A Critical Look at Three of Parallel Computing's Maxims

Bruce M. Maggs
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper takes a critical look at the following three maxims.

1. *Parallel architecture is converging on a design based on commodity microprocessor chips.*

2. *Wormhole routing is decidedly more efficient than store-and-forward routing.*

3. *The PRAM is an unrealistically ideal model of computation.*

1 Introduction

The last few years have seen the introduction of a large number of parallel computers, as well as the failures of several manufacturers. As the machines have been released, they have been thoroughly evaluated, and a wide range of benchmark data is now available. Furthermore, in this time our expertise in programming and extracting performance from parallel machines has greatly increased, a theory of parallel and network algorithms has developed, and our understanding of the modeling process has improved. Hence, now seems a propitious time to take a critical look at our maxims.

2 Architectural convergence

> *Parallel architecture is converging on a design based on commodity microprocessor chips.*

The economics of mass production make it difficult for supercomputer manufacturers to design custom processors that outperform commodity microprocessors. Hence, some have concluded that parallel computers must be built out of commodity processors, and indeed most of the current generation of parallel computers use them. Before agreeing with this conclusion, however, it is instructive to review the specifications and benchmark results for the current generation of parallel machines.

2.1 Modeling parallel computers

In order to discuss and compare the latest parallel computers, it is helpful to simplify the description of each machine's network network to a small number of parameters. The most important of these are latency and bandwidth. The *latency* L of a network is the time it takes for a message to traverse the network. In actual networks this will depend on the topology of the network, which particular ports the message is passing between, and the congestion of messages in the network. The latency, however, can often be usefully modeled by considering the worst-case time assuming that the network is not heavily congested. A *bisection* of a machine is a partition of its processors into two equal-sized sets. The *bisection bandwidth* of a network is the minimum, over all bisections, of the (bidirectional) capacity of the network to pass data between the two sets of processors. The bisection bandwidth depends on the topology of the network and the width of the communication channels. In practice, it is not always possible to utilize the full bisection bandwidth of a machine. Hence, the bandwidth can often be usefully modeled as the maximum rate at which processors can inject messages into the network without causing it to become heavily congested, assuming a uniform distribution of message destinations. In this case, the bandwidth can be expressed as the minimum *gap* g between successive injections of messages into the network.

Three models of computation that are based on latency and gap are the Postal model [2], the Bulk-Synchronous Parallel (BSP) model [27], and the LogP model [5]. In the Postal model, a network is described by a single parameter L, its latency, and it is assumed that the bandwidth of the network is sufficient to support as much traffic as the processors can generate. The Bulk-Synchronous Parallel model adds a second parameter g, the minimum ratio of computation steps to communication steps, i.e., the gap. The LogP model includes both of these parameters, and adds a third parameter o, the overhead, or wasted time, incurred by a processor upon sending or receiving a message. In general, the smaller than latency, gap, and overhead, the more powerful the network.

Latency Of the three parameters, large latency appears to be the easiest to cope with. There are a variety of automatic techniques for hiding latency with minimal hardware support provided that the amount of excess parallelism, called *slackness* [27], is sufficiently large. These techniques include assigning a lot of work to each processor, emulating many virtual processors on each physical processors, executing multiple independent threads (with rapid context switching) [25] on each processor, performing vector-accesses (gather and scatter), using split-phase operations, prefetching instructions and data, and performing redundant computation.

Gap Large gap is more difficult to overcome. Although some algorithmic techniques have been developed to overcome the gap, i.e., minimize total commu-

Machine	Processors	GFlops	Bisection	gap
Cray C916	16	16	15.4	1.04
Cray T916	16	30	25	1.2
Cray T3D	2048	307.6	9.6	32.0
IBM SP-2	128	34.0	0.32	106.4
CM-5	1024	131.1	0.45	291
Intel Paragon	1024	230.4	0.7	329

Table 1: Peak GFlops, bisection bandwidth, and gap or various parallel machines. Bisection bandwidth is given in GWords/sec (64-bit words). For each machine, the gap has been estimated by dividing the machine's peak GFlops by its bisection bandwidth. The data in this table was gathered primarily from the manufacturers' world-wide-web sites. The bisection bandwidths for the Cray C916 and T916 are half of their memory system bandwidths.

nication, they tend to be problem and data specific.

Gap is sometimes mistakenly viewed as reflecting the inherent difference in speed between processors and networks. This belief is based in part on a confusion of gap and latency. Although the speed of light provides an inherent lower bound on the latency of any network, the gap, which measures bandwidth, is not limited by the speed of light. What actually limits its bandwidth is the amount of money that a manufacturer is willing to spend on the construction of a communication network, which can be varied, for example, when deciding on the number of network pins per processor, the number of router nodes per processor, the number of engineers allocated to network design, and whether any modifications should be made to the processor chip. As Table 1 shows, the gaps for commerical parallel computers vary tremendously from machine to machine.

Overhead In solving problems that require a lot of communication, high overhead is debilitating. Unfortunately, commodity microprocessors, which were not designed with interprocessor communication in mind, tend to suffer from high overhead. The machines with large overheads also tend to have small network bandwidths. One explanation for the huge difference in network bandwidth between a machine like the Cray C916 and a machine like the Thinking Machines CM-5 (which has many more processors), for example, is that the overhead of sending a message on the CM-5 is so high that it could not make effective use of additional bandwidth.

