A Quantitative Comparison of Parallel Computation Models

Ben H.H. Juurlink   Harry A.G. Wijshoff

High Performance Computing Division, Department of Computer Science
Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands
{benj,harryw}@cs.leidenuniv.nl

Abstract

This paper experimentally validates performance related issues for parallel computation models on several parallel platforms (a MasPar MP-1 with 1024 processors, a 64-node GCel and a CM-5 of 64 processors). Our work consists of three parts. First, there is an *evaluation* part in which we investigate whether the models correctly predict the execution time of an algorithm implementation. Unlike previous work, which mostly demonstrated a close match between the measured and predicted running times, this paper shows that there are situations in which the models do not precisely predict the actual execution time of an algorithm implementation. Second, there is a *comparison* part in which the models are contrasted with each other in order to determine which model induces the fastest algorithms. Finally, there is an *efficiency validation* part in which the performance of the model derived algorithms are compared with the performance of highly optimized library routines to show the effectiveness of deriving fast algorithms through the formalisms of the models.

1 Introduction

The PRAM model [12] is one of the most widely used models for the design and analysis of parallel algorithms. Its shared memory abstraction and the assumption that the processors operate synchronously make it a relatively easy model to use. Unfortunately, the PRAM model makes certain simplifying assumptions which make it not well suited for predicting performance on actual parallel machines. Most importantly, because the PRAM model does not capture communication cost, it does not discourage the design of parallel algorithms with huge amount of interprocessor communication. In more realistic network models, communication is only allowed between directly connected processors. However, this does not seem to be a promising approach either, because it produces non-portable software.

For these reasons, several alternative models have been proposed that model the interprocessor communication cost. However, with a few exceptions, e.g. [4, 9, 11], no quantitative results comparing theoretical with experimental results are provided. This paper tries to fill that gap by comparing, in a quantitative manner, some of the models that have been proposed. Specifically, we focus on the following questions:

- First, do the models accurately predict the execution time of an algorithm implementation? If not, what are the reasons for the inaccuracy?
- How do the models compare with each other? In general, different models reward different techniques, some of which may be more important to capture than others.
- How do the model derived algorithms compare with implementations customized for the target architecture? It is inevitably true that the price to be paid for portability is performance, but what is the performance loss that can be expected?

To answer these questions we selected three problems that seem to cover a wide range of parallel processing applications, and designed algorithms for these problems using three different models. These algorithms were subsequently implemented on three substantially different parallel architectures.

The models considered are (1) the Bulk-Synchronous Parallel (BSP) model [20], (2) the Message-Passing Block PRAM (MP-BPRAM) [1], and (3) the Extended BSP or E-BSP model [17]. A BSP computer is characterized by three parameters: the number of processors P, the synchronization cost/communication latency L and the communication bandwidth g. In the BSP model, the communication network is assumed to be capable of routing any balanced communication pattern, in which each node sends at most h messages and receives at most h messages, in $g \cdot h + L$ time. Such communication patterns are called h-relations. All messages are assumed to be of some fixed short size. Hence, the model implicitly has a fourth parameter, the word size w.

The BSP model does not give special treatment to long messages. However, as is evidenced in many papers, e.g. [4], many existing architectures have special support for bulk transfers. In a MP-BPRAM the processors communicate by exchanging messages of arbitrary length, and a message of length m is transferred in time $m + \ell$, where ℓ is the startup time of a message transmission. An important restriction made by the MP-BPRAM is that a processor can send and receive at most one message in a single communication step.

Both the BSP model and the MP-BPRAM assume that communication bandwidth is independent of network traffic. For example, under the BSP model, sending h messages between two processors is just as expensive as when all processors send and receive h messages. The E-BSP model addresses this issue by viewing every communication pattern as an (M, h_1, h_2)-relation, in which each processor sends at most h_1 messages, receives at most h_2 messages and the total number of messages being routed does not exceed M.

This paper is organized as follows Section 2 describes, in more detail, the parallel computation models that we used as the basis for our work. In Section 3, the experimental platforms are described and the model parameters belonging to these architectures are determined. A brief outline of the algorithms that were implemented is given in Section 4 In Section 5, the models are evaluated by comparing the predictions with experimental data. Section 6 compares the BSP and MP-BPRAM model with each other. In Section 7 we compare the performance of the model derived matrix multiplication algorithms with the performance of the matrix multiply routines present on the MasPar and the CM-5. Concluding remarks are given in Section 8.

2 Parallel Computation Models

In this section, the models considered are briefly reviewed.

2.1 BSP Model

The Bulk-Synchronous Parallel model [20] consists of three attributes: (1) a set of processor/memory modules, (2) a communication network that delivers messages point-to-point, and (3) facilities to barrier synchronize the nodes. Computations in the BSP model are organized in a series of *supersteps*, with synchronization taking place between supersteps. Each superstep consists of a number of local operations followed by sending and receiving messages. The performance of a BSP computer is determined by the following parameters:

- The number of processor/memory modules P.
- The synchronization cost/communication latency L.
- The bandwidth factor g, which is defined as the ratio of local operations performed by all processors in one time unit to the total number of messages delivered by the router in one time unit.

McColl [19] also includes a speed parameter s which is defined as the number of floating point operations performed per time unit. We decided not to include this parameter because it depends very much on the application domain. We also do not normalize g and L w.r.t. processor speed. Instead, we use actual times (in μs).

The cost of a superstep S is determined as follows. Let c denote the maximum amount of local computations performed by any processor during S (measured in μs). Further, let h_s be the maximum number of messages sent by any processor during S, and let h_r be the maximum number of messages received by any processor during S. The cost of S is[1] $c + g \cdot \max\{h_s, h_r\} + L$. The parameters g and L are such that an arbitrary h-relation followed by a barrier synchronization can be completed in $g \cdot h + L$ time. The purpose of the parameter L is two-fold. It is the latency or startup cost of a communication, and it represents the cost of synchronizing the processors.

[1]In the original BSP model, the cost of a superstep S is $\max\{c, g \cdot h_s, g \cdot h_r, L\}$. Our cost definition more closely follows [6].

2.2 Message-Passing Block PRAM

The Message-Passing Block PRAM is a restrictive version of the Block PRAM model defined in [1] A Block PRAM consists of P processors, each provided with a local memory of unbounded size, communicating with each other through a shared global memory of unlimited size. Each processor $\langle i \rangle$, $0 \leq i < P$, may read or write a block of b_i contiguous locations from or to the shared memory in one step, and such an operation takes $\ell + \max_i b_i$ time, where ℓ is the startup time or latency. An access to a location of the local memory is assumed to take unit time. Any number of processors may access the shared memory simultaneously, but the blocks that are being accessed need to be disjoint.

In a Message-Passing Block PRAM there is no global memory but instead the processors communicate by exchanging messages[2]. A processor can send and receive only one message at a time, and a message of length m is transferred in time $m+\ell$. The model is synchronous, so that every processor awaits the completion of the longest block transfer before it proceeds to the next step. Because communication is generally slower than computation, even if block transfers are used, we model the time to send a message of m bytes by the formula $\sigma \cdot m + \ell$, where σ is the time per byte to send a message.

2.3 E-BSP Model

Both BSP and the MP-BPRAM assume that the parameters g and σ are independent of network traffic. The BSP model assumes that all h-relations are *full* h-relations in which all processors send and receive exactly h messages. Likewise, under the MP-BPRAM sending a single message of length m is just as expensive as when all processors participate. The E-BSP model [17] extends the basic BSP model to deal with unbalanced communication patterns, i.e., patterns in which the amount of data sent or received by each node is different. Consider for example a mesh connected parallel computer that employs a store-and-forward routing scheme, and assume that in unit time a processor can send a message to its neighbors. The value of g that can be obtained on this architecture is $\Theta(\sqrt{P})$. Consider now the following simple communication operation: sending h messages between two processors. Under the BSP model $g \cdot h + L = \Theta(h \cdot \sqrt{P})$ time is charged for this communication operation. However, such a communication pattern can be accomplished in $h + 2 \cdot \sqrt{P}$ time on this architecture because the messages can move in a pipelined fashion, encountering no conflicts in the network. To deal with unbalanced communication the E-BSP model views every communication pattern as an (M, h_1, h_2)-relation, in which each processor sends at most h_1 messages, receives at most h_2 messages and the total number of messages being routed does not exceed M. Note that an h-relation is just a special instance of an (M, h_1, h_2)-relation with $M = h \cdot P$ and $h_1 = h_2 = h$.

