Experiences with a model for parallel computation

B.H.H. Juurlink and H.A.G. Wijshoff
High Performance Computing Division
Department of Computer Science
Leiden University
P.O. Box 9512, 2300 RA Leiden
benj@cs.leidenuniv.nl and harryw@cs.leidenuniv.nl

Abstract

In this paper we study the practical viability of the BSP model of parallel computation as proposed by Valiant. This model is intended for simulating the often considered PRAM model on more realistic parallel computers with a fixed interconnection network. One of the main attributes of the BSP model is randomized routing. From experimentation on an existing parallel architecture, analytic models are derived which characterize the efficiency of this routing scheme. This characterization leads to the identification of the bottlenecks involved in building a parallel architecture in which the BSP model can efficiently be embedded.

1 Introduction

For parallel software to become widely used, it is clearly vital to separate parallel hardware issues from parallel software issues. Details of interprocessor communication, memory management and synchronization should be hidden from the programmer, since they depend on actual implementations in hardware. The Parallel Random Access Machine (PRAM) represents an ideal communication free architecture, where these low-level architectural features are hidden from the programmer. Unfortunately, a PRAM with a large number of processors is not likely to be built, because each processor must have its own physical path to the common memory. On the other hand, parallel computers with a fixed interconnection network cause algorithm development to be extremely difficult and dependent on the specific interconnection network implemented. A compromise between these two extremes would be formed by an efficient PRAM simulation on a network of processors.

To bridge this gap between what a general purpose parallel computer *should be* able to do and what they are able to do, Valiant [7, 8] introduced the *bulk-synchronous* model of parallel computation (BSP computer, for short), or XPRAM. In this model there does not exist a shared global memory, but instead each processor is attached to a local memory unit. In addition, there is a communication network that enables the processors to access the memories of other processors. Valiant showed that the BSP model can simulate an idealized PRAM in optimal time, provided the algorithm designed for the PRAM contains sufficient "parallel slackness". This means that the algorithm is written for v virtual parallel processors, but should run on p physical processors, where v is rather larger than p (e.g. $v \geq p \log p$). The extra parallelism in the algorithms is used to schedule and pipeline computation and communication efficiently. The basic task of the communication subsystem or *router* is to realize h-relations, i.e. communication patterns in which each processor sends and receives exactly h messages. To achieve optimal BSP simulation the router must realize h-relations in $O(h)$ time, for all h greater than some h_0. Valiant has shown that routing schemes and network topologies exist which can realize an h-relation

12th ACM Symposium on Principles on Distributed Computing, Ithaca NY
© 1993 ACM 0-89791-613-1/93/0008/0087....$1.50

in $O(h + \log p)$[1] time with overwhelming probability, where p is the number of processors.

In this paper we review the ideas of Valiant and discuss the trade-offs involved when embedding the BSP model into realistic parallel architectures. This paper is organized as follows. In section 2 we describe the randomized routing scheme, together with the experimentation conducted on a T800 transputer system. From this experimentation analytic models are derived which characterize the efficiency of the model. The results are analyzed and consequences for building a parallel architecture which would allow efficient execution under the BSP model are described in section 3. Concluding remarks are given in section 4.

2 Scalable routing

Routing determines the efficiency of interprocessor communication, thereby affecting overall system performance. Moreover, specific interconnection networks require different routing strategies to be utilized efficiently. This hampers the fitness of these networks for general purpose parallel computing. In order to be suited for general purpose parallel computers, the communication network must satisfy the following requirements:

1. The effective throughput of the communication network (the total number of packets delivered per second by the router) must scale linearly with the number of nodes.

2. The delay through the network should increase slowly with the number of nodes.

For our purposes, we consider a network as a graph $G = (V, E)$, where the vertices V correspond to processors and the edges E correspond to communication links. One class of interconnection networks that has been proposed as multiprocessor interconnection networks and that is known to have good communication capabilities is the binary hypercube. The hypercube network is defined as follows. Let $N = 2^n$. The n-dimensional hypercube is a graph $G = (V, E)$, where $V = \{0, 1, \ldots, N-1\}$ and $E = \{(v, v^{(i)}) | v \in V, 0 \leq i \leq n-1\}$. Here $v^{(i)}$ denotes the number obtained by complementing the $(i+1)$-th least significant bit in the binary representation of v. Thus, for example, if $v = 3$ then $v^{(0)} = 2$, $v^{(1)} = 1$ and $v^{(2)} = 7$.

