The E-BSP Model: Incorporating General Locality and Unbalanced Communication into the BSP Model

Ben H.H. Juurlink and 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. The BSP model was proposed as a step towards general purpose parallel computing. This paper introduces the E-BSP model that extends the BSP model in two ways. First, it provides a way to deal with unbalanced communication patterns, i.e., communication patterns in which the amount of data sent or received by each processor is different. Second, it adds a notion of general locality to the BSP model where the delay of a remote memory access depends on the relative location of the processors in the interconnection network. We use our model to develop several algorithms that improve upon algorithms derived under the BSP model.

1 Introduction

It has been stressed by many authors that the emergence of one or a few computational models is essential to the progress of parallel computing [9, 14], because it enables the programmer to write architecture-independent software. Such a model should strike a balance between simplicity of usage and reflectivity of existing parallel architectures. Despite many efforts, no consensus has been reached on which model should be used. One model that has gained some acceptance is the Bulk-Synchronous Parallel (BSP) model proposed by Valiant [15].

A BSP computer consists of a number of processors connected by a point-to-point message router. Furthermore, it includes a mechanism to barrier synchronize the nodes. Computations on the BSP model are organized in a series of supersteps, with synchronization taking place between supersteps. The performance of a BSP computer depends on three parameters: the number of processors P, the synchronization cost / communication latency s and the computational to communication throughput ratio g. Typical values for g and s are $g = s = \Theta(\sqrt{P})$ for meshes, $g = \Theta(1)$ and $s = \Theta(\log P)$ for hypercubes, and $g = s = \Theta(\log P)$ for hypercube-derivative networks such as butterflies.

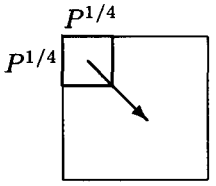
This paper introduces the *Extended* BSP or E-BSP model that extends the BSP model in two ways. First, it provides a method to deal with unbalanced communication patterns. Under the BSP model, the cost of communication depends on the largest amount of data sent or received by any processor. In many situations, this will be a large overestimate. Consider for example a personalized broadcast in which one processor sends $P - 1$ distinct items, each to a different destination. Under the BSP model, the cost of this operation is $\Theta(g \cdot P + s)$. However, it can be seen that on any network that contains a Hamiltonian path a personalized broadcast can be implemented in $\Theta(P)$ time.

The second extension we propose adds a notion of general locality (or network proximity) to the BSP model. The BSP model only distinguishes between local and non-local

memory access. This kind of locality has been termed "strict locality" by Heywood and Ranka [4]. The E-BSP model also represents "neighborhood locality" where the delay depends on the relative location of the processors in the interconnection network. As an example, consider a permutation routing problem on a binary tree. Under the BSP model, $\Theta(P)$ time is charged for this communication operation, as it must conservatively assume that all packets must pass through the root of the tree. In reality, the performance depends very much on the locality of the communication because some messages may be routed within subtrees without soaking up bandwidth near the root.

Related Work. De la Torre and Kruskal [2] considered a model called the Y-PRAM. This model represents communication locality by assuming that the architecture is recursively decomposable into halves. Informally, this means that it can partition itself into independent subsystems with the same structure as the whole network. Meshes and hypercubes have this property, but for example butterfly and shuffle-exchange networks do not. The E-BSP model does not demand this property.

Communication cost on the Y-PRAM depends on two functions $\delta(P')$ and $\beta(P')$ representing the latency and bandwidth inefficiency (the ratio of P' to the bandwidth) of the architecture. The time taken by a P'-processor Y-PRAM to make a total of M memory accesses is $\Theta(\beta(P') \cdot M/P' + \delta(P') + m)$, where m is the maximum number of accesses made by any processor. Thus, the Y-PRAM also models unbalanced communication, but the cost metric is too optimistic as the following example shows. Consider a communication operation on the mesh in which the $\sqrt{P}$ processors in the upper left corner each send $\sqrt{P} - 1$ messages to distinct processors outside this region.



Then, $P - \sqrt{P}$ messages need to cross the boundary of $2 \cdot P^{1/4}$ edges. It follows that the time required is at least $P^{3/4}/2 - P^{1/4}/2 = \Theta(P^{3/4})$. However, under the Y-PRAM, $\Theta(\sqrt{P})$ time would be charged for this operation.

