ELSEVIER

Information Processing Letters 58 (1996) 303-310

Information Processing Letters

Communication primitives for BSP computers

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

Received 22 January 1996
Communicated by T. Lengauer

Keywords: Parallel algorithms; Parallel computation models; Communication operations

1. Introduction

Current programming practice for parallel computers is based on carefully matching a problem to the interconnection topology of the target architecture. As a result, parallel software is not portable in any serious sense. Therefore, a major goal in contemporary computer science is to identify an intermediate model of parallel computation that captures the essential features of most current and foreseeable architectures. Valiant has called a model of this kind a "bridging model" [10], and proposed the *Bulk-Synchronous Parallel* (BSP) model as a viable candidate for this role. The BSP model is based on two parameters g and L that characterize the hardware of a machine. $L \geqslant 1$ is the synchronization periodicity and $1 \leqslant g \leqslant L$ is the communication throughput ratio.

A systematic study of BSP algorithms remains to be done. Some examples of BSP algorithms are given in [4,5,8,10]. This paper extends that work by studying a set of communication primitives that have been found useful when implementing scientific and engineering applications. In particular, we consider the following basic problems:

- Single-item broadcast.
- k-item broadcast.
- Prefix.
- 2D prefix.
- Sorting P elements, where P is the number of processors.
- Duplication.

The problems are formally described at the head of each section.

This paper is organized as follows. Section 2 describes the BSP model and introduces some terminology used in the rest of the paper. In Section 3 we present BSP algorithms for the broadcast and reduction operations. In Section 4 we describe a sorting algorithm for the BSP model that achieves reasonable performance even if the number of values to be sorted equals the number of processors. A BSP algorithm for duplication is presented in Section 5, and concluding remarks are given in Section 6.

2. The model

The Bulk-Synchronous Parallel (BSP) model [10] (see also [11], where it is called the XPRAM) consists of the following attributes:

* Corresponding author. Email: benj@cs.leidenuniv.nl.

PII S0020-0190(96)00073-7

- A set of processor/memory pairs.
- A router or communication network that delivers messages point-to-point.
- Facilities to barrier synchronize all or a subset of the nodes.

A program written for a BSP computer operates in *supersteps*. In each superstep, each processor can perform local operations on data present in its local memory, send some messages and receive some messages. After each period of L time units, a global check is made to determine whether the processors have completed the current superstep. If they have, they proceed to the next superstep. Otherwise, the next period of L time units is devoted to the current superstep. The performance of a BSP computer is determined by the following parameters:

- The number of processors P.
- The synchronization periodicity L, which is the number of time units between successive barrier synchronizations.
- 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.

For simplicity, we will assume in this paper that every superstep is either purely computation or purely communication. A BSP program then consists of a sequence of computation phases, with the necessary communications taking place between phases. As long as a processor does not communicate with other processors, it can execute code independently from the other processors. When processors communicate, a processor may not know how many messages it is due to receive, and thus it cannot know whether it can proceed safely. Consequently, every communication phase must be followed by a barrier synchronization at which point the memory accesses take effect. A similar mechanism has been implemented in the CM-5 [7].

The cost of a superstep S is determined as follows. If S is a computation step, then the cost of S is simply the maximum number w of local computation steps executed by any processor. A communication superstep is charged as follows. Let an h-relation denote the communication pattern in which each processor sends and receives at most h messages. All messages are assumed to be of small constant size. A communication step is viewed as an h-relation and its cost is $g \cdot h + L$. (In the original BSP model, the cost of S is $\max\{w, L, g \cdot h\}$. Our cost definition is between one and three times the cost in the original model, and avoids the use of max functions in complexity analysis.)

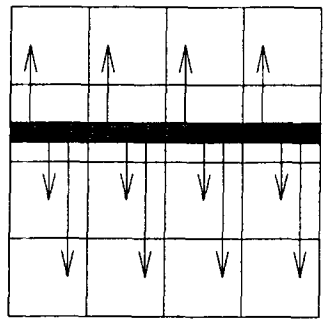

Fig. 1. Broadcast of the pivot row in LU decomposition.

