürstenallee 11, 33102 Paderborn, Germany
E-mail: benj@uni-paderborn.de

Abstract

Experimental data validating some of the proposed parallel computation models on the Intel Paragon is presented. This architecture is characterized by a large bandwidth and a relatively large startup cost of a message transmission, which makes it extremely important to employ bulk transfers. The models considered are the BSP model, in which it is assumed that all messages have a fixed short size, and the BPRAM, in which block transfers are rewarded.

1. Introduction

Experimental validation of models of parallel computation is extremely valuable. To emphasize this, Snyder[11] wrote: "[. . .] in the absence of evidence that it makes accurate predictions, what serious interest can there be in either the model, or algorithms developed with it?" Although some experimental results investigating the predictive capabilities of parallel models have meanwhile been published, the results were often unsatisfactory for the following reasons. First, sometimes the experimental data was collected on only one platform. In other cases, the results were gathered on systems with only a few processors. Third, often the communication overhead was only a small part of the total execution time. Since communication cost is the only cost captured in detail by most models, this provides no support for their predictive capabilities.

The Models. The models considered in this paper are the Bulk-Synchronous Parallel (BSP) model[13] and the Message-Passing Block PRAM (BPRAM)[1]. In this section, these models are briefly described.

The BSP model[13] consists of p processors, a communication medium that can deliver messages point-to-point, and a mechanism for barrier synchronization. BSP computations are organized in *supersteps*, separated by barrier synchronizations. In a superstep, a processor can perform local operations, and send and receive some messages. Two parameters are used to model communication cost: (1) the latency/synchronization cost L, and (2) the computational-to-communication throughput ratio g. Central to the BSP model is the concept of an h*-relation*: a communication pattern in which each processor sends and receives at most h messages. The parameters g and L are such that an arbitrary h-relation followed by a barrier synchronization takes $g \cdot h + L$ time. The cost of a superstep is therefore $w + g \cdot h + L$, where w is the maximum amount of local work performed by any processor, and h is the maximum number of messages sent or received by any processor.

In the BSP model, all messages have a fixed short size (essentially the word size of the machine). However, it is well-known that on many parallel architectures there is a large startup cost associated with a message transmission. The BPRAM[1] captures this by assuming that it takes $\sigma \cdot m + \ell$ time to transfer a message of length m, where ℓ denotes the startup cost of a message transmission and σ denotes the transfer time per byte. It is also assumed that a processor can send and receive at most one message in one communication step.

Of course, there are many other models which are not considered in this paper. The well-studied LogP model[4] is not considered because (1) on the Paragon, the overhead o dominates the gap g, so g can be ignored, and (2) the implemented BSP library sends messages in a staggered order. For the algorithms we experimented with, this technique was sufficient to ignore the capacity constraint of LogP.

Overview. In this paper, we extend the work commenced in [8] by presenting new experimental results collected on the Paragon. This architecture is characterized by a large bandwidth and a relatively large startup cost of a message transmission, which makes it important to employ bulk transfers. For this reason, the implemented BSP library postpones all communication until the end of the superstep and combines all packets destined for the same

[*]Supported in part by DFG-SFB 376 "Massive Parallelität" and by EU ESPRIT Long Term Research Project 20244 (ALCOM-IT).

processor into a single message. It was shown in [10] that this reduces the importance of sending large messages. Our results show, however, that on the Paragon this technique cannot reduce g to a value which is comparable to the reciprocal of the per-processor bandwidth. It is also shown that the BSP cost model overestimates the time required for performing permutation routing patterns. More seriously, it is shown that the execution time of an algorithm implementation can be improved by reducing the number of startups, even if this increases the communication volume as well as the number of barrier synchronizations. Clearly, this goes against the incentives provided by BSP. Finally, it is demonstrated that the BPRAM overestimates the time needed for routing unbalanced communication patterns.