3 Experimental Platforms

This section describes the experimental platforms and the experiments conducted to determine the parameters belonging to these platforms. In an earlier paper, we did a limited study for a T800 platform [15].

[2]Another model that has many of the aspects of the MP-BPRAM is the LogGP model [4].

3.1 MasPar MP-1

The MasPar MP-1 is a massively parallel SIMD architecture The MP-1 system used in this study consists of 1024 processors, called processor elements (PEs). Each PE is an 80 ns load/store arithmetic processor with 64 Kbytes of data memory The MasPar provides two types of communication: xnet and router communication. With xnet communication a PE can send data to eight neighboring PEs in horizontal, vertical and diagonal directions. In addition, the MasPar has a global router to communicate data between arbitrary PEs. This router is a circuit-switched expanded delta-network with a greedy routing scheme We worked exclusively with router communication.

When matching the BSP model to the MasPar MP-1 architecture, we faced a number of problems. In particular, BSP permits *memory pipelining*: a remote memory access can start before the previous one has completed. However, on the MasPar each PE can have at most one outstanding message at a time. We therefore define the MP-BSP model which is a small variation of BSP that reflects this architecture more accurately. The MP-BSP model is a synchronous model in which the processors communicate by writing into the local memory of other processors. Each step is either a computation step or a communication step:

1. In a *computation step*, each processor $\langle i \rangle$ performs an operation on operands present in its local memory.
2. In a *communication step*, each processor $\langle i \rangle$ writes a data item into the local memory of some other processor.

Let h_i be the number of processors accessing the local memory of processor $\langle i \rangle$ during a communication step. The cost of this step will be modeled by the formula $L + g \cdot \max_i h_i$. Thus, every communication step is an instance of a 1–h relation, in which each processor sends at most one message and is due to receive at most h messages.

In order to determine the MP-BSP parameters for the MasPar the following experiment was conducted. The array control unit (ACU) randomly picks a set of $\lceil P/h \rceil$ destinations. We then measured the time for a communication step in which $\lfloor P/h \rfloor$ nodes receive h messages consisting of $w = 4$ bytes, while the remaining destination (if any) receives $P - h \cdot \lfloor P/h \rfloor < h$ messages. The results of this experiment are shown in Fig. 1. Each data point in this figure represents the average of 100 experiments. The minimum and maximum measured times are also shown using vertical error bars.

Ideally, the data points would form a straight line with slope g and offset L. The observed behavior is not completely linear. Thus, by charging $g \cdot h + L$ time for routing 1–h relations, an error will be introduced. The large variation in the measurements is due to the limitation that there is only one router channel for each cluster of 16 PEs. If many destinations happen to fall in the same cluster, the router is considerably slower than if the destinations are distributed equally among the clusters. The MP-BSP parameters for the MasPar MP-1 are shown in Table 1. These parameters were determined by fitting a straight line to the data points in Fig. 1. Also shown in Table 1 are the MP-BPRAM parameters belonging to the MasPar. These parameters have been obtained by measuring the time taken by full block permutations. For reasons of space the results of these experiments are not shown, but the data points demonstrated that the

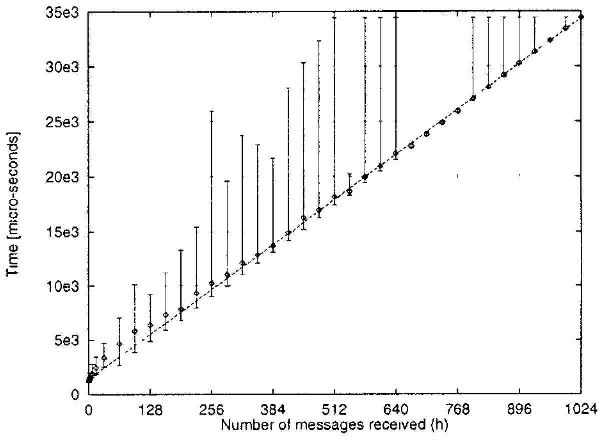

Figure 1: Time required for routing 1–h relations on the MasPar MP-1.

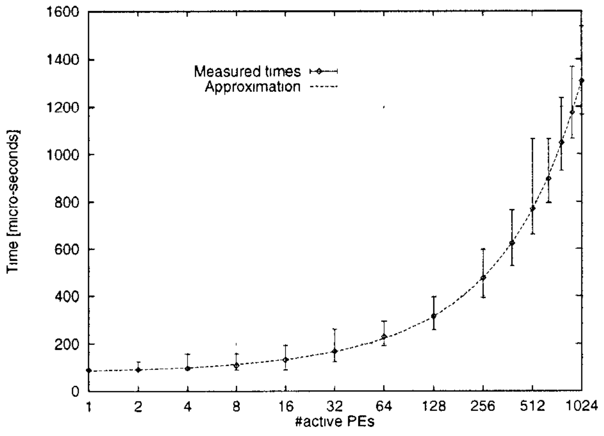

Figure 2: Time taken by partial permutations as a function of the number of active processors on the MasPar.

time to send a message of m bytes is well approximated by a straight line with slope σ and offset ℓ.

As was done for the BSP model, we used a small variation of E-BSP that reflects the MasPar more accurately: the cost of a communication step is modeled as a function of the number of processors that are active during that communication step Specifically, let $T_{\text{unb}}(P')$ denote the cost of performing a partial permutation when there are P' processors active. An approximation for $T_{\text{unb}}(P')$ was determined by conducting the following experiment. The ACU of the MasPar randomly selects a set of P' senders $s_1, s_2, \ldots, s_{P'}$ and a set of P' recipients $r_1, r_2, \ldots, r_{P'}$. Thereupon, we measured the time taken by a partial permutation in which processor $\langle s_i \rangle$ sends a message to processor $\langle r_i \rangle$. The results of this experiment are shown in Fig. 2. As is apparent in this figure, the communication time is very dependent on the number of PEs participating. For example, when there are 32 active PEs, a partial permutation takes about 13% of the time required by a full permutation. By performing a second order polynomial fit, we found that

$$T_{\text{unb}}(P') = 0\ 84 \cdot P' + 11.8 \cdot \sqrt{P'} + 73.3\ \mu s$$

yields a good approximation.

Architecture	P	g	L	σ	ℓ
MasPar	1024	32.2	1400	107	630
GCel	64	4480	5100	9.3	6900
CM-5	64	9.1	45	0.27	75

Table 1: Summary of the (MP-)BSP and MP-BPRAM parameters belonging to the MasPar MP-1, GCel and the CM-5 architecture. The parameters are given in μs.

3.2 GCel

The second experimental platform is a 64-node Parsytec GCel. Each node of the GCel consists of a T805 transputer, running at 30 Mhz, with 4 MB RAM. The nodes are organized as an 8×8 two-dimensional mesh. The native programming system of the GCel is Parix, but the programs for the GCel are implemented using so-called homogeneous PVM or HPVM which is built on top of Parix. HPVM provides a much better performance than standard PVM because heterogeneous operations are not supported, although its performance is still less than sparkling.

To determine the BSP parameters for the GCel under HPVM, we measured the time taken by randomly generated full h-relations. The MP-BPRAM parameters have been calculated by timing full block permutations. The parameters for the GCel are summarized in Table 1. The ratio $g/(w \cdot \sigma)$ is an indicator of the gain that can be obtained by grouping data into long messages. For the GCel, this ratio is about 120, which leads us to conclude that large messages are essential on this architecture.

3.3 CM-5

The Connection Machine Model 5 (CM-5) is a massively parallel MIMD computer based on the Sparc processor. Each node contains a Sparc Cypress processor running at 32 MHz, with 32 MB of local memory, a 64 KB direct-mapped cache and a network interface chip. The nodes are interconnected by a fat tree communication network, and a broadcast/scan/prefix control network. The programs for the CM-5 are written in Split-C [10], a parallel extension of the C programming language. They do not use the vector units, which are not available under Split-C.

The BSP and MP-BPRAM parameters for the CM-5 are shown in Table 1. On this architecture, the ratio $g/(w \cdot \sigma)$ is about 4.2 for 8-byte (double precision) messages, so it is much less important to use block transfers than on the GCel.