3. Broadcast and reduction

Broadcast is an important primitive in many parallel applications. For example, broadcasting the pivot row to all other processors is a basic substep in LU decomposition (see Fig. 1). Some parallel architectures have dedicated hardware to support broadcasting, but the BSP model assumes no such capabilities and a broadcast operation must be realized using point-to-point message transmission. We consider the following broadcast operations:

- A *single-item broadcast* in which a single data item needs to be sent from a single source to all other processors.
- A *k-item broadcast* in which multiple data items need to be sent from a single source to all other processors.

A BSP algorithm for a single-item broadcast was described in [5]. The algorithm executes a d-ary tree, for some $d \geqslant 2$. The height of the tree is $\log_d P$. During the lth superstep, $0 \leqslant l < \log_d P$, the processors at level l of the tree make d copies of the broadcast item and send one copy to each of their children. The total communication time can be seen to be

$$O((g \cdot d + L) \cdot \log_d P).$$

The asymptotic complexity is minimized by setting $d = \min\{L/g, P\}$. The communication time will

then be [1]

$$O(L \cdot \log P / \log(L/g)).$$

A k-item broadcast can be implemented by performing k single-item broadcast operations in sequence. This approach, however, will not fully utilize the bandwidth of the communication system. A better strategy is to spread the broadcast items as equally as possible among the P processors before replicating each item. Let the broadcast items be denoted $m_0, m_1, \ldots, m_{k-1}$. We now describe a BSP algorithm for the k-item broadcast problem.

Algorithm 2D-BCAST(k, P).
There are two cases

(1) If $k \geqslant P$, then the items are transmitted to every other processor in two supersteps.
   i. Processor 0 keeps items $m_0, m_1, \ldots, m_{\lceil k/P \rceil - 1}$ to itself and sends items m_j, $\lceil k/P \rceil \leqslant j < k$, to the processor with ID $j \operatorname{div} \lceil k/P \rceil$. In this step, a $k - \lceil k/P \rceil < k$-relation needs to be realized. Time $O(g \cdot k + L)$ is needed for this.
   ii. Each processor makes $P - 1$ copies of every item it received in the previous superstep and sends a copy to every other processor. In this step, no processor receives more than k messages and each processor sends at most $\lceil k/P \rceil \cdot (P - 1) < 2 \cdot k$ messages, for a total BSP cost of $O(g \cdot k + L)$.

(2) If $k < P$, the algorithm consists of three phases. Without loss of generality, assume that $a = P/k$ is an integer. If a is not an integer, we can make it so by adding at most $k - 1$ dummy elements.
   i. Processor 0 transmits item m_j, $1 \leqslant j < k$, to the processor with ID $j \cdot a$. The cost of this step is $O(g \cdot k + L)$.
   ii. Processor $j \cdot a$, $0 \leqslant j < k$, replicates the element it just received to the processors $j \cdot a + 1, \ldots, (j + 1) \cdot a - 1$ by using the single-item broadcast operation. This step requires $O(L \cdot \log(P/k) / \log(L/g))$ time; see the analysis of a single-item broadcast.
   iii. Each processor j, $0 \leqslant j < P$, sends its item to each processor that is congruent to $j \pmod a$. This also takes $O(g \cdot k + L)$ time.

[1] We use the notation $\log x$ to denote $\max\{1, \log_2 x\}$.

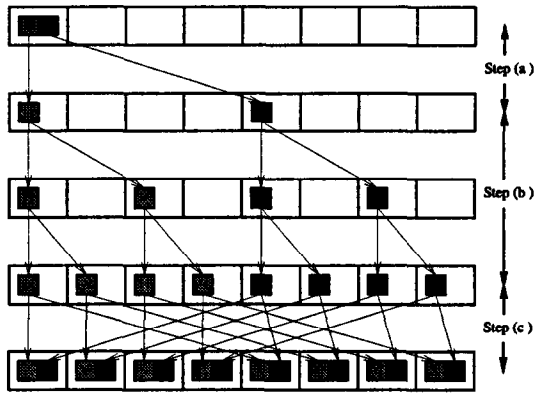

Fig. 2. k-item broadcast operation, $k < P$.