Related Work. The BSP model was analyzed experimentally in [5]. We have the following concerns about the experimental data presented there. First, the authors noted that the BSP cost model should not be expected to predict precise running times, since this is only possible for fairly simple algorithms. We remark that the algorithms we experimented with (purposely) belong to the class of algorithms for which a precise analysis seems feasible. Second, the results were collected on platforms with a rather small number of processors. As will be shown, many phenomena that might affect the accuracy of the BSP model do not become visible until at least a moderate number of processors is employed. Finally, for most applications considered in [5], the communication overhead was only a small component of the total execution time. We are interested in cases where the communication overhead accounts for a significant part of the overall running time, since it is the only cost modeled in detail by BSP.

Organization. Section 2 describes the Paragon architecture and presents experimental results for some simple benchmark programs that are used to determine several system parameters. In Section 3 a BSP and a BPRAM library are described, and experimental data is presented that is used to determine the BSP and BPRAM parameters of the Paragon. Thereupon, in Section 4, performance measurements are presented for the same application kernels as used in [8]. In Section 5 we give our conclusions.

This paper is intended as a supplement to [8]. For more detailed descriptions of the models and of the implemented algorithms, the reader is referred to that paper. The full version of this paper is available as a technical report[7].

2. Experimental Platform

In this section, the Paragon architecture[3, 2] is briefly described, and some results are presented that are used to determine several system parameters of the Paragon.

Every node of the Paragon system used in this investigation consists of two 50 MHz, i860 XP microprocessors with 32MB of DRAM. One processor, called the *message processor*, is dedicated to communication, so that the *compute processor* is released from message passing operations. The nodes are connected to Mesh Routing Chips, which are arranged in a 2D mesh with 175MB/s links. The network employs dimension-ordered wormhole routing.

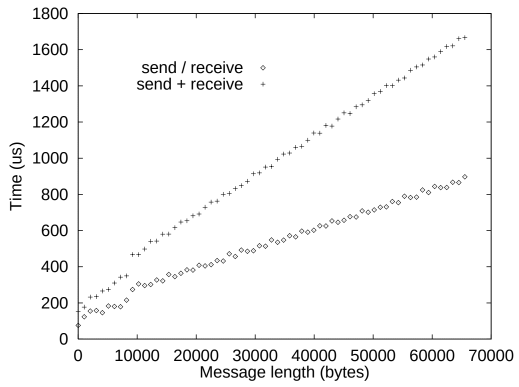

Figure 1. Time needed to send a message through an unloaded network.

We implemented several simple benchmark programs using the native NX message passing library. Fig. 1 shows the results of two experiments. In the first experiment, two processors repeatedly exchange a message between them (the well-known "ping-pong" (pp) benchmark). The results of this experiment are labeled "send/receive". In the second experiment, both processors send *and* receive a message simultaneously (the "ping-pong double" (pp2) benchmark). The results of this experiment are labeled "send + receive". It can be seen that the time taken by both experiments is well approximated by a straight line. By performing least squares fits to the measured data points, we found that the time needed to send a message of m bytes is given by:

$$T_{\text{pp}}(m) = t_{st} + t_b \cdot m \approx 146 + 0.0115 \cdot m \, \mu s, \qquad (1)$$
$$T_{\text{pp2}}(m) = t_{st} + t_b \cdot m \approx 216 + 0.0226 \cdot m \, \mu s. \qquad (2)$$

Thus, in the ping-pong benchmark the startup cost of a message transmission is roughly 146μs and the time per byte is about 11.5ns, which corresponds to an asymptotic bandwidth of 87MB/s. Half the achievable bandwidth is reached when the message size is about $m_{1/2} = 12.7$KB. The nominal bandwidth of 175MB/s is never reached, because NX splits messages into fixed size packets and there is a per-packet software overhead[9]. Other message passing interfaces for the Paragon, such as PAM[9], achieve higher bandwidth and lower latency than NX, but were not available. Also notice that it appears that a processor cannot send and receive simultaneously.