A fundamental routing problem is that of realizing a *permutation*, in which each processor sends and receives exactly one packet. In [6], Valiant proposed a randomized routing algorithm that, when realizing an arbitrary permutation on an N-node hypercube, terminates within $c \log N$ routing steps with high probability (e.g. with probability greater than $1 - 2^{-cn}$). Experiments by Brebner [1] indicate that the constant c is close to two. Their analysis can also be used to show that for a random input permutation, the "obvious" algorithm in which a packet is repeatedly transmitted along an edge for which a bit in the destination address differs from the current node, also terminates in $O(\log N)$ routing steps with high probability. Here the probability is defined on the space of possible input permutations. They define a routing step as the time needed to transmit a packet along a link. So, local computations are not counted. Moreover, their run-time analysis assumes that a processor can transmit along all of its links simultaneously. However, this model, which is generally accepted in literature, neglects one important delay component: *the intra-node delay* (called intra-node routing latency in [2]). The intra-node delay is defined as the delay from when a packet is first received at an input link until it is ready for transmission at an output link. We will present experimental results that show that the intra-node delay is of dominating influence on the overall system performance.

While $O(\log N)$ is optimal for permutation routing on an N-node hypercube, it is not sufficient for constructing scalable general purpose architectures. To achieve optimal BSP simulation, the router must realize h-relations in $O(h)$ routing steps, for all h greater than some h_0. For this we can use the fact that the proposed randomized routing scheme can realize an h-relation in $O(h + \log N) = O(h)$ if $h \geq \log N$ routing steps with high probability.

2.1 The Valiant-Brebner algorithm

In the Valiant-Brebner algorithm the network topology is the n-dimensional hypercube. The algorithm consists of two phases run consecutively. In the first phase each packet is sent to a randomly chosen node

[1] All logarithms in this paper are base 2.

and in the second phase each packet is routed to its correct destination. The first phase is added to eliminate *hot-spots*, which commonly arise when several packets collide at the same link. We quote the following results

Theorem 1 ([6]) *For any constant S there is a constant C such that, when realizing a permutation on the n-dimensional hypercube, both phases of the algorithm finish within Cn routing steps with probability greater than $1 - 2^{-Sn}$.*

Theorem 2 ([8]) *For all $h > 0$, there exist constants $\alpha_0, \beta, \delta > 0$, such that, when realizing an h-relation on the n-dimensional hypercube, both phases of the algorithm finish within αh, $\alpha > \alpha_0$, routing steps with probability greater than $1 - N^{\delta} 2^{-\alpha\beta h}$.*

The notion of time needs further explanation. First, it is assumed that each node can communicate on all of its links concurrently. This assumption reflects the architecture of the transputer. Secondly, local computations are not counted for. However, under the assumption that routing in each node is determined by just one routing circuit or the CPU, the routing circuitry or CPU has to be shared when several packets arrive at the same time (to perform the routing function), which causes extra delays. In a hypercube network as many as n packets, where n is the number of dimensions, can arrive at the same time. It follows that the time needed to determine for each packet which link should be used is proportional to n.

Simulation results by Brebner [1] indicate that, when realizing an h-relation on an n-dimensional cube, both phases finish in $h + \log N$ routing steps. This is clearly optimal as h steps are necessary to disperse the h packets from the source node and an additional $\log N$ steps are required to route the packets to their destination. However, in our case this result is not directly applicable because of two reasons. First, we have only implemented the second phase of the Valiant-Brebner algorithm, which can be shown to have the same time complexity if the h-relations are generated randomly. Secondly, because experimentation was done on a T800 transputer system which does not comprise separate routing logic, we have separated the application network from the communication network. This reduces the CPU time used by the communication subsystem to manage packet transfer at the cost of adding two steps to the route each packet must follow. In a practical system this is desirable.