An example is given in Fig. 2. From the analysis accompanying the description of the algorithm it can be seen that in both cases the communication time is

$$O(g \cdot k + L \cdot \log P / \log(L/g)).$$

Note that this result is clearly optimal as $\Omega(g \cdot k + L)$ time is needed to disperse the k items from the source node, and $\Omega(L \cdot \log P / \log(L/g))$ time is needed for a single-item broadcast.

Let $\oplus$ be an associative operator over a domain D. The prefix sums (also called scan) problem is defined as follows. Given an array $A[0 : P - 1]$ such that processor i initially contains the element $A[i]$, it is required to compute the sums

$$S[i] = A[0] \oplus A[1] \oplus \cdots \oplus A[i]$$

for $i = 0, \ldots, P - 1$. The parallel prefix algorithm given in [10] can also be thought of as executing a d-ary tree for some $d \geqslant 2$. The algorithm proceeds in two stages, first upward from the leaves to the root and then back to the leaves. The entire algorithm has computation time

$$O(d \cdot \log_d P),$$

and communication time

$$O((g \cdot d + L) \cdot \log_d P).$$

Again, the running time is minimized by choosing $d = \min\{L/g, P\}$. The total BSP cost of a parallel prefix will then be

$$O(L \cdot \log P / \log(L/g)).$$

The 2D prefix operation applies the prefix computation to each row of a $k \times P$ matrix $A[0 : k-1, 0 : P-1]$, where each processor holds one column of the matrix. Many network implementations use pipelining to obtain an efficient 2D algorithm. With this technique, a new prefix operation is initiated at every round; while level l of the tree sends its results to level $l-1$, the partial sums of the next prefix are moved from level $l+1$ to level l. If this technique is applied to a P-processor BSP, the communication cost will be

$$\begin{aligned} &O((g \cdot d + L) \cdot (k + \log_d P)) \\ &\quad = O(L \cdot (k + \log P / \log(L/g))). \end{aligned}$$

In order to obtain an efficient 2D parallel prefix algorithm we will first transpose the $k \times P$ matrix A from row major order to column major order. For simplicity, we will assume that $k = 2^m$ for some integer m, although this is not essential. Thus either k divides P or P divides k. The algorithm is given next.

Algorithm 2D-PREFIX(k, P).
Again, there are two cases

(1) If $k \geq P$, then the algorithm consists of the following three steps.
   i. For all j, $0 \leq j < P$, processor j sends $A[i, j]$, $0 \leq i < k$, to the processor with ID $(i \cdot P + j) \operatorname{div} k$. After this step, processor i will contain all elements pertaining to the rows $i \cdot k/P, i \cdot k/P + 1, \ldots, (i+1) \cdot k/P - 1$. The BSP cost of this step is $O(g \cdot k + L)$.
   ii. Each processor performs a prefix operation on every row it contains. This step requires $O(k)$ local operations.
   iii. Route all prefix sums to the correct output location. I.e., let $S[i, j] = A[i, 0] \oplus A[i, 1] \oplus \cdots \oplus A[i, j]$ for $0 \leq i < k$ and $0 \leq j < P$. Send $S[i, j]$ to processor j. As step (i), this also takes $O(g \cdot k + L)$ time.

(2) If $k < P$, then there are six steps.
   i. For all j, $0 \leq j < P$, processor j sends $A[i, j]$, $0 \leq i < k$, to the processor with ID $(i \cdot P + j) \operatorname{div} k$. Let $q = P/k$, $x_i = i \operatorname{div} q$ and $y_i = i \bmod q$. After this step, processor i will contain the elements $\{A[r, t] \mid r = x_i \wedge y_i \cdot k \leq t < (y_i + 1) \cdot k\}$. The BSP cost of this step is $O(g \cdot k + L)$.
   ii. Each processor performs a prefix operation on the elements it contains. This step takes $O(k)$ local operations.
   iii. Let $A_i(j_1 : j_2)$ denote the partial sum $A[i, j_1] \oplus \cdots \oplus A[i, j_2]$. For every processor i, let $b_i = A_{x_i}(y_i \cdot k : (y_i + 1) \cdot k - 1)$. Execute a prefix operation on the b_is of each interval of P/k consecutive processors. This step requires $O(L \cdot \log(P/k) / \log(L/g))$ time.
   iv. Every processor i, unless its ID is congruent to $P/k - 1 \pmod{P/k}$, sends the result of the previous step to processor $i+1$. This takes $O(L)$ time.
   v. Every processor adds the element it just received (if any) to each element it contains. As Step (ii), this takes $O(k)$ local operations.
   vi. Route all prefix sums to their correct output location, which takes $O(g \cdot k + L)$ time.