Eq. (1) is valid only if there is no congestion in the network. However, because messages are not injected into the

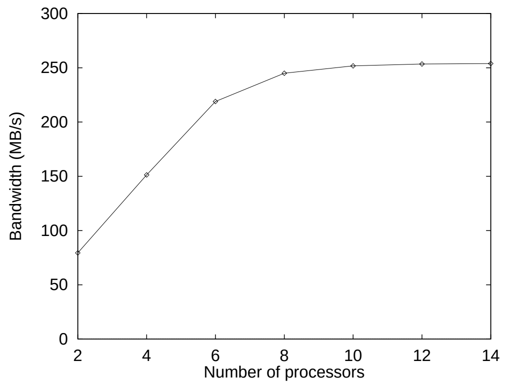

Figure 2. Effect of bandwidth limitations on a configuration of size $p \times 1$.

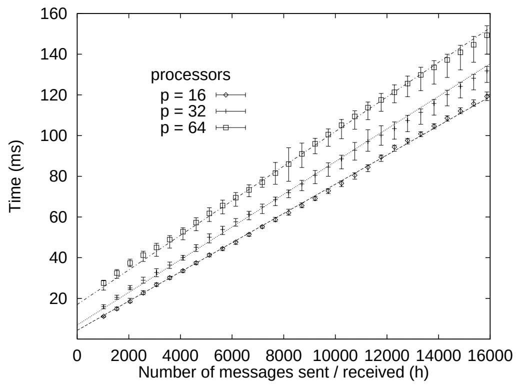

Figure 3. Time required for routing h-relations on the Paragon.

network at the full hardware transfer rate, bisection bandwidth limitations only play an important role when the network configuration is large. In other words, it appears that the network has *excess bandwidth*. To illustrate this, Fig. 2 shows the bandwidth achieved when each processor i sends a message of 64KB to processor $p - i - 1$ on a $p \times 1$ configuration. Bandwidth is defined as the total amount of data transferred over the link connecting processors $p/2 - 1$ and $p/2$ divided by the total time consumption. There are two regions of interest. When $p \leq 6$, the time consumption is dominated by the time needed to inject the message into the network. Because of this, the achieved bandwidth increases almost linearly with the number of processors. When $p \geq 8$, the bandwidth stays constant, because in this region the link connecting processors $p/2 - 1$ and $p/2$ forms a bottleneck.

3. Matching the Models to the Machine

In order to implement the algorithms in a BSP and BPRAM like fashion, we implemented two small communication libraries on top of the native NX message-passing library. In this section, these two libraries are briefly described, as well as the experiments conducted to determine the BSP and BPRAM parameters belonging to the Paragon.

BSP Library. The implemented BSP library is similar to the recently released BSPlib library[6]. It provides functions for Direct Remote Memory Access (DRMA), Bulk-Synchronous Message Passing (BSMP), and barrier synchronization.

In order to reduce the importance of sending large messages, the implemented BSP library postpones all communication until the end of a superstep and combines all packets destined for the same processor into a single message. This is done as follows. Each processor contains $p - 1$ output buffers in which the messages are written. Upon reaching a synchronization barrier, an all-to-all broadcast is executed in order to exchange information like buffer size,

Network configuration	p	g (in μs)	L (in μs)
4×2	8	6.50	3.1×10^3
4×4	16	7.18	4.3×10^3
8×4	32	8.03	6.9×10^3
6×6	36	8.10	7.4×10^3
8×8	64	8.55	17.0×10^3

Table 1. BSP parameters.

number of packets in the buffer, etc. This is implemented in a butterfly-like fashion requiring $\log p$ startups. After that, each processor allocates an input buffer large enough to store all incoming messages. Finally, the messages are transmitted in a staggered all-to-all exchange pattern.