Organization. Section 2 describes the E-BSP model and the metric used to determine the complexity of algorithms. In Section 3 we apply the E-BSP model to the linear array and mesh interconnection networks. Section 4 describes a collection of algorithms for some basic communication primitives that occur frequently in practice, such as broadcasting data from one processor to every other processor. In Section 5 we develop algorithms for regular grid problems and matrix multiplication. Concluding remarks are given in Section 6.

2 The E-BSP Model

Like a BSP computer, the E-BSP model consists of the following attributes: (1) a set of processor/memory nodes, (2) a point-to-point message router, and (3) facilities to synchronize the nodes. We assume a barrier style synchronization, similar to the mechanism described in [15]. More precisely, a computation consists of a sequence of *supersteps*,

with synchronization taking place between supersteps. In each superstep, a processor can perform local operations on data present in its local memory, send some messages and (implicitly) receive some messages. To simplify presentation, we assume that every superstep is either purely computation or purely communication.

The only difference between BSP and E-BSP is the cost of a communication superstep. Let an h-relation denote the communication pattern in which each processor sends and receives at most h messages. Under the BSP model, all communication supersteps are viewed as h-relations and their cost is modeled by the formula $g \cdot h + s$. In order to be generally applicable, the BSP model conservatively assumes that all h-relations are *full* h-relations in which each processor sends and receives *exactly* h messages. As argued in the introduction, in many situations this will turn out to be a large overestimate. To deal with unbalanced communication we view each communication superstep as an (M, k_1, k_2)-relation [11], which is a more general form of communication and which in many instances reflects practical situations better than the routing of h-relations.

Let $[P]$ denote the set $\{0, 1, \ldots, P-1\}$ of processor IDs. A routing problem is described in terms of a *routing distribution*. We formulate this notion as follows.

Definition 1. A routing distribution $\mathcal{R}$ is a bag of pairs taken from the set $[P] \times [P]$. Each pair specifies the source and the destination of one packet. A routing problem is the problem of simultaneously routing the packets of a given distribution.

Using this definition we are able to give a concise description of an (M, k_1, k_2)-relation, and of a routing problem with locality L.

Definition 2. A routing distribution $\mathcal{R} = \langle (s,d) | s, d \in [P] \rangle$ is an (M, k_1, k_2)-relation if $\#\mathcal{R} \leq M$, and for all $t \in [P]$: $\#\langle (s,d) \in \mathcal{R} | s = t \rangle \leq k_1$ and $\#\langle (s,d) \in \mathcal{R} | d = t \rangle \leq k_2$. A routing distribution $\mathcal{R}$ has locality L if for all $(s,d) \in \mathcal{R}$, $|d - s| \leq L$.

Note that a full h-relation is just a special instance of an (M, k_1, k_2)-relation with $M = h \cdot P$ and $k_1 = k_2 = h$. In Section 3, we determine the cost of routing (M, k_1, k_2)-relations and of distributions with locality L for the linear array and mesh networks.

At this point it is important to make the distinction between the locality ℓ of a routing distribution on the underlying implementation architecture and the locality L on the E-BSP model. The first is usually defined as the maximum distance a packet has to travel, e.g., in a mesh it is the Manhattan distance between the source and the destination and in a hypercube it is the Hamming distance. The locality L of a routing distribution depends on the processor numbering. We illustrate this for the mesh. If we number the nodes in the mesh snake-like row-major (see Fig. 1(a)), we have the property that if the locality on the E-BSP model is L, then the locality on the underlying architecture is $\ell = L$. Fortunately, a better indexing scheme exists. Consider the Peano indexing scheme [3, 8], which numbers the nodes as illustrated in Fig. 1(b) for a 4×4 mesh. This scheme has the property that any two nodes whose indices differ by d are at physical distance $\leq 3 \cdot \sqrt{d}$ in the mesh [1]. Hence, if the locality of a routing distribution is L, then the locality on the underlying architecture is $\ell = O(\sqrt{L})$.

Note that although communication locality depends on the processor numbering, the programmer does not need to be aware of any specific interconnection topology. The knowledge that communication is faster when the processors are "close by" contains information itself.