If k is not a power of 2, then we can expand the matrix A to an $2^{\lceil \log k \rceil} \times P$ matrix B such that $B[i, j] = A[i, j]$ for $0 \leq i < 2^{\lceil \log k \rceil}$ and $0 \leq j < P$, and $B[i, j]$ is arbitrary otherwise. This will increase the number of elements in each processor by at most a factor of 2, and affects the running time only by a constant factor. From the analysis accompanying the description of algorithm 2D-PREFIX, it can be seen that the total BSP cost in both cases is

$$O(g \cdot k + L \cdot \log P / \log(L/g)),$$

which is clearly an improvement (by a factor of L/g) over the pipelined implementation described earlier. To see that this result is optimal consider the copy operator defined as $x \oplus y = x$ for all $x, y \in D$. The 2D prefix operation with the copy operator can be used to broadcast k items from processor 0 to all other processors. Thus, a lower bound for the k-item broadcast problem implies a lower bound for the 2D prefix operation.

4. Sorting

Many parallel sorting algorithms rely upon efficient algorithms for sorting P keys. Algorithms for sorting N keys on a P-processor BSP computer were presented before in [5,10]. These algorithms are optimal when N is sufficiently larger than P, but for the case $N = P$ they do not perform very well. In this section we

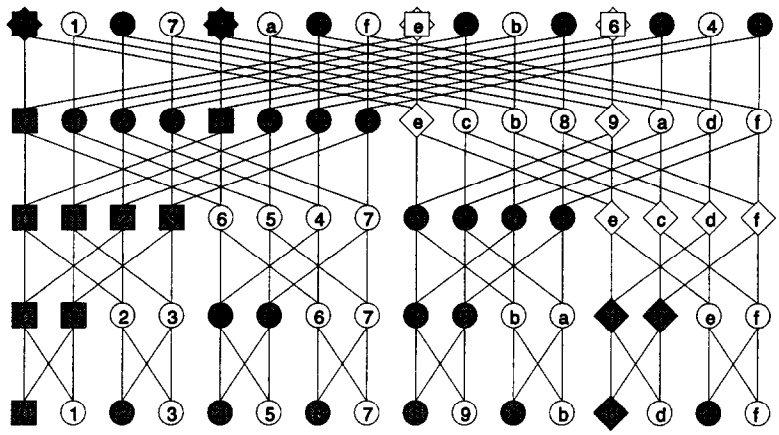

Fig. 3. Simulating a merge stage on a P-processor BSP computer, where $P = 16$ and $L/g = 4$. A shaded node designates a comparator in which the minimum of two keys is selected. Nodes in the shape of a box are simulated by processor 0. Processor 12 simulates the nodes that have the shape of a diamond.

describe a BSP sorting algorithm based on Batcher's bitonic sort [2] that achieves reasonable performance even if the number of keys to be sorted equals the number of processors. Bitonic sort performs $\log P$ merge stages, and the ith merge stage, $1 \leqslant i \leqslant \log P$, consists of i merge steps. The communication structure in each merge stage resembles a butterfly of $\log P$ levels and P columns. A direct implementation of bitonic sort on a P-processor BSP computer takes $O(L \cdot \log^2 P)$ time. However, when L is large compared to g as in most current parallel architectures, this is not a very efficient solution.