Fig. 3 shows the time required for routing randomly generated full h-relations (using DRMA) on several configuration sizes. Because we believe that the BSP cost model does not penalize fine-grain communication, we used a packet size equal to the word size of the machine (4 bytes). The experimental data shows that the time required for routing h-relations is well approximated by a straight line with slope g and offset L. By fitting straight lines to the measured points, we obtained the BSP parameters summarized in Table 1.

Obviously, a naive implementation of the BSP library that sends messages as soon as they are generated would achieve a value of g comparable to the startup cost. So, the value of g is reduced by a factor of 17–22 (depending on the network size), but the network capacity is not the limiting factor. Instead, the bulk of the cost of sending a packet is due to buffering. To illustrate this, Fig. 4 breaks down the time required for routing h-relations on a 4×2 configuration into a fixed overhead part, a variable overhead part, and the time needed to send the messages through the net. The fixed overhead part includes the cost of $p - 1 + \log p$ startups, the time needed for buffer allocation and barrier synchroniza-

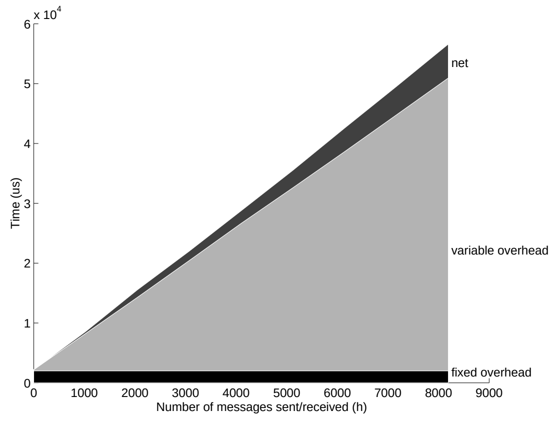

Figure 4. Breakdown of the time required for routing h-relations on a 4×2 configuration.

tion. The variable overhead part is the time needed to write the messages into the output queue and to read the messages from the input queue. This requires about 5.9μs per message. Sending the messages through the net takes only about 0.5μs each. Although we believe that the value of g can be improved, we do not believe that it can be reduced by much. This is borne out an implementation of active messages on the Paragon[9], which showed that storing data into memory shared by the compute and message processor causes write-throughs to memory and cache invalidations.

Another important point of Fig. 3 and Table 1 is that L includes the cost of $p-1$ startups. While g increases very slowly with the system size (since the transmission time is dominated by send and receive overheads), L increases at a rate roughly linear with the number of processors.

BPRAM Library. The implemented BPRAM library provides only one communication primitive that sends a message to a specified destination processor and receives a message from an arbitrary processor. In keeping with the BPRAM semantics, all processors must call this function simultaneously to prevent deadlock, and the communication pattern should correspond to a (partial) permutation.

Fig. 5 plots the time required for routing full block permutations as a function of the message length. By performing a least squares fit on the measurements, we obtained the BPRAM parameters given in Table 2. These parameters correspond closely to the system parameters of the Paragon. Eq. (2) shows that when a processor sends and receives simultaneously, σ is bounded from below by 22.6ns, which nearly matches the σ attained on a 4×2 configuration. Furthermore, σ grows slowly as the configuration size increases (because it appears that the Paragon has excess bandwidth), but increases more rapidly when the configuration becomes larger than 6×6. Also notice that ℓ stays almost constant, because the number of startups per communication step increases very slowly with the number of processors.

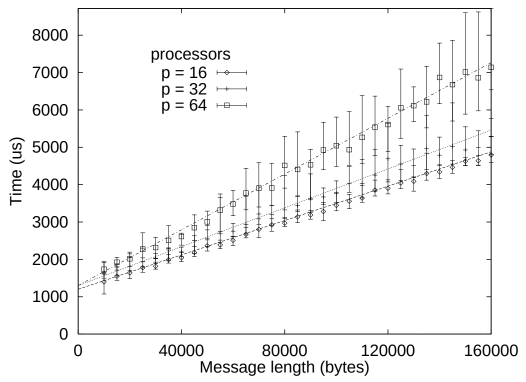

Figure 5. Time needed for routing full block permutations on the Paragon.