4 Description of the Algorithms

To validate the models, we selected three problems (matrix multiplication, sorting and all pairs shortest path) from dense linear algebra and discrete mathematics that seem to cover a wide range of parallel processing applications. More specifically, we have chosen matrix multiplication as this routine is present in almost all dense linear algebra computations. For sorting we picked two different algorithms. One algorithm is bitonic sort which represents recursive doubling techniques. The other is based on an even distribution of workload. The problem from graph theory is all pairs shortest path which has a communication structure that is similar to many other important algorithms such as LU decomposition. For reasons of space, the descriptions will be concise. The reader is referred to [18] for a more detailed account.

4.1 Matrix Multiplication

We implemented the following algorithm for multiplying two $N \times N$ matrices $C = A \cdot B$, which follows a strategy similar to the one described in [2, 13], and was also adapted for the BSP model in [8]. Lower bound proofs as in [2, 13] show that this algorithm is optimal under the BSP model.

The algorithm uses $P = q^3$ processors. It is convenient to think of the processors as being arranged in a $q \times q \times q$ array, i.e., let the processors be designated by $\langle i, j, k \rangle$ for $0 \leq i, j, k < q$. The matrices A, B and C are partitioned into q^2 square submatrices A_{ij}, B_{ij} and C_{ij}, $0 \leq i, j < q$, of size $N/q \times N/q$ each. Each of these submatrices is further subdivided into q subblocks A_{ij}^k, B_{ij}^k and C_{ij}^k for $0 \leq k < q$ of size $N/q^2 \times N/q$, where A_{ij}^0 consists of the first N/q^2 rows of A_{ij}, A_{ij}^1 consists of the second set of N/q^2 rows, and so on. Initially, processor $\langle i, j, k \rangle$ contains the subblocks A_{ij}^k and B_{ij}^k, and the subblock C_{ij}^k of the resulting matrix will also be stored there.

The algorithm consist of four supersteps. In the first, every processor $\langle i, j, k \rangle$ transmits each entry of A_{ij}^k to the processors $\langle i, j, * \rangle$ and, similarly, each entry of B_{ij}^k is replicated q times and sent to the processors $\langle *, i, j \rangle$. Since the matrices are initially distributed equally among the processors, the BSP cost of this superstep is $2 \cdot g \cdot N^2/q^2 + L$. In the second superstep, processor $\langle i, j, k \rangle$ locally computes $\widehat{C}_{ijk} = A_{ij} \cdot B_{jk}$. If we view an addition and a multiplication as a compound operation that takes time α, the cost of the second superstep is given by $\alpha \cdot N^3/P$. Now, let the partial products $\widehat{C}_{ijk}$ also be divided into q subblocks $\widehat{C}_{ijk}^l$. In the third superstep processor $\langle i, j, k \rangle$ transmits $\widehat{C}_{ijk}^l$ to the processor $\langle i, k, l \rangle$. Finally, the processors locally compute the sum of the appropriate submatrices. The total running time $T_{\text{bsp-mm}}$ of the matrix multiplication algorithm is:

$$T_{\text{bsp-mm}} = \alpha \cdot N^3/P + \beta \cdot N^2/q^2 + 3 \cdot g \cdot N^2/q^2 + 2 \cdot L.$$

Under the MP-BSP model, care must be taken to avoid concurrent writes to the same memory module. To achieve this, the communication is staggered. For example, in Phase 2, processor $\langle i, j, k \rangle$ first sends data to $\langle k, i, j \rangle$, then to $\langle (k+1) \bmod q, i, j \rangle$ and so on. The MP-BSP complexity of the matrix multiplication algorithm is given by:

$$T_{\text{mp-bsp-mm}} = \alpha \cdot N^3/P + \beta \cdot N^2/q^2 + 3 \cdot (g + L) \cdot N^2/q^2.$$

The algorithm can be easily restructured to use blocks of size N^2/P for data transfer. Note that the ability to use blocks of this size depends on the initial distribution of the matrices. If the initial distribution is different, an extra communication phase bringing the data in the desired layout is required. In the BSP model this is not an issue. The predicted running time of this version is:

$$T_{\text{bpram-mm}} = \alpha \cdot N^3/P + \beta \cdot N^2/q^2 + 3 \cdot q \cdot (\sigma \cdot w \cdot N^2/P + \ell).$$

In this expression, w denotes the computational word size in bytes.

4.1.1 Optimizing Local Computation

Local computations remain unspecified in the models. However, it is important to optimize the local matrix multiply carefully in order to obtain competitive results. On the MasPar, the most non-algorithmic performance improvement one can make is to keep variables in registers. In our implementation an optimized blocked inner-product algorithm is used that keeps part of the matrix C in registers, reducing the cost of the local matrix multiply by roughly 40%. On the CM-5, the local matrix multiply is designed to pay careful attention to the local cache size, and has a kernel written in assembly[3]. This optimized routine achieves 6.5 to 7.5 Mflops for square matrices of size 32×32 to 256×256, whereas the peak performance is about 9.0 Mflops. When $N = 512$, the performance drops to 5.2 Mflops. For the predictions, we take $\alpha = 2 \ (7.0 \ 10^6)^{-1} \approx 0.29 \ \mu s$. The factor of 2 appears in this formula because an addition and a multiplication is viewed as a compound operation.

4.2 Bitonic Sort

Batcher's bitonic sort [5] works by repeatedly merging so-called bitonic sequences until the entire sequence is sorted. The basic algorithm for sorting $N = P$ keys consists of $\log P$ merge stages, and the dth stage, $1 \leq d \leq \log P$, comprises d merge steps. In the jth step of the dth stage, $1 \leq j \leq d$, every processor exchanges the key in its possession with the processor whose address is identical except in the $(d-j)$th bit, and it subsequently picks the minimum or the maximum of the two keys depending on its address. To deal with multiple keys per processor, every processor first sorts the set of N/P keys it contains locally, and the sorted lists are merged in $\log P$ stages. Each processor now executes a linear-time sequential merging procedure and outputs N/P keys in each merge step[4].

Let $M = N/P$ denote the number of keys per processor. The BSP cost of the dth merge stage is $T_{\text{merge}} = d \cdot (\alpha \cdot M + g \cdot M + L)$, where the term $\alpha \cdot M$ accounts for the time taken by a local merge, and the term $g \cdot M + L$ corresponds to the communication time in each step. The total time taken by bitonic sort is:

$$T_{\text{bsp-bitonic}} = T_{\text{local-sort}} + \sum_{d=1}^{\log P} T_{\text{merge}}.$$

Under the MP-BSP model, the expression is slightly different: the time taken by a bitonic merge stage is $T_{\text{merge}} = d \cdot (\alpha \cdot M + (g + L) \cdot M)$, and the time for the entire bitonic sort becomes:

$$T_{\text{mp-bsp-bitonic}} =$$
$$T_{\text{local-sort}} + 0.5 \cdot \log P \cdot (\log P + 1) \cdot (\alpha \cdot M + (g + L) \cdot M).$$

As with the matrix multiplication algorithm, bitonic sort can be directly adapted to utilize block transfers. The total running time of this variation is:

$$T_{\text{bpram-bitonic}} =$$
$$T_{\text{local-sort}} + 0.5 \cdot \log P \cdot (\log P + 1) \cdot (\alpha \cdot M + \sigma \cdot w \cdot M + \ell).$$

[3]The local matrix multiply is due to Culler et al.

[4]When $N \geq P^2$, a better way of implementing bitonic sort that requires only two all-to-all communications in each merge stage is given in [11]. We did not implement that variation because it requires that $N \geq P^2$.

4.2.1 Local Sort

To sort the keys locally, we used an 8-bit radix sort which requires time

$$T_{\text{local-sort}} = (b/r) \cdot (\beta \cdot 2^r + \gamma \cdot N/P)$$

Here, b denotes the number of bits in a key and 2^r is the radix of the sort. The coefficients β, γ were determined empirically on each platform.

4.3 Sample Sort

The second sorting algorithm that we implemented is a sample sort based on [7]. Until now, we were able to break down any communication pattern in the MP-BSP model into a sequence of permutations. In this algorithm concurrent writes to the same memory module cannot easily be avoided. Under the MP-BPRAM, it is *forbidden* that a processor needs to receive multiple messages in a communication step, so a different algorithm is needed.

Sample sort proceeds in three phases. In Phase 1, the *splitter* phase, every processor randomly picks a set of S samples from its keys, where S is called the *oversampling ratio*. After that, the $P \cdot S$ samples are sorted using bitonic sort. Finally, the samples with ranks $S, 2 \cdot S, \ldots, (P-1) \cdot S$ are chosen as the splitters and broadcast to every other processor. The BSP complexity of this phase is:

$$T_{\text{splitter}} = T_{\text{bsp-bitonic}}(P \cdot S) + g \cdot (P - 1) + L.$$

Phase 2, the *send* phase, begins with each processor sorting the keys it contains. Since the keys and splitters are now sorted, the buckets to which each key belongs can be determined in $\Theta(M + P)$ time[5].