To obtain a faster sorting algorithm the following strategy is applied. In each merge stage, $O(\log(L/g))$ consecutive levels of comparators are melted together. To simulate these levels each processor has to send and receive $O(L/g)$ messages, which can be done in one superstep with communication cost $O(L)$. Furthermore, the computation cost in each superstep is bounded by $O(L/g)$; each processor has to simulate $O(L/g)$ comparators from the first of $O(\log(L/g))$ consecutive levels, $O(L/(2 \cdot g))$ from the second level and so on. An example is given in Fig. 3. The described procedure reduces the depth of each merge stage by a factor of $O(\log(L/g))$. Therefore, the total BSP complexity is given by

$$O(L \cdot \log^2 P/\log(L/g)).$$

Note that most comparators are simulated by several processors. This might seem redundant at first since the same communication cost can be achieved by observing that a butterfly of $\log P$ levels and P columns can be represented by $\log P/\log(L/g)$ stages, where each stage consists of $P/(L/g)$ independent butterfly graphs of $\log(L/g)$ levels and L/g columns (see for example [1]). In this case, only $g \cdot P/L$ processors are used. However, each processor now has to simulate $(L/g) \cdot \log(L/g)$ comparators in each superstep and the communication time $\Theta(L)$ will not dominate the computation cost $\Theta((L/g) \cdot \log(L/g))$ unless $g \geqslant \log(L/g)$.

5. Duplication

Given N items $d_1, d_2, \ldots, d_N$, initially equally distributed over P processors. The duplicate operation produces $d_i.c$ copies of each item d_i, where $d_i.c$ is a tag associated with d_i. Let $M = \sum_{i=1}^{N} d_i.c$ be the resulting total number of elements and let M_i denote the sum of the tags of the items in processor i, $0 \leqslant i < P$. We will require that after duplication each processor contains M/P elements. This operation was first introduced by Schwartz [9] and was also considered in [3].

The basic idea of our algorithm is to equally distribute the work among the P processors such that each processor has to make M/P duplicates. Without loss of generality we will assume that M/P is an in-

Processor ID	0	1	2	3	4	5	6	7
Contents	$(d_1,5)$	$(d_3,6)$	$(d_5,2)$	$(d_7,6)$	$(d_9,1)$	$(d_{11},1)$	$(d_{13},1)$	$(d_{15},5)$
	$(d_2,3)$	$(d_4,1)$	$(d_6,1)$	$(d_8,3)$	$(d_{10},1)$	$(d_{12},1)$	$(d_{14},1)$	$(d_{16},2)$
M_i	8	7	3	9	2	2	2	7
$\sum_{j=0}^{i} M_j$	8	15	18	27	29	31	33	40
l_i	0	1	3	3	5	5	6	6
r_i	1	2	3	5	5	6	6	7
After	$(d_1,5)$	$(d_2,3)$	$(d_3,4)$	$(d_5,2)$	$(d_7,4)$	$(d_8,2)$	$(d_{12},1)$	$(d_{15},3)$
distribution		$(d_3,2)$	$(d_4,1)$	$(d_6,1)$	$(d_8,1)$	$(d_9,1)$	$(d_{13},1)$	$(d_{16},2)$
				$(d_7,2)$		$(d_{10},1)$	$(d_{14},1)$	
						$(d_{11},1)$	$(d_{15},2)$	

Fig. 4. Illustrating the steps of the duplication algorithm.

teger. If M/P is not an integer, we can make it so by adding exactly one dummy item. The algorithm combines techniques from [3] and [6].

For $0 \leqslant i < P$, let l_i and r_i be integers such that $l_i \cdot M/P \leqslant \sum_{j=0}^{i-1} M_j < (l_i+1) \cdot M/P$ and $r_i \cdot M/P < \sum_{j=0}^{i} M_j \leqslant (r_i+1) \cdot M/P$. Processor i will distribute its items over $l_i,\ldots,r_i$. An example is given in Fig. 4 in which $P = 8$, $N = 16$ and $M = 40$. Note that it may be necessary to split an item into several items, which might increase the number of messages sent or received by any processor. However, no processor will receive more than M/P messages.

Algorithm DUPLICATE(P).

(1) Each processor i, $0 \leqslant i < P$, computes M_i. This step takes $O(N/P)$ local operations.

(2) For each i, $0 \leqslant i < P$, compute l_i, r_i and M/P. This can be done in $O(L \cdot \log P/\log(L/g))$ time using three applications of the prefix operation.