Network configuration	p	σ (in μs)	ℓ (in μs)
4×2	8	2.18×10^{-2}	1.0×10^3
4×4	16	2.30×10^{-2}	1.2×10^3
8×4	32	2.60×10^{-2}	1.3×10^3
6×6	36	2.60×10^{-2}	1.2×10^3
8×8	64	3.73×10^{-2}	1.3×10^3
10×10	100	5.20×10^{-2}	1.3×10^3

Table 2. BPRAM parameters.

4. Application Kernels

We implemented the same algorithms as used in [8]: matrix multiplication, bitonic sort, sample sort and all pairs shortest path. In this section, the performance results for these application kernels are presented.

Matrix Multiplication. Fig. 6 compares the performance achieved by the BSP matrix multiply with the performance attained by the BPRAM matrix multiply. To place the data on a common scale, performance is given as MFlops per node. It is well-known that the efficiency of matrix multiplication increases as the matrix dimension n increases, since the communication overhead becomes less significant. However, the BSP version never achieves performance comparable to the BPRAM algorithm. Even for $n = 880$, the communication cost in the BSP version still accounts for 20% (39%) of the total execution time when $p = 8$ ($p = 64$). In both cases, it accounts for less than 2.5% of the total time in the BPRAM version. Obviously, when the matrix size is increased further, the BSP algorithm will eventually reach performance comparable to the BPRAM algorithm. This is unsatisfactory, however, because for matrix multiplication it is easy to find an algorithm whose communication overhead grows much slower than the amount of local computation.

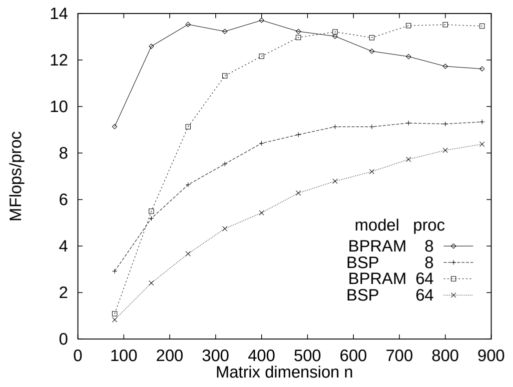

Figure 6. Performance of the matrix multiplication algorithms.

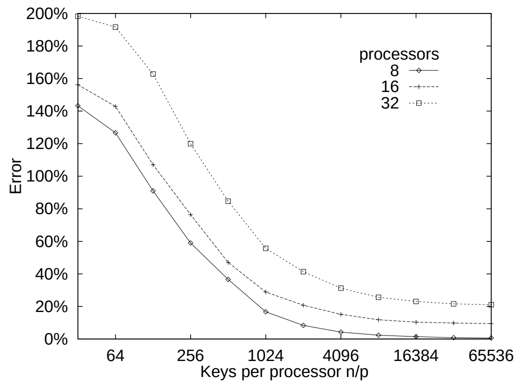

Figure 7. Errors in the BSP predictions for bitonic sort.

Bitonic Sort. Fig. 7 plots the errors in the BSP predictions for bitonic sort, where the error is defined as $(T_{\text{pred}} - T_{\text{meas}})/T_{\text{meas}}$. It can be seen that especially for small values of n/p, there are significant errors in the predictions. They are caused by the fact that the communication patterns in bitonic sort correspond to one-to-one routing patterns, whereas the BSP model assumes that every superstep involves the routing of an h-relation (all-to-all). The implemented BSP library routes every communication pattern using $d + s + \log p$ startups, where d (s) is the maximum number of messages sent (received) by any processor. In bitonic sort $d = s = 1$, but in an arbitrary h-relation $d = s = p-1$. So, when the startup times dominate, the BSP model overestimates the actual execution times. Note that the error does not stay constant but grows as $\Theta(p/\log p)$.