The algorithm was tested for randomly generated h-relations using serial simulation. The simulation results indicate that the number of routing steps $r(n, h)$ needed by the algorithm to perform an h-relation on an n-dimensional hypercube approximately follows the rule

$$r(n, h) = 1.22h + 1.6n.$$

Thus $r(n, h)$ grows with h but with a constant of proportionality somewhat larger than one. We have also established an estimate for the intra-node routing delays. Clearly, as each node has to perform the routing function for each packet that is received on one of its input ports, the total time spent in intra-node routing is proportional to the number of packets that has to pass through a single node. The simulation results indicate that this number can be estimated by

$$d(n, h) = (1.1 + 0.7n)h,$$

indicating that the expected traffic through one node per time interval does increase with n. Thus we may assume that the time $T(n, m, h)$ needed for realizing an h-relation on an n-dimensional hypercube with packets containing m bytes can be estimated by

$$T(n, m, h) = (1.22h + 1.6n)\frac{m}{B} + (1.1 + 0.7n)ht_d,$$

where B is the transmission bandwidth and t_d is the intra-node routing delay (the time needed to perform the routing function). In general this will be an overestimate, since data transfer is an autonomous DMA process which hardly consumes any CPU time.

2.2 Global design of the router

In this section we discuss the implementation of the Valiant-Brebner algorithm on a transputer system consisting of 16 T800-20MHz transputers. All code has been written in OCCAM-2. Each transputer has 1 Megabytes of local memory and four bi-directional links which can be used to exchange data with other transputers in the network. The links are rated at 20 Megabits per second, but the effective transfer rate is less because 3 control bits are transmitted for every 8 bits of data. Due to acknowledgements, the effective transfer rate of one channel is less when both

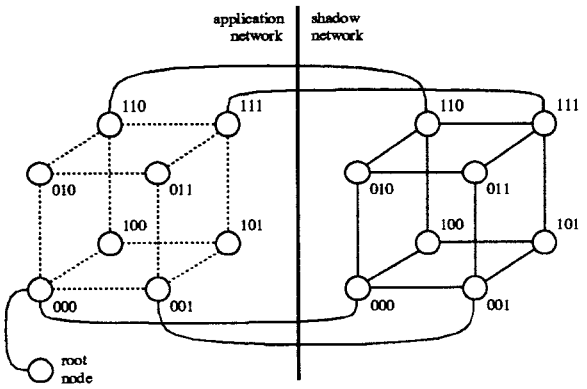

Figure 1: Configuration of the network, four links are not shown.

channels of a link are used concurrently. A simple experiment using two transputers connected by one link yielded that the effective transfer rate of one channel is $B = 0.625$ Megabytes per second.

Obviously, the CPU time used by the communication subsystem must be minimized so that the CPU availability for the application process is as high as possible. On the other hand, the application process should not slow down communication unnecessarily. In view of these considerations we allow an application to use only half of the transputers of the network, while the other half of the network, the *shadow* or *communication* network, is dedicated to routing. Figure 1 illustrates the configuration of the network. The shadow network forms a 3-dimensional hypercube and, in addition, each application node is connected to a node in the communication network.

The structure of the router is shown in figure 2, where arrows represent unidirectional channels. Let T be the identification number of this transputer and let link i ($i = 0, 1, 2$) be connected to the transputer with number $T^{(i)}$. Each of the three input processes waits for incoming packets at one of the input links. If a packet arrives, its destination address is inspected in order to determine which link the packet should be forwarded to. If the destination address matches the current node, the packet is added to the local output queue. Otherwise, a random element of the set of dimensions in which the destination address differs from the current node is chosen. This packet is added to the corresponding output queue. Packets from the appli-

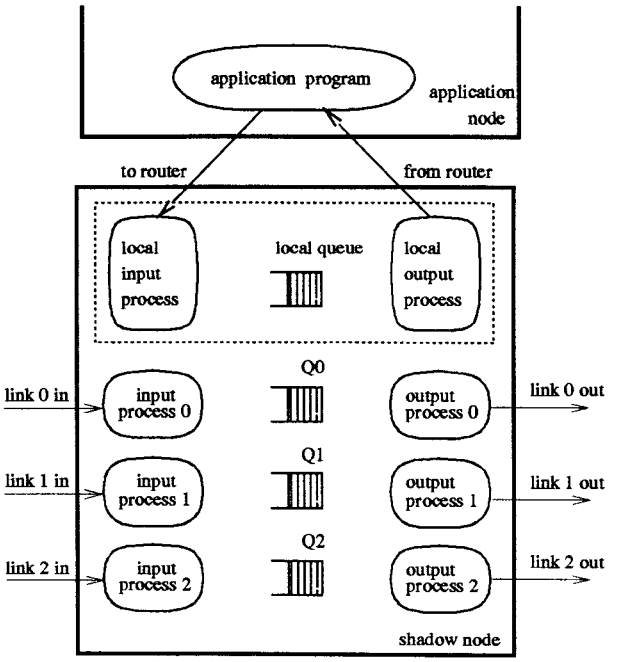

Figure 2: The structure of the router.

cation program are received by the local input process, which has the same functionality as the other input processes. Packets are passed in FIFO order. Output processes are associated with every output queue, transfering the packets to the corresponding output links.