When sample sort was implemented, we encountered an inconvenience of the **pp_rsend** library routine of the MasPar programming language MPL. Because a local memory address within the destination processor has to be supplied, the keys cannot be routed directly to their buckets. Idem, when pairwise sends and receives are used, each processor needs to know the number of keys it is due to receive. The addresses are calculated by performing a multi-scan operation on the number of keys every processor needs to send to each bucket, for which an optimal BSP algorithm has been presented in [16]. The cost of this operation is $T_{\text{scan}} = 2 \cdot (g \cdot P + L)$. In the last step of the send phase the keys are routed to their buckets. Let $M_{\max}$ denote the maximum number of keys in a any bucket, the BSP cost of this phase is:

$$T_{\text{send}} = T_{\text{local-sort}}(M) + \alpha \cdot (M + P) + T_{\text{scan}} + g \cdot M_{\max} + L.$$

In the final phase of sample sort, the keys are sorted locally using radix sort. This takes time:

$$T_{\text{sort-buckets}} = T_{\text{local-sort}}(M_{\max}).$$

4.3.1 Irregular Communication and Bulk Transfer

Unlike matrix multiplication and bitonic sort, sample sort cannot straightforwardly be adapted to utilize block transfers. Three substeps must be implemented differently. the broadcast of the splitters, the multi-scan operation and the distribution of the keys to their buckets.

[5]Here we slightly deviate from the scheme presented in [7]. There, the bucket to which each key belongs is determined by performing a binary search over the array of splitters.

The broadcast of the splitters can be viewed as transposing an array of size $P \times P$, where column i is stored in processor $\langle i \rangle$. This can be done as follows. Each processor is assigned the task of transposing a submatrix of size $\sqrt{P} \times \sqrt{P}$. To do so it needs to receive $\sqrt{P}$ messages of length $\sqrt{P}$. After the submatrices are transposed locally, each block of length $\sqrt{P}$ is routed to its correct destination. The resulting communication time is $2 \cdot \sqrt{P} \cdot (\sigma \cdot w \cdot \sqrt{P} + \ell)$. With a similar approach, the multi-scan operation can be performed in $4 \cdot \sqrt{P} \cdot (\sigma \cdot w \cdot \sqrt{P} + \ell)$ time.

The most difficult part is the implementation of the send substep. Because each MP-BPRAM processor may receive at most one message during a communication step, the keys cannot be routed directly to their appropriate buckets. A possible scheme for routing messages using block transfers is given in [14]. We will not provide the details of this algorithm, but just present the complexity of the send phase:

$$T_{\text{send-to-buckets}} = 4 \cdot \sqrt{P} \cdot (4 \cdot \sigma \cdot w \cdot N/P^{1.5} + \ell).$$

4.4 All Pairs Shortest Path

An important problem in graph theory is all pairs shortest path. The problem is the following. Given a directed graph $G = (V, E)$, where $V = \{v_0, v_1, \ldots, v_{N-1}\}$ is a set of N vertices and $E \subseteq V \times V$ is a set of edges. With each edge (v_i, v_j) a length l_{ij} is associated. The task is to compute, for each pair of vertices v_i and v_j, the length of the shortest path from v_i to v_j. This problem can be solved by a number of algorithms. We have opted to implement a parallel version of Floyd's algorithm [3], because it admits a straightforward parallelization.

Floyd's sequential all pairs shortest path algorithm and a pseudocode description of its parallelization is given below. A matrix D is used to store the currently shortest path between any pair of nodes v_i and v_j. Initially, $D[i, i] = 0$, $D[i, j] = l_{ij}$ if an edge with length l_{ij} between v_i and v_j exists, and $D[i, j] = \infty$ otherwise. In the parallel version, the matrix D is partitioned into P square subblocks D_{ij} of size $M \times M$ each, where $M = N/\sqrt{P}$, and processor $\langle i, j \rangle$ is assigned the task of updating, in each iteration k, the entries pertaining to the subblock D_{ij}.

```
for $k = 0$ to $N - 1$ do
    for $i = 0$ to $N - 1$ do
        $X[i] = D[i, k]$
        $Y[i] = D[k, i]$
    od
    for $i = 0$ to $N - 1$ do
        for $j = 0$ to $N - 1$ do
            $D[i, j] = \min\{D[i, j], X[i] + Y[j]\}$
        od
    od
od
```

For $k = 0$ **to** $N - 1$ **repeat** steps (1)—(3).

1. The processors $\langle s, t \rangle$ that contain a segment of $D[*, k]$, send it to the processors $\langle s, * \rangle$.

2. The processors $\langle s, t \rangle$ that contain a segment of $D[k, *]$, send it to the processors $\langle *, t \rangle$.

3. Processor $\langle s, t \rangle$ computes the new values of the entries pertaining to the submatrix D_{st}.

The most interesting part of this algorithm is the broadcast of the "active" row and column in Steps (1) and (2). We describe the implementation of the broadcast of the active column; the broadcast of the active row can be done similarly. There are two cases. If $M \geq \sqrt{P}$, then there are two supersteps. In the first, the processors $\langle s, t \rangle$ that contain a segment of length M of the active column scatter these elements across the processors $\langle s, * \rangle$ so that each of them receives a subsegment of length $M/\sqrt{P}$. During the second superstep, each processor broadcasts the subsegment it contains to every other processor in the same row. The total communication cost can be seen to be $T_{\text{bcast}} = 2 \cdot (g \cdot M + L)$. If $M < \sqrt{P}$, an extra phase broadcasting each item to $\sqrt{P}/M$ processors is required. In this case, the communication cost is $T_{\text{bcast}} = 2 \cdot (g \cdot M + L) + (g + L) \cdot \log(\sqrt{P}/M)$ The total running time of this version of the all pairs shortest path algorithm is given by:

$$T_{\text{bsp-apsp}} = \alpha \cdot N^3/P + 2 \cdot N \cdot T_{\text{bcast}}.$$

Again, under the MP-BSP model, the analysis is slightly different: $T_{\text{bcast}} = 2 \cdot (g + L) \cdot M$ when $M \geq \sqrt{P}$, and $T_{\text{bcast}} = (g + L) \cdot (2 \cdot M + \log(\sqrt{P}/M))$ if $M < \sqrt{P}$.

4.4.1 E-BSP Analysis

So far, we have ignored unbalanced communication because all h-relations were full h-relations. However, the first superstep in the broadcast of the active row/column corresponds to an $(N, N/\sqrt{P}, N/P)$-relation. In this section, we analyze the all pairs shortest path algorithm under the E-BSP model for the MasPar.

Again, two cases are distinguished. If $M \geq \sqrt{P}$, the first phase of the broadcasting procedure consists of M communication steps and in each step $\sqrt{P}$ PEs are active. The cost of this phase is therefore $M \cdot T_{\text{unb}}(\sqrt{P})$. During the second phase, all processors are active. Adding the cost of the two phases gives $T_{\text{bcast}} = M \cdot T_{\text{unb}}(\sqrt{P}) + M \cdot T_{\text{unb}}(P)$. If $M < \sqrt{P}$, the extra phase consists of $\log(\sqrt{P}/M)$ communication steps, and in the ith step $2^i \cdot N$ PEs are active. In this case the communication cost is $T_{\text{bcast}} = M \cdot T_{\text{unb}}(\sqrt{P}) + M \cdot T_{\text{unb}}(P) + \sum_{i=0}^{\log(\sqrt{P}/M)-1} T_{\text{unb}}(2^i \cdot N)$.

5 Comparing Measured And Predicted Performance

In this section, the models are evaluated by comparing the measured execution times of the algorithms described in the previous section with the times predicted by the models.

5.1 BSP Model

Fig. 3 shows the measured and predicted performance of the MP-BSP version of the matrix multiplication algorithm on the MasPar. It can be seen that the MP-BSP cost yields a reasonable approximation of the actual runtime. For all measured data points, the deviation is less than 14%, and can be attributed to the fact that for $h = 1$, the time required for routing 1-h relations is not very well approximated by $g + L \approx 1430\ \mu s$. In reality, the time taken by a 1-1 relation is about 1300 μs on the average.