Sample Sort. Fig. 8 compares the measured times per key of the BSP and BPRAM version of sample sort on a 64-processor configuration. We found the performance of the BPRAM algorithm rather disappointing. With many keys per processor, it is only slightly faster than the BSP

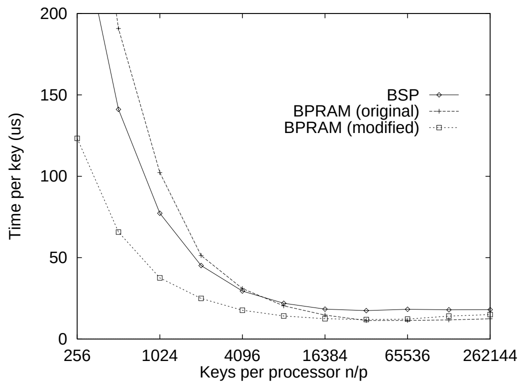

Figure 8. Performance of the sample sort algorithms.

algorithm, and with few keys per processor, it is even outperformed by the BSP algorithm. But, the BPRAM cost model provides enough information to guide us towards a more efficient solution: the number of startups is too large. In particular, in the original algorithm the keys are routed directly to their destination processors, requiring $p - 1$ startups. We thereupon modified the algorithm so that the keys are routed to their destinations in a butterfly-like fashion, requiring $\log p$ startups. It is important to note that this increases the communication volume as well as the number of synchronizations, and therefore goes against the incentives provided by the BSP model.

The curve labeled ‘BPRAM (modified)’ in Fig. 8 shows the performance achieved by the modified algorithm. It can be seen that it is faster than the original algorithm until the number of keys per processor increases beyond 32K. At that point, the communication volume starts to show up in the communication overhead. In any case, the modified algorithm is always faster than the BSP algorithm.

All Pairs Shortest Path. The BSP as well as the BPRAM model assume that communication time is independent of network traffic. In other words, both models ignore the issue of unbalanced communication. However, because it appears that the Paragon has excess bandwidth, one has to use a large configuration in order to show the effects of unbalanced communication. In this section, the all pairs shortest path (APSP) problem is taken to investigate the effects of bisection bandwidth limitations on the accuracy of the BPRAM. Since the time needed for routing h-relations is dominated by send and receive overheads, unbalanced communication has almost no effect on the accuracy of the BSP model.

We modified the APSP algorithm described in [8] in order to obtain good overall performance. The original algorithm consisted of n iterations, each involving $2 \cdot \log p$ startups. The modified algorithm is derived from the algorithm given in [12]. Briefly, it consists of $\sqrt{p}$ iterations in which

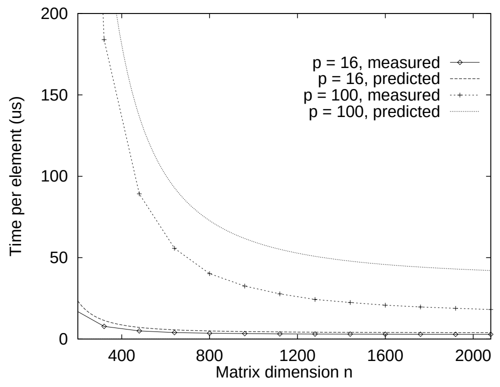

Figure 9. Predicted and measured communication times per element for APSP.

the transitive closure of the diagonal block is computed and broadcast to all processors in a binary tree fashion. Unbalanced communication occurs at the top levels of the broadcast tree, since then only a few processors are active.