In our experiments, we generate a random h-relation and measure the time taken by the router to transmit all packets to their correct destination. This experiment is repeated twenty times and the average is taken, to rule out experimental error as much as possible.

The test program executed on each application node consists of two processes *send* and *receive*. The send process terminates when it has transmitted h packets along the channel `to.router` and the receive process terminates when it has received h packets from the channel `from.router`. The form of the test program is as follows:

```
SEQ i = 0 FOR test.size
  SEQ
    generate.h.relation()  -- Generate a random
    synchronize()          -- h-relation
    -- Start timer
    -- Now run the send and receive process
```

```
-- in parallel. This call will return when
-- both processes have terminated.
PAR
  send(to.router)
  receive(from.router)
synchronize()
-- Stop timer and store completion time
```

Note that not only the time needed for routing all packets through the network is recorded, but also the time required for synchronization. However, with increasing h the time needed for synchronizing becomes less important and is in fact negligible. For example, for all $h \geq 50$ synchronization consumes less than 8 percent of the completion time.

Synchronization on an N-node hypercube network is realized in $2\log(N)$ steps by the following standard technique. A virtual binary tree of depth $\log(N)$ is mapped onto the hypercube. A synchronization packet of one unit is passed upwards in this tree starting at the leaves. A non-leaf node sends a packet after receiving packets from its successors in the tree. After $\log(N)$ routing steps the root node has received the synchronization packet. At this point the root processor sends a packet of one unit downwards in the tree informing all nodes that they have been synchronized.

2.3 Experimental results

In section 2.1 we found by simulation that the time needed for realizing an h-relation on an n-dimensional hypercube with packets containing m bytes could be estimated by

$$T(n,m,h) = (1.22h + 1.6n)\frac{m}{B} + (1.1 + 0.7n)ht_d \ ,$$

where B is the link speed (0.625 megabyte per second for the transputer system we used for the experiments) and t_d is the intra-node routing delay. To establish the correctness of this approximation, four cases of packet length are considered: 8, 16, 32 and 64-byte packets. Clearly, if packet size increases, then the transmission time of a packet grows proportionally while the intra-node delay stays approximately the same. Thus, we identify two experiments: to determine $T(n,m,h)$ as a function of n and h for a given packet size, and to determine $T(n,m,h)$ as a function of the packet length m. The parameter t_d has been obtained by running simple test programs. These measurements yielded $t_d = 14.5\mu$s. Based on this value our model predicts (in microseconds)

$$\begin{aligned} T(2,8,h) &= 51.9h + 41 \ , \\ T(2,16,h) &= 67.5h + 82 \ , \\ T(2,32,h) &= 98.8h + 164 \ , \\ T(2,64,h) &= 161.2h + 328 \ , \end{aligned}$$

and,

$$\begin{aligned} T(3,8,h) &= 62.0h + 61 \ , \\ T(3,16,h) &= 77.6h + 123 \ , \\ T(3,32,h) &= 108.9h + 246 \ , \\ T(3,64,h) &= 171.3h + 492 \ . \end{aligned}$$

In figure 3 the average completion time of the test program for the 2-dimensional hypercube and the 3-dimensional hypercube are shown. All timing results are given in microseconds.

The graphs clearly indicate that for a fixed network size the average completion time can be expressed as $ah+b$. A least-squares fit on the actual timings allows an approximation of the constants a and b. In doing so the following results were obtained

$$\begin{aligned} T(2,8,h) &= 42.2h + 183 \ , \\ T(2,16,h) &= 55.8h + 227 \ , \\ T(2,32,h) &= 88.7h + 348 \ , \\ T(2,64,h) &= 149.0h + 534 \ , \end{aligned}$$

and,

$$\begin{aligned} T(3,8,h) &= 53.4h + 255 \ , \\ T(3,16,h) &= 83.1h + 372 \ , \\ T(3,32,h) &= 102.5h + 506 \ , \\ T(3,64,h) &= 157.3h + 747 \ , \end{aligned}$$

which are close to our estimates. Note that for packets containing 8 or 16 bytes, there is a significant difference between routing on a 2-dimensional and routing on a 3-dimensional hypercube. This significant difference can be explained by the fact that the time needed to perform the routing function (approximately 14.5 μs) is almost the same as the packet transfer time (12.8 μs and 25.6 μs respectively). Thus for these cases the completion time is dominated by the intra-node routing delay. For 32 and 64-byte packets, the relative difference between routing on a 2- and routing on a 3-dimensional hypercube is much less, because the intra-node delay is significantly smaller than the transmission time of a packet.