The predicted and measured performance on the CM-5 are depicted in Fig. 4. As is apparent from this figure, the

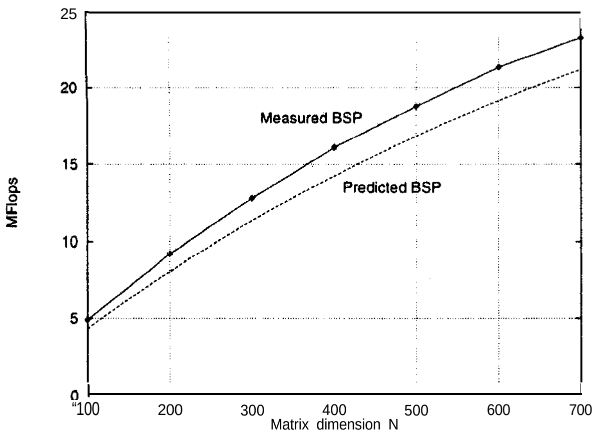

Figure 3: Measured and predicted performance of the MP-BSP version of the matrix multiplication algorithm on the MasPar.

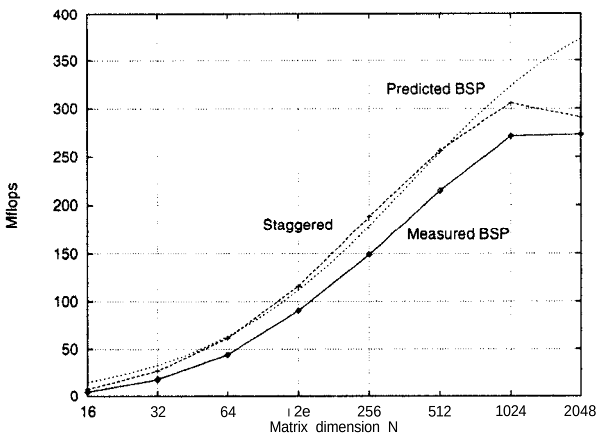

Figure 4: Measured and predicted performance of the BSP version of the matrix multiplication algorithm on the CM-5.

BSP model does not precisely estimate the actual performance. When the matrix dimension N is small ($N \leq 64$) or when N is large ($N \geq 1024$), the primary source of error is in the local computation. Due to cache effects, the local matrix multiply takes significantly longer than $\alpha \cdot N^3/P$ time. However, even for N = 256, the BSP model predicts an execution time of 188 milliseconds but the measured time is about 227 milliseconds. The relative error is 2170 which is too large to ignore. The defect is the result of processor contention which is not captured by the BSP model. In the first implementation of the matrix multiplication algorithm each processor (i, $j, k\rangle$ first sends data to (z, j, O), then to (i, j, 1), and so on, causing stalls to occur. This can easily be avoided by staggering the communication: processor $\langle i, j, k\rangle$ starts sending data to (i, j, k), then to (i, j, (k + 1) mod $P^{1/3}\rangle$ and so on, just as was done explicitly under the MP-BSP model. The curve labeled "Staggered" in Fig. 4 shows the measured performance of this implementation. Now, there is indeed a close match between the predicted and actual performance, except of course for small and large values of N.

Fig. 5 plots the predicted and measured times per key of bitonic sort on the MasPar. The total running time is obtained by multiplying the time per key by the number of keys per processor N/P. It can be seen that the MP-BSP model overestimates the actual running times by almost a factor of 2.0 on this architecture. This is entirely due to the fact that bitonic sort makes use of a communication pattern that is especially cheap on the MasPar global router. We have conducted experiments that show that permutations in which every processor communicates with the processor whose address is identical except in one bit require approximately 590 μs. This is less than 50% of the time taken by an average random permutation.

The measured and estimated times per key of bitonic sort on the GCel are shown in Fig. 6. As can be seen, there is a huge difference between the times predicted by the BSP model and the actual times per key. To explain this behavior, we measured the time required for performing h identical permutations, which is the communication pattern that arises in bitonic sort. We call such patterns h-h permutations. The results of this experiment are shown in Fig. 7. Until approximately h = 300, h-h permutations are slightly cheaper than randomly generated h-relations. Af-

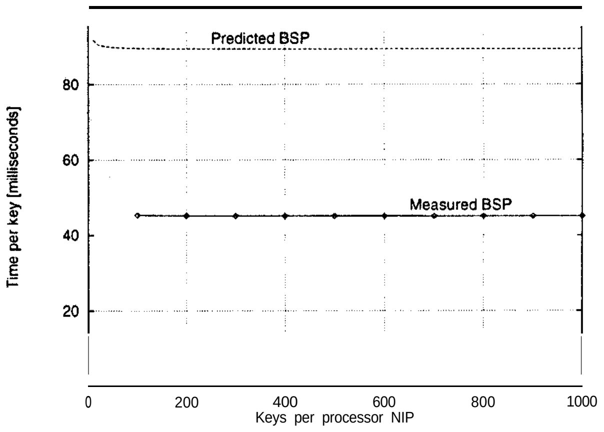

Figure 5: Measured and predicted times per key of bitonic sort on the MasPar.

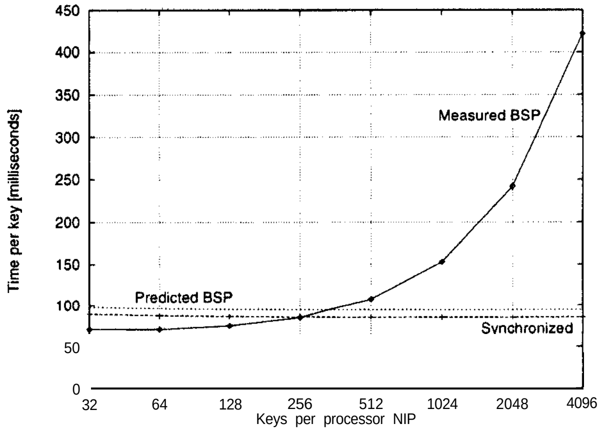

Figure 6: Measured and predicted times per key of bitonic sort on the GCel.

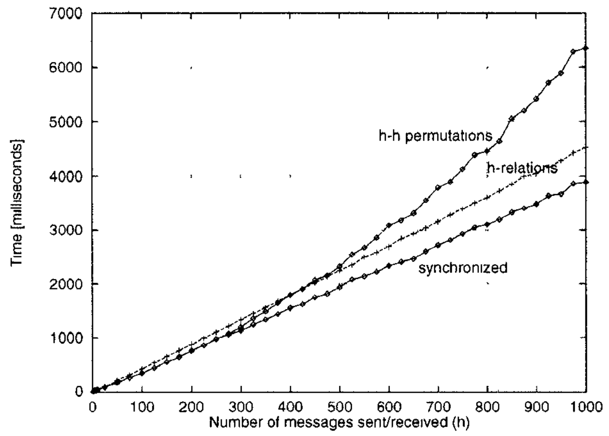

Figure 7: Time required for performing *h-h* permutations vs. the time required for performing randomly generated *h*-relations on the GCel under PVM.

ter that, the timing results become noisy and unpredictable, and the time taken by *h-h* permutations keeps elevating. It appears that the processors "drift out of sync"; an observation which also has been made in [9] for a different model. To reduce this effect, a barrier synchronization was added after each node has sent and received 256 messages. Fig. 7 shows that this modification eliminates the performance drop, and Fig. 6 demonstrates that now there is indeed a close match between the measured and expected times per key.

5.2 Message-Passing Block PRAM

Fig. 8 and Fig. 9 show the measured and predicted performance of the MP-BPRAM version of the matrix multiplication algorithm on the MasPar and the CM-5, respectively. Comparing the two curves verifies that on the MasPar (Fig. 8) the MP-BPRAM cost yields a very good approximation of the actual performance: all errors are less than 3%. On the CM-5 (Fig. 9), the predictions are also quite accurate provided that the local computations are precisely modeled.

The observed and estimated times per key for the MP-BPRAM variation of bitonic sort on the MasPar are depicted in Fig. 10. As was also observed in the BSP version of this algorithm, the MP-BPRAM significantly overestimates the measured times per key due to the fact that bitonic sort makes use of a fixed communication pattern that is particularly cheap on the MasPar global router. The MP-BPRAM predictions are slightly more precise than the times predicted by BSP, because the router is somewhat less sensitive to the actual communication pattern when long messages are being sent.

The measured times per key for bitonic sort on the GCel and the times predicted by the MP-BPRAM are displayed in Fig. 11. As is apparent from the figure the estimated times are very accurate; they almost coincide with the measured data points.