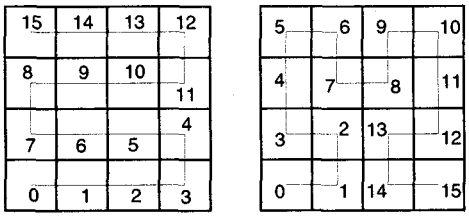


Fig. 1. Mesh indexing schemes: (a) snake-like row-major, (b) Peano indexing scheme.

3 Implementation Architectures

In this section, we apply the E-BSP model to the linear array and mesh networks. The full version of this paper [5] also considers fat trees.

We consider store-and-forward packet routing, and assume that in one time step the processors can send packets to all adjacent nodes. The processors in the linear array are indexed using the natural processor numbering and the Peano indexing scheme is used for the mesh. We write $T_{\text{unb}}(M, k_1, k_2)$ to denote the routing time of (M, k_1, k_2)-relations, $T_{\text{rwl}}(h, L)$ signifies the routing time of h-relations with locality L, and the time taken by an (M, k_1, k_2)-relation with locality L is denoted by $T_{\text{r}}(M, k_1, k_2, L)$. Further, we let $k_{\min} = \min\{k_1, k_2\}$ and $k_{\max} = \max\{k_1, k_2\}$.

Linear Array. A linear array is characterized by having both a large communication latency as well as a small communication bandwidth. An arbitrary h-relation requires $\Theta(h \cdot P)$ time on this network, and the synchronization cost is $s = \Theta(P)$. As such, it does not fit in very well with the BSP model, which does not distinguish between general and strict locality. The following theorem shows that realizing an (M, k_1, k_2)-relation with locality L requires substantially less time than performing a $k_{\max}$-relation if either the packets are not distributed evenly, or when the packets only need to travel a short distance. The proof of this theorem can be found in [5].

Theorem 3. *An (M, k_1, k_2)-relation with locality L can be performed on a P-processor linear array in $T_r(M, k_1, k_2, L) = \min\{k_{\min} \cdot L, M\} + P - 2$ time.*

We give two examples. The routing time of an h-relation in which each packet travels a distance of at most d, is $\Theta(h \cdot d + P)$. Under the BSP model, $\Theta(h \cdot P)$ time is charged for this h-relation. Second, the BSP model charges $\Theta(P^2)$ time for a personalized broadcast. Under the E-BSP model, $\Theta(P)$ time suffices.

Meshes. Sibeyn and Kaufmann [13] studied the (k_1, k_2)-routing problem on meshes. This is essentially the same as an (M, k_1, k_2)-relation, except that the total number of packets M is assumed to be equal to $k_{\min} \cdot P$. But of course, their results can be directly applied to the (M, k_1, k_2)-routing problem.

Lemma 4. *(M, k_1, k_2)-relations can be routed in $O(\sqrt{k_1 \cdot k_2} \cdot \sqrt{P})$ time on a $\sqrt{P} \times \sqrt{P}$ mesh.*

Routing with locality on meshes was considered in [12]. There, it was shown that if the maximal distance ℓ a packet has to go in either direction is known to all processors,

then h-relations can be performed in $O(h \cdot \ell)$ time. By allowing an extra number of $O(\sqrt{P})$ packet steps to determine the locality of a routing distribution, and by using the previous result together with the Peano indexing scheme, we get:

Theorem 5. *(M, k_1, k_2)-relations with locality L can be routed in $T_r(M, k_1, k_2, L) = O(\min\{\sqrt{k_1 \cdot k_2} \cdot \sqrt{P}, k_{\max} \cdot \sqrt{L} + \sqrt{P}\})$ time on a $\sqrt{P} \times \sqrt{P}$ mesh.*

4 Communication Primitives

In this section we use the E-BSP model to develop several algorithms for some basic communication primitives. We start by considering the single item broadcast operation in which a single data item needs to be sent from processor 0 to every other processor. This can be implemented by executing a broadcast tree, as illustrated in Fig. 2. Specifically, let $q_1, q_2, \ldots, q_d$ be integers such that $q_i > 0$ and $\prod_{i=1}^{d} q_i = P$. The algorithm consists of d supersteps. In the first superstep, processor 0 sends the item to the processors $P/q_1, 2 \cdot P/q_1, \ldots, P - P/q_1$. During the second superstep, the processors $i \cdot P/q_1$, $0 \leq i < q_1$, send their item to the processors $i \cdot P/q_1 + j \cdot P/(q_1 \cdot q_2)$ for $j = 1, 2, \ldots, q_2 - 1$, and so on. Note that the nodes in the broadcast tree can have different fan-outs. Under the BSP model, the nodes in the optimal broadcast tree have identical fan-out. It can be seen that the sketched algorithm has communication time $\sum_{i=1}^{d} T_r(q_1 \cdot q_2 \cdots q_i, q_i, 1, q_i \cdot q_{i+1} \cdots q_d)$.