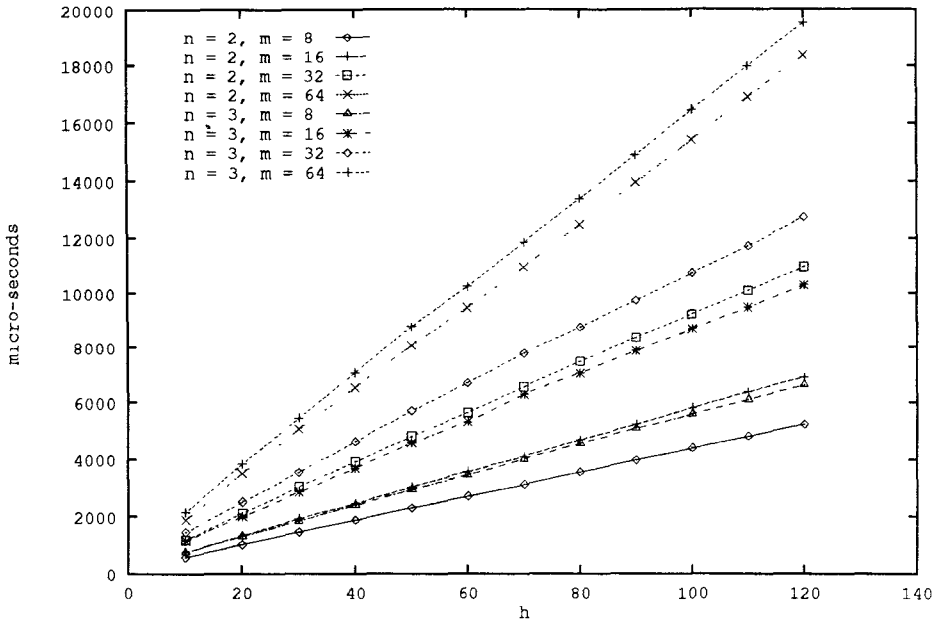

Figure 3: Experimental results of the router. The plots show that the completion time of the test program grows linearly with h.

Since packets are addressed randomly in the test program, the average number of packets that has to pass through a single connection can be determined by multiplying the average number of links traversed by the number of packets in the network and dividing it by the number of links in the network. For the links connecting an application node to a communication node or vice versa (*external* links), this is equal to h. The average number passing through an *internal* link (a link connecting two nodes in the communication network) is

$$\frac{hN \log N}{2N \log N} = \frac{h}{2}.$$

Thus, the effective transfer rate (in bytes per second) on each external link and internal link is given by

$$\rho(n, m, h) = \frac{mh}{T(n, m, h)},$$

and

$$\rho(n, m, h) = \frac{mh}{2T(n, m, h)},$$

respectively. If we fix $h = 100$, the effective transfer rate on the external links is given by

$$\begin{aligned}
\rho(2, 8, 100) &= 1.82 \times 10^5 \\
\rho(2, 16, 100) &= 2.75 \times 10^5 \\
\rho(2, 32, 100) &= 3.47 \times 10^5 \\
\rho(2, 64, 100) &= 4.15 \times 10^5 \\
\rho(3, 8, 100) &= 1.43 \times 10^5 \\
\rho(3, 16, 100) &= 1.85 \times 10^5 \\
\rho(3, 32, 100) &= 2.98 \times 10^5 \\
\rho(3, 64, 100) &= 3.89 \times 10^5
\end{aligned}$$

The bidirectional data rate of the transputer system used for the experiments is limited to 0.625 megabyte per second. Thus for packets that are 64 bytes long, the transfer rate is approximately 65% of the theoretically possible transfer rate. This shows that the implementation is very efficient with very little overhead.