5.3 E-BSP Model

Fig. 12 plots the measured and predicted execution times of the all pairs shortest path algorithm on the MasPar. Comparing the observed performance with the performance es-

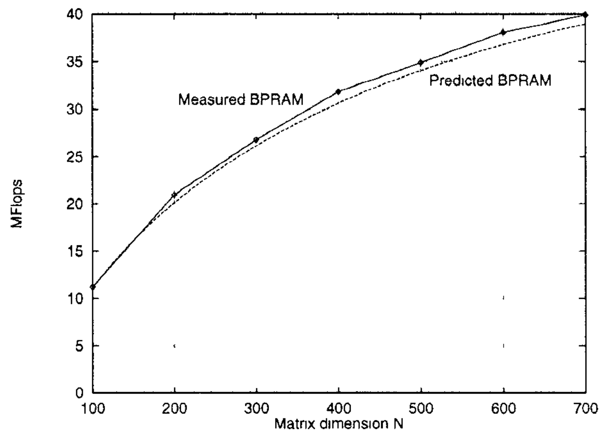

Figure 8: Measured performance of the matrix multiplication algorithm on the MasPar, and the performance predicted by the MP-BPRAM model.

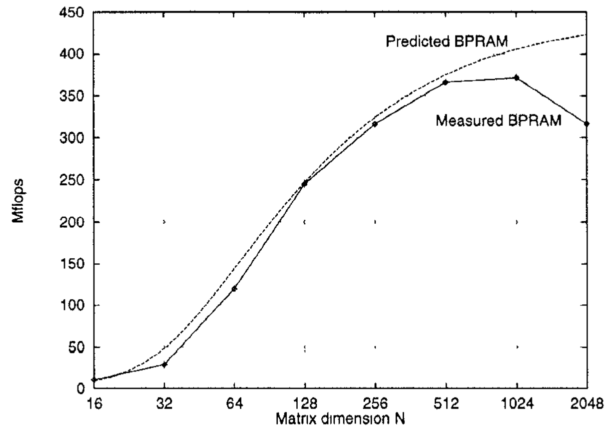

Figure 9: Measured and predicted performance of the MP-BPRAM matrix multiplication version on the CM-5.

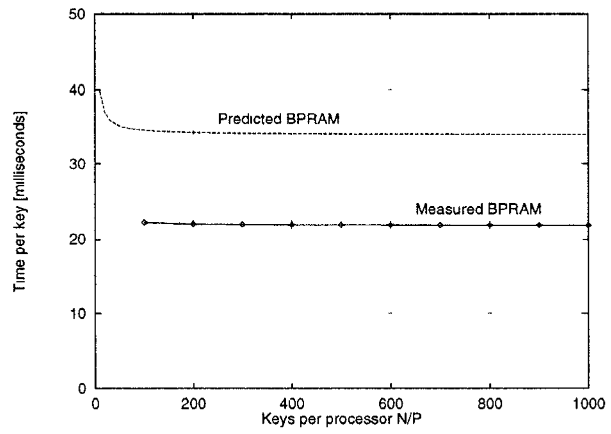

Figure 10: Measured and predicted times per key of the MP-BPRAM version of bitonic sort on the MasPar.

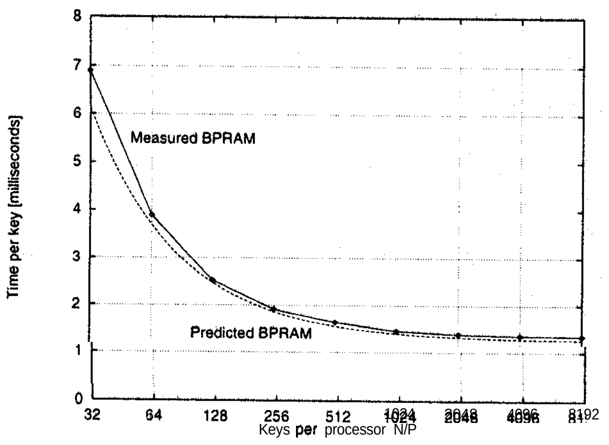

Figure 11: Measured and estimated times per key of bitonic sort on the GCel.

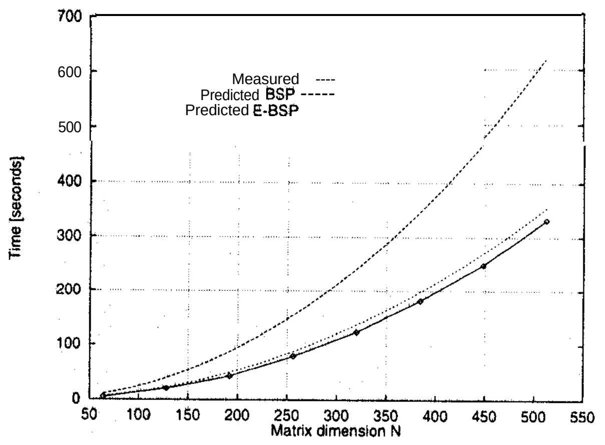

Figure 13: Predicted and measured execution times of the all pairs shortest path algorithm on the GCel.

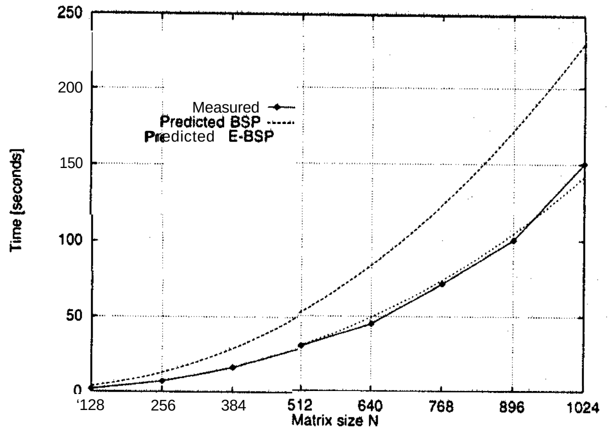

Figure 12: Predicted and measured execution times of the all pairs shortest path algorithm on the MasPar.

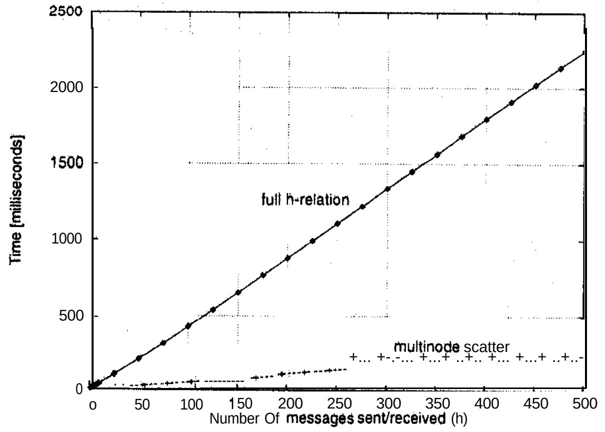

Figure 14: Comparison of the total times taken by full h-relations and by multinode scatter communication operations cm the GCel.

timated by the MP-BSP model shows that there is a significant error in the predictions, For example, at $N = 512$, the MP-BSP model predicts an execution time of 53.9 seconds but the measured time is 30.3 seconds; the prediction is off by 78% from the actual time. This defect is the resuits of unbalanced communication which is not captured by BSP (nor the MP-BPRAM). Also shown in Fig, 12 are the times estimated under the E-BSP model for the MasPar (using $T_{\mathrm{unb}}(P')$, cf. Section 4.4.1), and it verifies that the E-BSP model gives a much better estimation of the actual execution times,

On the GCel (Fig. 13), there is also a substantial error in the times predicted by BSP. To explain this behavior, we measured the time taken by a multinode scatter communication operation, in which $\sqrt{P}$ source processors scatter h data items across the remaining processors so that each of them receive at most $\lceil h/\sqrt{P}\rceil$ messages. This is the communication pattern that arises in the first superstep of the broadcast of the active row/column. The results of this experiment are shown in Fig. 14. A multinode scatter takes $g_{\mathrm{mscat}} \cdot h + L$ time, where $g_{\mathrm{mscat}} \approx 492$ μs. This communication pattern is up to a factor of 9.1 cheaper than a full h-relation, so a better estimate of the execution time is obtained by using g_{mscat} in the analysis of the first superstep. Fig. 13 demonstrates that this modification yields predictions that closely match the measured data points.