In order to obtain more concrete results, we need to fix the q_i's. On a linear array, the cost of the ith superstep is $O(\min\{q_1 \cdot q_2 \cdots q_i, q_i \cdot q_{i+1} \cdots q_d\} + P)$. By choosing $q_1 = P$, only one superstep is required and the running time becomes $O(P)$, which is clearly optimal.

On the mesh, we slightly deviate from the algorithm described above. In this case, processor 0 first sends the broadcast item to the processors $1, 2, \ldots, P^{1/3} - 1$. This corresponds to a $(P^{1/3})$-relation with locality $P^{1/3}$ which can be performed in $O(\sqrt{P})$ time on the mesh. After that, there is a redistribution phase in which processor i, $1 \leq i < P^{1/3}$, sends its copy to processor $i \cdot P^{2/3}$. This superstep is a permutation routing and also takes $O(\sqrt{P})$ time. This process is repeated: each processor $i \cdot P^{2/3}$ replicates its copy to the processors $i \cdot P^{2/3} + j$ for $1 \leq j < P^{1/3}$, and again the copies are redistributed so that the processors $i \cdot P^{1/3}$, $0 \leq i < P^{2/3}$, each receive a copy. The broadcast is complete when each processor $i \cdot P^{1/3}$ broadcasts its copy to the processors $i \cdot P^{1/3} + j$

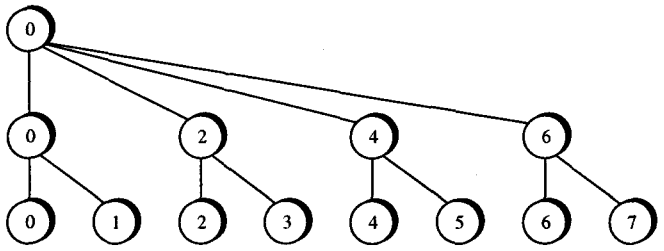


Fig. 2. Broadcast tree for $P = 8$, $d = 2$, $q_1 = 4$ and $q_2 = 2$.

for $1 \leq j < P^{1/3}$. It can be verified that the entire algorithm runs in the optimal $O(\sqrt{P})$ communication time.

In comparison, the BSP algorithm for this problem takes $\Theta(\log P \cdot P)$ time on a linear array and $\Theta(\log P \cdot \sqrt{P})$ time on a mesh. In both cases, the E-BSP algorithm reduces the cost by a factor of $\log P$.

The next data movement operation we consider is the N-item broadcast operation in which one processor contains N items that need to be sent to every other processor. We distinguish between the case $N < P$ and $N \geq P$. If $N < P$, the algorithm consists of 4 phases. In the first phase, the items are distributed among the processors $0, P/N, 2 \cdot P/N, \ldots, N - P/N$. This takes $T_{unb}(N, N, 1)$ time. In the second phase, each processor $i \cdot P/N$ broadcast its item to the processors $i \cdot P/N, i \cdot P/N + 1, \ldots, (i + 1) \cdot P/N - 1$ using the single item broadcast operation. Let the broadcast items be denoted by $D[0], D[1], \ldots, D[N-1]$. During the third phase, the items are permuted so that processor i receives the item $D[i \bmod N]$. After this permutation, every group of N consecutively numbered processors collectively contain all items. The fourth and final phase consists of $\log N$ supersteps, and in superstep i, $0 \leq i < \log N$, every processor sends the 2^i items in its possession to the processor whose ID is identical except in the $(\log N - i)$th bit. The trick with this butterfly-like scheme is that most of the communication takes place between processors which are "close by".