Besides the fact that the total time spent in intranode routing increases as the dimensionality of the hypercube increases, there are several other reasons preventing the performance of the router to scale with the number of nodes. One reason is that the service time of a DMA controller depends on the network size. The service time is defined as the time interval starting at

the time the DMA controller is ready to transmit until the time transmission actually starts. Obviously, the service time increases as the number of links attached to each node increases. Another reason is contention that arises if several DMA controllers access the same memory over a single bus.

Thus it is clear that for small packets of 8 bytes (typically four address bytes and four data bytes), the intra-node routing delay is of dominating influence on the overall system performance. For 64 byte packets, the effective throughput is dominated by the network capacity.

A possible conclusion is that if a BSP computer (in which the router must deliver a huge amount of *small* packets) is to be built, the communication subsystem must also incorporate high performance communication nodes in order to be sure that the router can realize an h-relation in ah time units for all h greater than some h_0, where a is a small constant independent of the number of processors. A possible solution is obtained by using a multi-level crossbar network to connect the input ports to the output ports. However, a potential bottleneck is the switch controller which determines the switch settings, since it can only establish one physical connection at a time. The major drawback of such a local interconnection is its cost complexity: $O(k^2)$ for a $k \times k$ crossbar. This and other node models have been examined in [2].

3 Implementations of BSP computers

Let us now look at the implications the preceding remarks have for implementations of the BSP computer. Suppose a superstep consists of L instructions (L is called the *periodicity parameter* in [7]), 32% of which are data-transfer instructions. Suppose also that 75% of the data-transfer instructions are `load` instructions (which cause two packets to be send in the communication network) and that the remaining 25% are `store` instructions. These percentages have been taken from reference [4]. Let $\overline{C}$ denote the *average instruction execution time* (the time needed to execute a program divided by its instruction count) and let $T_{comm}(N,h)$ denote the time needed for realizing an h-relation on an N-node hypercube network (the *communication* time). Then the execution time $T(N,L)$ of this superstep can be estimated by

$$\begin{aligned} T(N,L) &= L\overline{C} + T_{comm}(N, 2 \times \#\text{load} + \#\text{store}) \\ &= L\overline{C} + T_{comm}(N, 0.56L). \end{aligned}$$

Now we can either assume a DMA based communication mode as in the transputer architecture, or assume that a crossbar-switch is used to perform the intra-node routing. With a DMA based protocol, the execution time of the superstep is

$$T(N,L) = L\overline{C} + (0.56L + \log(N))t_m + 0.28L\log(N)t_d,$$

where $t_m = m/B$ (the packet transmission time) and where we charge optimal $(h+\log(N))t_m + \frac{1}{2}h\log(N)t_d$ time for realizing an h-relation on an N-node hypercube. Thus if we want to keep the efficiency E at a desired level between 0 and 1, we must have

$$E = \frac{L\overline{C}}{L(\overline{C} + 0.56t_m + 0.28\log(N)t_d) + \log(N)t_m},$$

or,

$$E = (1 + \frac{0.56}{\lambda} + \frac{0.28\log(N)}{\mu} + \frac{log(N)}{\lambda L})^{-1},$$

where $\lambda = \overline{C}/t_m$ and $\mu = \overline{C}/t_d$. Figure 4 shows how large the ratio $\overline{C}/t_m$ must be in order to achieve an efficiency $E = 0.2$, 0.4, 0.6 and $E = 0.8$, where we have taken L to be a constant (in this case 20) and t_d to be equal to t_m. The graphs show that the efficiency can be maintained at a desired value for increasing number of processors only if the ratios $\overline{C}/t_m$ and $\overline{C}/t_d$ are also increased. For example, to achieve an efficiency of $E = 0.8$, $\overline{C}/t_m$ must be larger than 8.84 when $N = 32$, $\overline{C}/t_m \geq 10.16$ when $N = 64$ and when $N = 128$, $\overline{C}/t_m \geq 11.48$. Assuming 8 byte packets and that $\overline{C} = 100$ ns, 928 Mbyte/second channel bandwidth is required to achieve an efficiency of 0.8 on an 128-node hypercube. For $E = 0.2$, $\lambda = \overline{C}/t_m$ is relatively independent of the network size N. However, if the percentage of time spent in intra-node routing (t_d) increases relative to the time needed for a basic routing step (t_m), a larger spread would occur since network growth increases the total time spent in intra-node routing on the order of $\log(N)$.