On the CM-5 (Fig. 15), the BSP model accurately predicts the actual running times of the all pairs shortest path algorithm. On this architecture, due to its large bisection bandwidth, there is only a minor difference between the time taken by a full h-relation and the time taken by a scatter operation.

6 Comparison Between the Models

This section concentrates on the second question: how do the models compare with each other? In this paper, we mainly investigate the gain that can be obtained by grouping data into a single long message. In [18], an example is given that demonstrates that by ignoring unbalanced communication the BSP model may incorrectly predict that one algorithm is superior to another.

As stated before, the maximum improvement that can be achieved by sending large blocks of data instead of many small packets is the ratio $g/(w \cdot u)$. In general, however, the

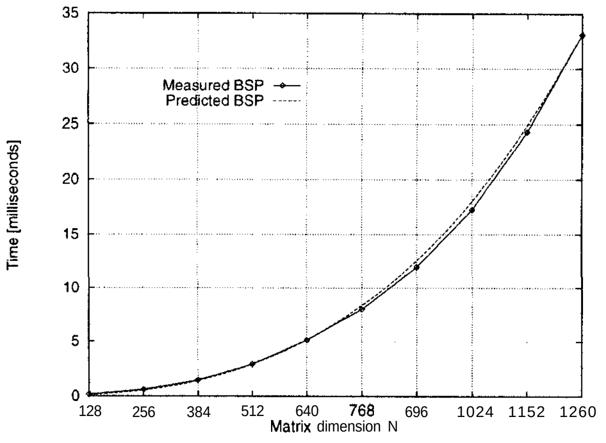

Figure 15: Predicted and measured execution times of the all pairs shortest path algorithm on the CM-5.

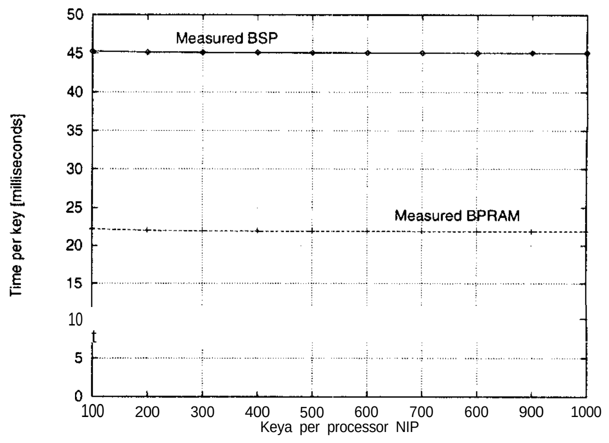

Figure 17: Comparison between the MP-BSP and MP-BPRAM versions of bitonic sort cm the MasPar.

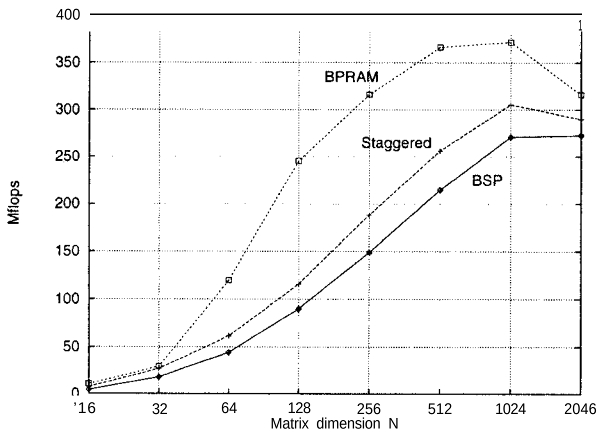

Figure 16: Comparison between the BSP and MP-BPRAM versions of the matrix multiply on the CM-5.

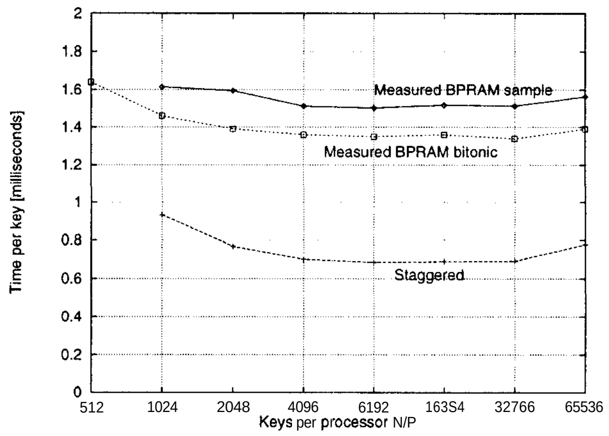

Figure 18: Measured times per key for the MP-BPRAM versions of bitonic sort and sample sort on the GCel.

performance enhancement for complete applications will not be this large. This is caused by the fact that in many parallel algorithms, the communication component in the total running time grows at a smaller rate than the computational time. Furthermore, in order to utilize block transfers the algorithm designer may have to adapt the algorithm which in fact may increase the running time.

Fig. 16 compares the performance of the MP-BPRAM and the (staggered) BSP variants of the matrix multiplication algorithm on the CM-5. Clearly, the MP-BPRAM version is faster than the versions that use fixed size short messages. For example, at N = 512, the measured performance is 366 Mflops for the long message version and 256 Mflops for the staggered BSP variant, corresponding to an improvement of 43%. This is an example of an algorithm in which the communication component grows at a smaller rate than the arithmetic time; the ratio $g/(w \cdot \sigma)$ is about 4.2 on this architecture but a similar overall performance improvement is not observed.

This is not the case in bitonic sort, since the local sorting step contributes very little to the total running time. A comparison of the two versions of bitonic sort on the MasPar is given in Fig. 17. For this algorithm-architecture combination, the performance enhancement obtained is a factor of about 2.1, whereas the maximum improvement is $(g+L)/(w \cdot \sigma)$ = 3.3.

On the GCel, there is a huge difference between sending short and long messages. The importance of capturing bulk transfer in a computational model for this architecture is reflected in bitonic sort (cf. Fig. 6 and Fig. 11). With 4K keys per processor, the measured time per key of the synchronized BSP version is 86.1. milliseconds, whereas the MP-BPRAM variation requires only 1.36 milliseconds per key. The MP-BPRAM version has almost two orders of magnitude improvement over the BSP version, which uses fixed size short messages. In our opinion, the high startup cost of a message transmission on this architecture makes it an absolute requirement that block transfers are included in the computational model.

Fig. 18 compares the running times of the MP-BPRAM versions of bitonic sort and sample sort on the GCel. The performance of sample sort is somewhat disappointing. Although it is the most efficient sorting algorithm in theory, it does not outperform bitonic sort. The reason is apparent. The send substep alone, in which the keys are routed to their appropriate buckets, requires about 16. u. w. N/P μs, and the cost of bitonic sort is dominated by an expression

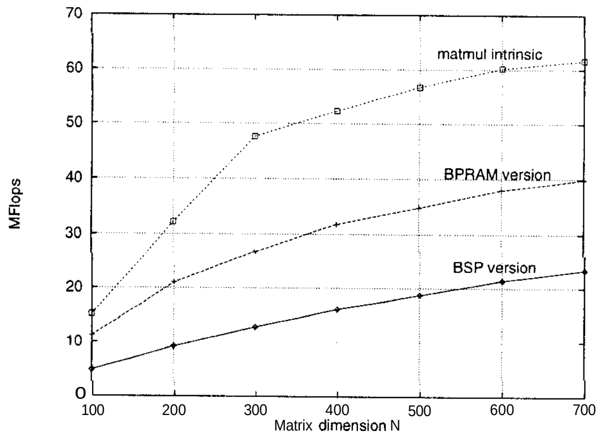

Figure 19: Comparing the performance of model derived matrix multiplication algorithms with the matmul intrinsic on the MasPar.

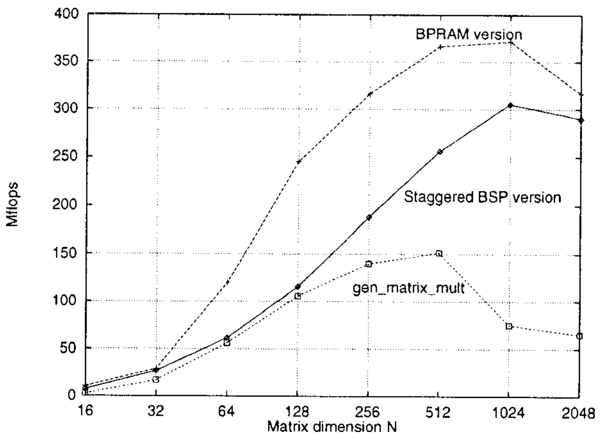

Figure 20: Comparing the performance of model derived matrix multiplication algorithms with the **gen_matrix_mult** ( ) routine present on the CM-5,

of the form 21 σ w N/P for P = 64. The large constant in the running time of the send substep is due to the restriction that a processor may send or receive at most one message in a single communication step. The lowest curve labeled 'Staggered' shows the measured times per key for an implementation in which each processor $\langle j \rangle$ packs the keys destined for the same processor in a single message and then send the message to the appropriate bucket, again in a staggered fashion to avoid stalls. This variation may violate the single-port communication restriction, but yields an improvement by a factor of approximately 2.