If $N \geq P$, the second and third phase can be omitted. In this case, the final phase consists of $\log P$ supersteps, and in superstep i, $0 \leq i < \log P$, a $(2^i \cdot N/P)$-relation with locality $L = P/2^{i+1}$ needs to be performed. By substituting the appropriate cost functions, we obtain the following result.

Theorem 6. *If $N < P$, an N-item broadcast can be implemented in $O(P \cdot \log N)$ time on a linear array, and in $O(\sqrt{N} \cdot \sqrt{P})$ time on a mesh. If $N \geq P$, then it can be solved in $O(N \cdot \log P)$ time on a linear array, and in $O(N)$ time on a mesh.*

In contrast, the BSP algorithms for this problem take $O(g \cdot N + s)$ time if $N \geq P$, and $O(g \cdot N + s \cdot \log P / \log(s/g))$ time if $N < P$ [7]. We give two examples. For the mesh, the first bound yields $O(N \cdot \sqrt{P})$, whereas the E-BSP cost is $O(N)$. A speedup by a factor of $\sqrt{P}$ is attained. On the linear array, the second bound yields $O(P \cdot N)$ and we achieve a speedup of $N/\log N$ w.r.t. the BSP time.

Another useful broadcasting operation is when each processor has N items that need to be broadcast to all other processors. This operation is referred to as an all-to-all broadcast or gossip. It can be implemented efficiently on the E-BSP model using a *pipelining* strategy. Let B_i denote the block of N items initially stored in processor i. There are two stages, a forward and a backward stage, and each stage consists of $P - 1$ supersteps numbered $0, 1, \ldots, P - 2$. During superstep i of the forward stage, each processor j, $i \leq j \leq P - 2$, sends the block B_{j-i} to processor $j + 1$. Similarly, in superstep i of the backward stage, each processor j, $1 \leq j \leq P - i - 1$, sends the block B_{j+i} to processor $j - 1$. Hence, all communications have locality 1 on the E-BSP model. This gives:

Theorem 7. *An all-to-all broadcast operation with initially N elements residing in each processor can be performed in $O(P \cdot (N + P))$ time on a linear array and in $O(P \cdot (N + \sqrt{P}))$ time on a mesh.*

The BSP algorithm for this problem takes $\Theta(g \cdot P \cdot N)$ communication time. On both networks, the cost is reduced by a factor of g provided N is sufficiently larger than P.

5 Grid Problems and Matrix Multiplication

As a good example of a problem with a high degree of locality, consider a finite difference application, e.g., Laplace operator. At every time step $t+1$, each element of the matrix A needs to be updated according to

$$A_{t+1}[i,j] = f(A_t[i,j], A_t[i-1,j], A_t[i+1,j], A_t[i,j-1], A_t[i,j+1]) \quad (1)$$

The usual (and for the BSP model optimal within a small constant factor) solution divides the matrix A into P square blocks of size $N/\sqrt{P} \times N/\sqrt{P}$ (see Fig. 3(a)). Each processor is assigned the task of updating the elements of one block of the matrix. Suppose that the computation of f involves 4 arithmetic operations and that the computation consists of m iterations, then the BSP cost of the algorithm is $4 \cdot m \cdot N^2/P + 4 \cdot g \cdot m \cdot N/\sqrt{P} + m \cdot s$. It can be seen that the blocks can be mapped onto a $\sqrt{P} \times \sqrt{P}$ mesh such that the computation only involves neighbor-to-neighbor communication. For other networks this may not be possible. However, if we divide A into slices of size $N \times N/P$ as in Fig. 3(b), then the communication has locality 1 on the E-BSP model, but a node needs to send more messages ($2 \cdot N$ instead of $4 \cdot N/\sqrt{P}$). We take a partitioning approach somewhere in between. Let $q_1 \cdot q_2 = P$. The matrix A is tiled into P blocks A_{ij}, $0 \leq i \leq q_1 - 1$ and $0 \leq j < q_2 - 1$, of size $N/q_1 \times N/q_2$ each (Fig. 3(c)). The subblock A_{ij} is assigned to the processor with ID $i \cdot q_2 + j$. Each iteration now involves two communication supersteps: a $(2 \cdot N/q_1)$-relation with locality 1, and a $(2 \cdot N/q_2)$-relation with locality q_2. We have shown:

Theorem 8. *m steps of Equation (1) can be computed in $4 \cdot m \cdot N^2/P + m \cdot T_{rwl}(2 \cdot N/q_1, 1) + m \cdot T_{rwl}(2 \cdot N/q_2, q_2)$ time on a P-processor* E-BSP.