This brings us to the question of how to balance the machine. How much bandwidth is required to achieve a satisfactory processor utilization? How many nodes

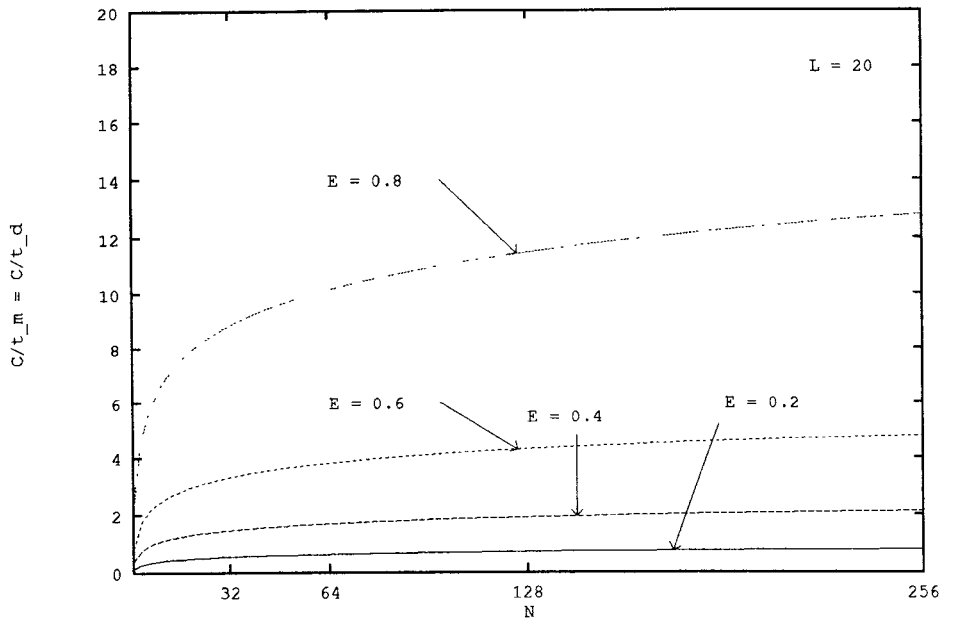

Figure 4: The ratio $\overline{C}/t_m$ versus network size N while maintaining constant efficiency.

are needed to match the targeted performance? And can this be realized in foreseeable technologies? Assuming 20 Mbytes/second channels (as in the recently introduced T9000 transputer manufactured by INMOS) and 8 byte packets (four data bytes and four address bytes), the performance rate of an N-node BSP computer can be written as

$$\frac{NL}{T(N,L)} = \frac{NL}{L\overline{C} + (0.56L + \log(N)) \times 4 \times 10^{-7} + 0.28L\log(N)t_d}.$$

This expression can be simplified if we take $L = \log(N)$ and $t_d = 400$ ns (thus the intra-node delay is equal to t_m). The performance rate is then

$$\text{performance rate} = \frac{N}{\overline{C} + 6.24 \times 10^{-7} + 1.12 \times 10^{-7} \times \log(N)}.$$

Peak performance of, for example, the CRAY-YMP/8 is 2667 Mflops. So, if we want the BSP computer to match the performance of the CRAY-YMP/8, we must have

$$N \geq 1664.2 + 298.7 \times \log(N) + 2667 \times 10^6 \times \overline{C}.$$

It follows that a 13-dimensional hypercube, with 8192 nodes, is needed to match performance of the CRAY-YMP/8. The average instruction execution time may then be as large as 1 μs.

The situation changes if we use a $\log(N) \times \log(N)$ crossbar to connect the input ports to the output ports. In that case a conflict occurs only when two or more input ports request to use the same output port simultaneously. Provided sufficiently large buffer lengths, this does not, however, degrade the performance of the routing scheme, since in the original routing scheme packets that compete for the same link are already serialized. Under these assumptions, the time needed to perform an h-relation on an N-node hypercube becomes $(h + \log(N))t_m$ and we can now write the execution time $T(N, L)$ of a superstep as

$$T(N, L) = L\overline{C} + (0.56L + \log(N))t_m.$$

With this model the efficiency is

$$E = (1 + \frac{0.56}{\lambda} + \frac{\log(N)}{\lambda L})^{-1}, \text{ where } \lambda = \overline{C}/t_m.$$