7 Model Implementations Versus Machine-Specific Implementations

In this section, we take the matrix multiplication algorithm as a case study to validate the efficiency of the model derived algorithms on two platforms; the MasPar and the CM-5. On the GCel, a highly optimized mathematical library was not available.

Fig. 19 compares the performance of the model derived matrix multiplication algorithms with the performance measured for the **matmul** intrinsic. Evidently, the intrinsic is more efficient than our implementations for all measured data points. At N = 700, the measured performance of the Mp-Bpram version is 39.9 Mflops and the **matmul** intrinsic achieves 61.7 Mflops (a 1K MasPar MP-1 system has a peak performance of 75 Mflops, single precision), which corresponds to a performance penalty of 35%. However, it is important to note that the intrinsic matmul is highly optimized and squeezes the highest performance from this architecture. In the light of this, the performance penalty incurred going through general computational models seems to be very acceptable.

A comparison of the model derived implementations and the **gen_matrix_mult** routine present in the Connection Machine Scientific Software Library (CMSSL) is given in Fig. 20. Surprisingly, the model versions are much faster than the implementation that uses **gen_matrix_mult**. The Mp-Bpram version peaks at 372 Mflops which is 65% of the peak performance (64 9 = 576 Mflops), but **gen_matrix_mult** never achieves more than 151 Mflops. It needs to be mentioned, however, that the implementations do not use the vector units. For example, if compiled for the vector-units model, **gen_matrix_mult** achieves 1016 Mflops at N = 512.

8 Conclusions

This paper presented many experimental results collected on three parallel platforms evaluating some of the proposed parallel computation models. Our work consisted of an evaluation part, a comparison part and an efficiency validation part.

In the evaluation part the models' predictions were compared with experimental results. Unlike previous studies, which mostly demonstrated a close match between the measured and predicted execution time, our work shows that there are situations in which the models do not accurately predict the actual running time, This occurred in the following circumstances:

- Under the Bsp model the communication schedule (the order in which the messages are sent and received) is irrelevant. However, on real machines, when all processors simultaneously send data to the same processor stalls will occur. This caused the Bsp model to overestimate the performance of the initial implementation of the matrix multiplication algorithm on the CM-5 by 21%. The LogP model [9] captures this aspect by assuming that the network has a finite capacity.

- Certain contention free communication patterns that occur frequently in practice require much less time than normally expected. For example, on the MasPar, the pattern that arises in bitonic sort is twice as fast as normally predicted.

- The Bsp model charges for a full h-relation, even if only a partial h-relation needs to realized. Similarly, the Mp-Bpram model assumes that in every communication step all processors send and receive a message. For example, on the MasPar, a partial permutation with 32 active PEs takes about 13% of the time taken by a full permutation. On the GCel, a multinode scatter is up to a factor of 9.1 cheaper than a full h-relation.

We especially believe that unbalanced communication is an aspect which should not be ignored. The point is that only high bandwidth networks such as binary hypercubes and fat trees have the property that partial h-relations take about the same time as full h-relations, but low bandwidth networks such as meshes do not. On this, practically important, topology sending h messages between two processors is up to a factor of $\Theta(\sqrt{P})$ (i.e., non-constant) cheaper than a full h-relation.

In the comparison part the MP-BPRAM and the BSP model were compared with each other. The gain that can be obtained by grouping data into a single long message depended on the algorithm and on the architecture. On the GCel, there is a huge difference (up to a factor of 120) between sending a few large messages instead of many small ones. The MasPar and the CM-5 support fine-grain communication and the maximum improvement that one can achieve is 3.3 and 4.2 respectively. On these architectures, a satisfactory performance can be obtained by using fixed size short messages, but larger than one computational word, as was done implicitly in [9]. For example, with 16-byte messages, the difference decreases to 1.37 on the MasPar and to 2.1 on the CM-5.

Lastly, there was an efficiency validation part in which we showed the effectiveness of deriving fast algorithms through the formalisms of the models by comparing the performance of our implementations with the performance of optimized library routines. The model derived algorithms either outperformed the vendor supplied routines (CM-5) or incurred a performance penalty of 35% (MasPar), which can be called acceptable. However, matrix multiplication might be called an embarrassingly parallel problem and it remains to be investigated whether acceptable performance can also be achieved for problems that are harder to parallelize.

Acknowledgments

Computational support was provided by the Interdisciplinary Center for Complex Computer facilities Amsterdam (IC^{3}A), Leiden University and by the National Center for Supercomputing Applications (NCSA) at Urbana-Champaign. The authors wish to thank Arno van Duin, Arjan Bik and the anonymous referees for their comments.

References

[1] A. Aggarwal, A.K. Chandra, and M. Snir. On Communication Latency in PRAM Computations. In *Proc. Symp. on Parallel Algorithms and Architectures*, pages 11–21. ACM, 1989.

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

[3] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms* Addison-Wesley, 1983.

[4] A. Alexandrov, M.F. Ionescu, K.E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model. In *Proc. 7th Symp. on Parallel Algorithms and Architectures.* ACM, 1995.

[5] K.E. Batcher. Sorting networks and their applications. In *Proc. AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.

[6] R.H. Bisseling and W.F. McColl. Scientific Computing on Bulk Synchronous Parallel Architectures. Technical Report 836, University of Utrecht, December 1993.

[7] G.E. Blelloch, C.E. Leiserson, B.M. Maggs, C.G. Plaxton, S.J. Smith, and M. Zagha. A Comparison of Sorting Algorithms for the Connection Machine CM-2. In *Proc. 3rd Symp. on Parallel Algorithms and Architectures*, pages 3–16. ACM, 1991.

[8] T Cheatham, A Fahmy, D.C. Stefanescu, and L G. Valiant. Bulk Synchronous Parallel Computing – A Paradigm for Transportable Software. In *Proc. 28th Hawaii Int. Conf. on System Science*, Jan. 1995.

[9] 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*, pages 1–12. ACM SIGPLAN, May 1993

[10] D.E. Culler, A. Dusseau, S.C Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel Programming in Split-C. In *Proc. Supercomputing*, Nov. 1993.

[11] D.E. Culler, A C Dusseau, R.P. Martin, and K.E. Schauser. Fast Parallel Sorting under LogP: from Theory to Practice. In T. Hey and J. Ferrante, editors, *Portability and Performance for Parallel Processing.* Wiley, 1994.

[12] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc. 10th Symp. on Theory of Computing*, pages 114–118. ACM, 1978.

[13] J.W. Hong and H.T. Kung. I/O complexity: The red-blue pebble game. In *Proc. 13th Symp. on Theory of Computing*, pages 326–333. ACM, May 1981.

[14] J. JáJá and K.W. Ryu. The Block Distributed Memory Model. Technical Report CS-TR-3207, UMIACS-TR-94-5, University of Maryland, 1994.

[15] B.H.H. Juurlink and H.A.G. Wijshoff. Experiences with a model for parallel computation. In *12th Symp. on Principles of Distributed Computing*, pages 87–96. ACM, August 1993.

[16] B.H.H. Juurlink and H.A.G. Wijshoff. Communication Primitives for BSP Computers, 1995. To appear in *Inf. Proc. Letters.*

[17] B.H.H. Juurlink and H.A.G. Wijshoff. The E-BSP Model: Incorporating Unbalanced Communication and General Locality into the BSP Model. Technical Report 95-44, Leiden University, 1995.

[18] B.H.H. Juurlink and H.A.G. Wijshoff. A Quantitative Comparison of Parallel Computation Models. Technical Report 96-01, Leiden University, 1996.

[19] W.F. McColl. General Purpose Parallel Computing. In A. Gibbons and P. Spirakis, editors, *Lectures on Parallel Computation*, chapter 13. Cambridge University Press, 1993.

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