(3) Let $i_0, i_1,\ldots,i_k$ be the processors such that $l_{i_j}+1 \leqslant r_{i_j} - 1$. In this step, processor i_j distributes its items over $l_{i_j}+1,\ldots,r_{i_j}-1$ such that each of these processors will have to make M/P duplicates. This will be done in the following five steps.

i. Let $q_{i_j} = r_{i_j} - l_{i_j} - 1$ and note that $q_{i_j} \cdot M/P \leqslant M_{i_j}$. For a set of items S, let $\text{sum}(S) = \sum_{d \in S} d.c$. In this step, each i_j forms a set of items S_{i_j} such that $\text{sum}(S_{i_j}) = q_{i_j} \cdot M/P$. This can be done as follows. Starting with an empty set, i_j adds items to S_{i_j} until $\text{sum}(S_{i_j})$ is equal to or greater than $q_{i_j} \cdot M/P$. If $\text{sum}(S_{i_j}) > q_{i_j} \cdot M/P$, then the last item added to S_{i_j} can be split into two such that one of them makes $\text{sum}(S_{i_j})$ equal to $q_{i_j} \cdot M/P$. This will increase the number of items in i_j by at most one. This step takes $O(N/P)$ local operations.

ii. Each processor i_j sends the items belonging to S_{i_j} to $l_{i_j}+1$. Because $|S_{i_j}| \leqslant N/P$, the BSP cost of this step is $O(g \cdot N/P + L)$.

iii. Processor $l_{i_j}+1$ broadcasts the $O(N/P)$ items it has just received to the processors $l_{i_j}+1,\ldots,r_{i_j}-1$. This is an $O(N/P)$-item broadcast operation and requires $O(g \cdot N/P + L \cdot \log P/\log(L/g))$ time.

iv. Each processor $l_{i_j}+1$ broadcasts the value $l_{i_j}+1$ to the processors $l_{i_j}+1,\ldots,r_{i_j}-1$. This step takes $O(L \cdot \log P/\log(L/g))$ time.

v. Using the value received in the previous step, each processor can determine the elements that were destined for it in $O(N/P)$ local operations. The other elements can be discarded.

(4) Each processor i distributes its remaining items over l_{i_j} and r_{i_j} such that l_{i_j} and r_{i_j} will have to make $(l_i+1) \cdot M/P - \sum_{j=0}^{i-1} M_j$ and $\sum_{j=0}^{i} M_j - r_i \cdot M/P$ duplicates, respectively. Again, this may require splitting an item into two. In this step, each processor will receive at most M/P messages and no processor sends more than $N/P - 1$ messages. Therefore, the communication cost of this step is $O(g \cdot M/P + L)$.

The total BSP cost can be seen to be

$$O(g \cdot M/P + L \cdot \log P/\log(L/g)),$$

regardless of the M_is.

Problem	BSP Complexity	Implementation Architecture	Running time
Single-item broadcast /	$O(L \cdot \log P / \log(L/g))$	Hypercube	$O(\log^2 P / \log\log P)$
Prefix		Butterfly	$O(\log^2 P)$
		Mesh	$O(\sqrt{P} \cdot \log P)$
k-item broadcast /	$O(g \cdot k + L \cdot \log P / \log(L/g))$	Hypercube	$O(k + \log^2 P / \log\log P)$
2D prefix		Butterfly	$O(k \cdot \log P + \log^2 P)$
		Mesh	$O(k \cdot \sqrt{P} + \sqrt{P} \cdot \log P)$
Sorting	$O(L \cdot \log^2 P / \log(L/g))$	Hypercube	$O(\log^3 P / \log\log P)$
		Butterfly	$O(\log^3 P)$
		Mesh	$O(\sqrt{P} \cdot \log^2 P)$
Duplication	$O(g \cdot M/P + L \cdot \log P / \log(L/g))$	Hypercube	$O(M/P + \log^2 P / \log\log P)$
		Butterfly	$O(M/P \cdot \log P + \log^2 P)$
		Mesh	$O(M/\sqrt{P} + \sqrt{P} \cdot \log P)$

Fig. 5. Table of results.

6. Concluding remarks

The values of g and L that can be obtained in practice depend on (1) the network topology, (2) the routing algorithm and (3) technological factors such as the channel capacities and the network cycle time. In asymptotic terms, the following values for g and L can be expected.