The graph of figure 5 shows that λ has to grow very slowly if we want to maintain the efficiency at a desired level. Compared to the graph of figure 4, an

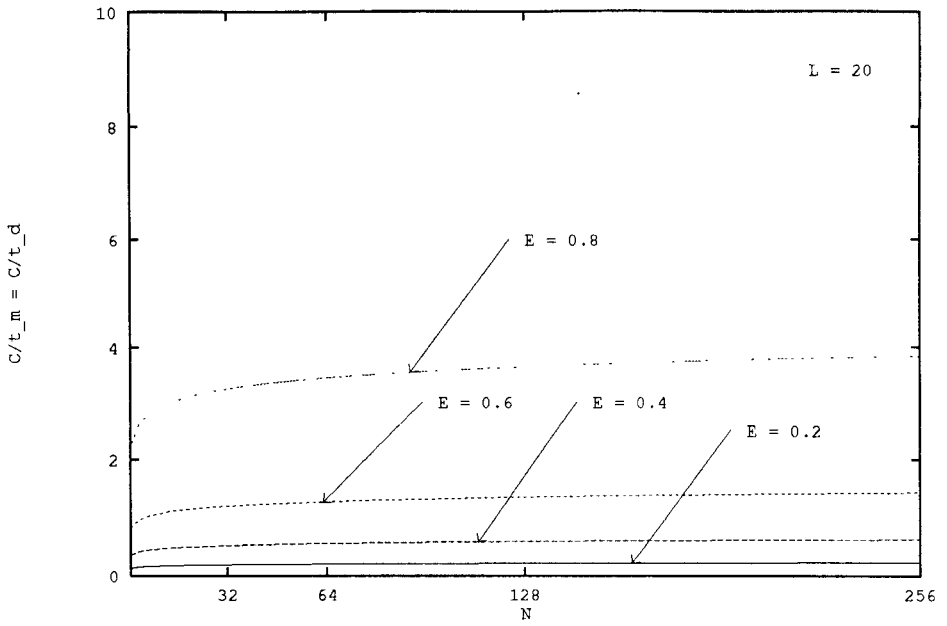

Figure 5: The ratio $\overline{C}/t_m$ versus network size N while maintaining constant efficiency, assuming that a communication node has a protocol processor for each input port.

efficiency 0.8 is achieved at a much earlier point. Assuming 20 Mbytes/second link bandwidth and 8-byte packets, the performance rate of an N-node BSP computer can now be written as

$$\text{performance rate} = \frac{NL}{L\overline{C} + (0.56L + \log(N)) \times 4 \times 10^{-7}},$$

and when $L = \log(N)$, a BSP computer would be faster than the CRAY-YMP/8 if

$$N \geq 2667 \times 10^6 \times (\overline{C} + 6.24 \times 10^{-7}).$$

It follows that when the average instruction execution time is less than 144 ns, 2048 nodes are sufficient to match the performance of the CRAY-YMP/8.

4 Conclusions and future work

In this paper we have studied the BSP model of parallel computation as proposed by Valiant. We have seen that the intra-node routing delay causes the time needed for realizing an h-relation to behave as $O(h \times \log p)$ instead of $O(h + \log p)$ as required. Further experimentation is required to determine the exact relationship between communication node architecture (local interconnection) and network throughput.

Current work focuses on porting the router to other systems in order to study the behavior of the router for larger network sizes. Furthermore, we are constructing a simulator that closely follows the behavior of a real system for carrying out the performance analysis of BSP algorithms.

References

[1] G.J. Brebner. *Parallel Computation on Sparse Networks of Processors.* PhD thesis, Department of Computer Science, Edinburgh University, 1983.

[2] P.W. Dowd and M. Carrato. High speed routing in a parallel processing environment: A simulation study. In *The 24th Annual Simulation Symposium*, pages 60–72, New Orleans, Louisiana, April 1991.

[3] A.R. Karlin and E. Upfal. Parallel hashing: An efficient implementation of shared memory. *Journal*

of the ACM, 35(4):876–892, October 1988.

[4] D.A. Patterson and J.L. Hennessy. *Computer Architecture, A Quantative Approach.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[5] A.G. Ranade. How to emulate shared memory. In *28th Annual IEEE Symp. on Foundations of Computer Science*, pages 185–194, 1987.

[6] L.G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 11, 1982.

[7] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8), 1990.

[8] L.G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science.* North Holland, Amsterdam, 1990.