The resulting communication time is determined by the relative values of q_1 and q_2. On the linear array, the communication time in each iteration is $T_{\text{comm}} = O(N/q_1 + N + P)$. In this case, the cost is independent (within a small constant factor) of the choice for q_1 and q_2. On the mesh, we have $T_{\text{comm}} = O(N/q_1 + N/\sqrt{q_2} + \sqrt{P})$ and the communication time is minimized by setting $q_1 \simeq P^{1/3}$ and $q_2 \simeq P^{2/3}$. This gives $T_{\text{comm}} = O(N/P^{1/3} + \sqrt{P})$.

A similar strategy can be used for multiplying two $N \times N$ matrices $C = A \cdot B$. Each of these matrices is partitioned into P submatrices A_{ij}, B_{ij} and C_{ij}, $0 \leq i < q_1$ and $0 \leq j < q_2$, of size $N/q_1 \times N/q_2$ each, where $q_1 \cdot q_2 = P$. The processor with ID

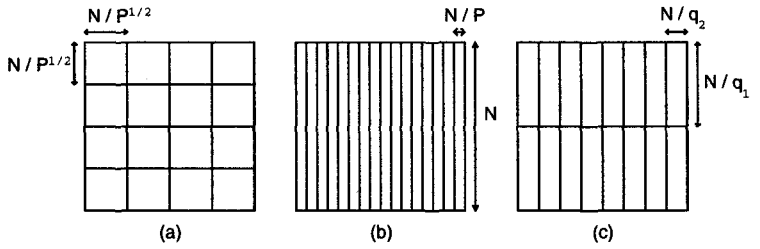


Fig. 3. Data partitioning: (a) Blocked, (b) Sliced, (c) Hybrid.

$i \cdot q_2 + j$ is assigned the task of computing C_{ij}. Then, each processor needs to receive N^2/q_1 elements pertaining to N/q_1 rows of A and N^2/q_2 elements corresponding to N/q_2 columns of B.

Let $M = N^2/P$. The submatrix A_{ij} needs to be sent to the processors with IDs $i \cdot q_2, i \cdot q_2 + 1, \ldots, (i+1) \cdot q_2 - 1$. This is done by a similar strategy as employed in the algorithm for the all-to-all broadcast operation. This step can therefore be performed in $O(q_2 \cdot (M + P))$ time on the linear array and in $O(q_2 \cdot (M + \sqrt{P}))$ time on the mesh.

The broadcast of the B submatrices is treated similarly, but in this case the processors that communicate with each other are not numbered consecutively. Instead, their indices differ by q_2. It can be verified that this step can be performed in $O(q_1 \cdot (q_2 \cdot M + P))$ time on a linear array and in $O(q_1 \cdot (\sqrt{q_2} \cdot M + \sqrt{P}))$ time on a mesh.

Now every processor contains the elements it needs and each submatrix can be computed in $O(N^3/P)$ computation time. By choosing q_1, q_2 such that the total communication time is minimized, we get:

Theorem 9. *Two $N \times N$ matrices, $N \geq P$, can be multiplied in $O(N^3/P)$ computation time, and in $O(N^2)$ communication time on a linear array and in $O(N^2/P^{1/3})$ communication time on a mesh.*

Proof. For the linear array, let $q_1 = P$ and $q_2 = 1$. For the mesh, take $q_1 = P^{1/3}$ and $q_2 = P^{2/3}$ □

The optimal BSP algorithm for multiplying two $N \times N$ matrices (using $(+, \times)$ only) requires $O(N^3/P)$ computation time and $O(g \cdot N^2/P^{2/3} + s)$ communication time (see for example [10]). Our algorithm reduces the communication complexity by a factor of $P^{1/3}$ if the underlying implementation architecture is a linear array and by a factor of $P^{1/6}$ on the mesh.