Network	g	L
Hypercube	$\Theta(1)$	$\Theta(\log P)$
Butterfly	$\Theta(\log P)$	$\Theta(\log P)$
Mesh	$\Theta(\sqrt{P})$	$\Theta(\sqrt{P})$

Here, we assume a hypercube with all-port communication, i.e., in one step the processors can send a message to all their neighbors. We will now substitute these typical values of g and L in the BSP complexity of the operations considered in this paper in order to obtain more concrete results. The result of that is summarized in Fig. 5.

In comparison, it is possible to perform a single-item broadcast/prefix operation on a P-processor hypercube in $O(\log P)$ time. In this case, there is a gap of $\log P / \log\log P$ between the network implementation and the BSP implementation. In general, a single communication step of the network model can be simulated on the BSP model in time $\leqslant L$. On a hypercube, the k-item broadcast/2D prefix implementation given in this paper takes $O(k + \log^2 P / \log\log P)$ time. On the other hand, k prefix operations can be implemented to run in $O(k + \log P)$ time on a P-processor hypercube. Thus, the BSP implementation is optimal (up to a constant factor) provided that $k \geqslant c \cdot \log^2 P / \log\log P$ for some constant c. For the butterfly and mesh families of networks, the algorithms given in this paper do not perform very well. This is because $L/g = \Theta(1)$ on these types of architectures. It should be noted also that for most problems we have considered the single element per processor case. If these problems are scaled appropriately, an increase in performance will result.

Some generalities are beginning to emerge. For example, in some cases (broadcast, sorting) we have employed the fact that L/g-relations can be realized essentially as fast as 1-relations. Another important observation is that techniques that work well on network models of parallel computation, such as pipelining prefix computations on the hypercube and shuffle-exchange families of networks, do not behave equally well on the BSP model. This is because these computations only require neighbor communication on these types of networks, whereas the BSP model does not differentiate between nearby and remote processors.

Acknowledgments

The authors wish to acknowledge the anonymous referee who suggested the modified version of bitonic sort.

References

[1] A. Aggarwal, A.K. Chandra and M. Snir, On communication latency in PRAM computations, in: *Proc. Symp. on Parallel Algorithms and Architectures* (1989) 11–21.

[2] K.E. Batcher, Sorting networks and their applications, in: *Proc. AFIPS Spring Joint Computer Conf.* (1968) 307–314.

[3] Y. Ben-Asher, D. Egozi and A. Schuster, 2-D SIMD algorithms for perfect shuffle networks, *J. Parallel Distributed Comput.* **16** (1992) 250–257.

[4] R.H. Bisseling and W.F. McColl, Scientific computing on bulk synchronous parallel architectures, Tech. Rept. 836, University of Utrecht, 1993.

[5] A.V. Gerbessiotis and L.G. Valiant, Direct bulk-synchronous parallel algorithms, in: O. Nurmi and E. Ukkonen, eds., *Proc. 3rd Scandinavian Workshop on Algorithm Theory*, Lecture Notes in Computer Science **621** (Springer, Berlin, 1992) 1–18.

[6] J. JáJá and K.W. Ryu, Load balancing and routing on the hypercube and related networks, *J. Parallel Distributed Comput.* **14** (1992) 431–435.

[7] C.E. Leiserson, Z.S. Abuhamdeh, D.C. Douglas et al., The network architecture of the connection machine CM-5, in: *Proc. 4th Symp. on Parallel Algorithms and Architectures* (1992) 272–285.

[8] W.F. McColl, Scalable parallel computing: A grand unified theory and its practical development, in: B. Pehrson and I. Simon, eds., *Proc. 13th IFIP World Computer Congress* (Elsevier, Amsterdam, 1994).

[9] J.T. Schwartz, Ultracomputers, *ACM Trans. Programming Languages Systems* **2** (1980) 484–521.

[10] L.G. Valiant, A bridging model for parallel computation, *Comm. ACM* **33** (8) (1990).

[11] L.G. Valiant, General purpose parallel architectures, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science* (Elsevier, Amsterdam, 1990).