Fig. 9 compares the communication time taken by the algorithm with that predicted by the BPRAM. To place the data on a common scale, the communication time is divided by the number of elements per processors. The local computation time is not included, since the communication overhead is only a small fraction of the total time. As expected, when p is small, the predictions are quite accurate. Except when the matrix dimension n is small, they almost coincide with the measured times. On the other hand, when p is large, the BPRAM significantly overestimates the actual communication times, due to unbalanced communication.

5. Conclusions

In this paper, we evaluated some of the proposed parallel computation models quantitatively on the Paragon. Concerning the accuracy of the models, we found that in most cases the BSP model as well as the BPRAM were able to predict the execution times of the algorithms fairly accurately. Two notable exceptions were bitonic sort and all pairs shortest path. Because the communication patterns in bitonic sort are one-to-one instead of all-to-all, BSP overestimated the execution time by a significant amount. Due to unbalanced communication, the BPRAM significantly overestimated the communication time for all pairs shortest path.

Comparing BSP with BPRAM, this paper has shown that *any reasonable model for the Paragon must incorporate the startup cost of a message transmission*. Although postponing all communication until the end of a superstep and combining all packets with the same destination into one message reduces the value of g by roughly a factor of 20 compared to a naive implementation that sends packets when they are generated, it does not reduce g to a value which is comparable to the reciprocal of the bisection bandwidth. The reason is that memory accesses and buffer management account for a significant part of the communication overhead. More seriously, as the sample sort algorithm has shown, the execution time of an algorithm implementation can sometimes be significantly reduced by performing optimizations that go against the incentives provided by the BSP model.

Acknowledgments. The author would like to thank Pat Morin and Ingo Rieping for their comments.

References

[1] A. Aggarwal, A. Chandra, and M. Snir. On Communication Latency in PRAM Computations. In *Proc. Symp. on Parallel Algorithms and Architectures*. ACM, 1989.

[2] R. Berrendorf, H. Burg, U. Detert, R. Esser, M. Gerndt, and R. Knecht. Intel Paragon XP/S - Architecture, Software Environment, and Performance. Technical Report KFA-ZAM-IB-9409, Forschungszentrum Jülich GmbH, 1994.

[3] J. Brandenburg. Technology advances in the Intel Paragon system. In *Proc. 5th Symp. on Parallel Algorithms and Architectures*. ACM, 1993. Industrial presentation.

[4] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proc. 4th Symp. on Principles and Practice of Parallel Programming*. ACM SIGPLAN, 1993.

[5] M. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas. Towards Efficiency and Portability: Programming with the BSP Model. In *Proc. 8th Symp. on Parallel Algorithms and Architectures*. ACM, 1996.

[6] J. Hill, W. McColl, D. Stefanescu, M. Goudreau, K. Lang, S. Rao, T. Suel, T. Tsantilas, and R. Bisseling. BSPlib - The BSP Programming Library, 1997. Reference manual.

[7] B. Juurlink. Experimental Validation of Parallel Computation Models on the Intel Paragon. Technical Report TR-RSFB-98-055, Paderborn University, 1998.

[8] B. Juurlink and H. Wijshoff. A Quantitative Comparison of Parallel Computation Models. In *Proc. 8th Symp. on Parallel Algorithms and Architectures*. ACM, 1996. Full version available as TR 96-01, Leiden University, The Netherlands.

[9] L. Liu and D. Culler. Evaluation of the Intel Paragon on Active Message Communication. In *Proc. of Intel Supercomputer Users Group Conference*, 1995.

[10] D. Skillicorn, J. Hill, and W. McColl. Questions and Answers About BSP. *Journal of Scientific Programming*, 1997. To appear.

[11] L. Snyder. Experimental Validation of Models of Parallel Computation. In J. van Leeuwen, editor, *Computer Science Today*. Springer LNCS 1000, 1995.

[12] J. Ullman and M. Yannakakis. The Input/Output Complexity of Transitive Closure. *Annals of Mathematics and Artificial Intelligence*, 3, 1991.

[13] L. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8), 1990.