6 Summary and Concluding Remarks

In this paper we investigated the possibility of augmenting the BSP model with locality and unbalanced communication. It was shown that in many cases (M, k_1, k_2)-routing and routing with locality reflects practical situations better than the routing of h-relations. Although results were mainly given for the linear array and for the mesh, we believe that our observations apply to other networks as well. If the network has a large bandwidth and a small diameter (for example a full-port hypercube), then unbalanced communication and locality can probably be ignored. However, small bandwidth networks such as the practically important mesh are modeled less accurately by the BSP model.

It can be argued that overestimation of the actual runtime is acceptable as long as the relative performance of two algorithms is correctly predicted. However, the BSP model sometimes wrongly predicts that one algorithm is superior to another. Consider for example the single-item broadcast on a linear array. Under the BSP model the optimal solution consists of $\log P$ supersteps, and in each superstep each processor containing a copy of the broadcast item sends it to a processor which has not previously received a copy. The cost of this algorithm is $\Theta(P \cdot \log P)$. The one-level algorithm in which the source processor sends $P - 1$ messages, one to every other processor, has a BSP cost of $\Theta(P^2)$. In reality, however, the runtime of this algorithm is only $\Theta(P)$ packet steps.

We acknowledge that the E-BSP model trades simplicity for efficiency. The model complicates the design and analysis of parallel algorithms more than the BSP model. This is caused by two reasons. First, the BSP model allows to ignore mapping issues whereas the E-BSP model does not. Secondly, the routing time cannot always be expressed as a simple linear function of h. Therefore, one could look at a model that charges $\Theta(g_1 \cdot M/P + g_2 \cdot h + s)$ time for a communication superstep, where M is the total number of messages being routed and h is the maximum number of messages sent or received by any processor.

Experimental data verifying the results of this paper are given in [6].

Acknowledgments. We would like to thank Marinus Veldhorst and the anonymous referees for their comments.

References

1. G. Chochia, M. Cole, and T. Heywood. Implementation of the Hierarchical PRAM on the 2D Mesh: Analysis and Experiments. In *Symp. on Parallel and Distributed Processing*. IEEE, Oct. 1995.
2. P. de la Torre and C.P. Kruskal. Towards a Single Model of Efficient Computation in Real Parallel Machines. In *PARLE '91*, pages 7–24. Springer, 1991. LNCS 505.
3. T. Heywood and C. Leopold. Dynamic Randomized Simulation of Hierarchical PRAMs on Meshes. In *Aizu International Symposium on Parallel Algorithm/Architecture Synthesis*, March 1995.
4. T. Heywood and S. Ranka. A Practical Hierarchical Model of Parallel Computation I: The Model. *Journal of Parallel and Distributed Computing*, 16:212–232, 1992.
5. 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.
6. B.H.H. Juurlink and H.A.G. Wijshoff. A Quantitative Comparison of Parallel Computation Models. In *Proc. 8th Symp. on Parallel Algorithms and Architectures*, 1996. To appear.
7. B.H.H. Juurlink and H.A.G. Wijshoff. Communication Primitives for BSP Computers. *Inf. Proc. Letters*, 1996. To appear.
8. C. Kaklamanis and G. Persiano. Branch-and-Bound and Backtrack Search on Mesh-Connected Arrays of Processors. In *Proc. 4th Symp. on Parallel Algorithms and Architectures*, pages 118–126. ACM, 1992.
9. W.F. McColl. General Purpose Parallel Computing. In A. Gibbons and P. Spirakis, editors, *Lectures on Parallel Computation*, chapter 13. Cambridge University Press, 1993.
10. W.F. McColl. Scalable Computing. In *Computer Science Today: Recent Trends and Developments*. Springer, 1995. LNCS 1000.
11. D. Peleg and E. Upfal. The Generalized Packet Routing Problem. *Theoretical Comp. Sci.*, 53:281–293, 1987.
12. J.F. Sibeyn. *Algorithms for Routing on Meshes*. PhD thesis, Utrecht University, 1992.
13. J.F. Sibeyn and M. Kaufmann. Deterministic 1–k Routing on Meshes, With Applications to Worm-Hole Routing. Technical Report MPI-I-93-163, Max-Planck-Institut für Informatik, 1993.
14. H.J. Siegel, S. Abraham, and W.L. Bain et al. Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing. *Journal of Parallel and Distributed Computing*, 16:199–211, 1992.
15. L.G. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8), 1990.