Overhead, however, should not be viewed as an inherent or unavoidable penalty for interprocessor communication. In particular, the overhead on the parallel machines with custom processors is negligible. In the Cray C916, for example, a processor can inject up to 8 64-bit words into the network in each cycle [20]. In the J-Machine, a processor can compose and inject a four word message into the network in four clock cycles [19]. In iWarp, a processor can access a logical channel in two cycles [3]. These machines demonstrate that overhead can be eliminated by providing on-chip support for communication.

2.2 Benchmark Performance

Five years ago, a typical measure of the power of a massively parallel computer was the peak rate at which it could perform floating-point operations, i.e., how many GigaFlops (billion floating point operations per second) it could perform. Manufacturers produced machines that achieved very high peak floating point performance, but this measure was found to be non-predictive because the machines could rarely achieve it.

More recently, the performance of these machines has been judged against a set of standard benchmarks, which include the LINPACK and NAS [1, 23] benchmarks. The results of the NAS benchmarks for several machines summarized in Table 2. The same data is shown in a different format in Table 3. Here, each machine's performance is given as a fraction of peak floating point performance. Two conclusions that can be drawn from these benchmarks is that machines with custom processors, high communication bandwidth, and no overhead perform well across the board, and that peak floating-point performance is relevant only on embarrassingly parallel problems.

2.3 Networks of workstations

Some argue that not only will parallel architecture converge on commodity microprocessors, but that networks of workstations will eventually displace parallel computers. There are undoubtably many problems to which networks of workstations can be applied. In making the movie *Toy Story*, for example, Pixar reportedly engaged a farm of 100 Sun Sparc 5 workstations. Similarly, in designing the R10000 microprocessor, MIPS employed over 150 workstations, including 10 SGI Challenge Multiprocessors with 8-12 processors each, in full-time simulations. Both of these applications require relatively little communication between workstations, however. Before concluding that networks of workstations can also be applied to more communication intensive tasks, it is worth noting that in addition to large overhead, there is currently an enormous difference in network bandwidth between the local area networks that are used to connect workstations and the network in a machine like the Cray T932. For example, the bisection bandwidth of the Cray T932 is more than 320,000 times larger than that of 10Mbit/sec Ethernet.

The most recent generation of commodity microprocessors include on-chip support for interprocessor communication. In particular, the MIPS R4400, R8000, and R10000, and the Intel Pentium Pro all provide support for cache coherence and cache consistency when used in a system in which multiple processors and memory modules share a bus. Hence, it could be said that commodity microprocessor architecture is converging on a design in which interprocessor communication is supported on chip.

2.4 Summary

In the current generation of parallel machines, the machines with the best across-the-board performance

Machine	Processors	EP	MG	CG	FT	IS	LU	SP	BT
Cray C916	16	15.66	9.85	11.58	14.46	13.18	11.85	13.21	12.99
Cray T916	8	15.17	8.32	10.75	11.67	9.16	10.42	11.38	11.07
Cray T3D	1024	66.85	29.38	7.41	29.41	10.59	21.64	27.33	50.04
IBM SP-2	128	29.34	13.73	4.83	7.62	6.53	9.4	12.58	15.27
CM-500	256	20.92	8.67	1.98	8.38	3.0	3.02	4.93	8.98
Intel Paragon	512	36.79	5.03	3.03	8.92	2.75	2.57		

Table 2: The performance of various parallel machines on the NAS Parallel Benchmarks, Class B, as of December 1995. The abbreviations are EP (Embarrassingly Parallel), MG (Multigrid), CG (Conjugate Gradient), FT (3-D FFT PDE), IS (Integer Sort), LU (LU Simulated CFD Appl.), SP (SP Simulated CFD Appl.), and BT (BT Simulated CFD Appl.). The benchmarks are listed in order of increasing communication requirements. Times are given in multiples of the time for a one-processor Cray C90.

Machine	Processors	EP	MG	CG	FT	LU	SP	BT
Cray C916	16	0.67	0.34	0.32	0.58	0.53	0.54	0.57
Cray T916	8	0.70	0.31	0.32	0.50	0.49	0.49	0.52
Cray T3D	1024	0.15	0.05	0.01	0.06	0.05	0.06	0.11
IBM SP-2	128	0.59	0.22	0.06	0.14	0.20	0.16	0.32
CM-500	256	0.44	0.15	0.03	0.16	0.07	0.10	0.19
Intel Paragon	512	0.43	0.02	0.01	0.05	0.02		

Table 3: The fraction of peak performance achieved by each machine on each benchmark (excluding Integer Sort). The numbers were calculated by dividing achieved GFlops/sec by peak GFlops/sec.

use custom processors. By providing on-chip support for communication, and thus eliminating overhead, these machines are able to utilize much higher network bandwidths. As future commodity microprocessors begin to provide more on-chip support for communication, the parallel machines using them can be expected to become more competitive.

3 Wormhole routing

> *Wormhole routing is decidedly more efficient than store-and-forward routing.*

Many papers begin with a similar remark, and there is no denying that wormhole routing has become the routing method of choice in the latest generation of massively parallel computers, appearing in experimental machines such as iWarp [3] and the J-Machine [19], and commercial machines such as the Intel Paragon, Cray T3D [12], and Connection Machine CM-5 [17]. Nevertheless, in light of recent analytical discoveries, and based on the benchmark performance of the current generation of parallel machines, the validity of this maxim deserves a hard look.

In a wormhole router, the bits in a message are grouped into a sequence of *flits*, where a flit is the smallest unit of information that can be queued at a node of the network. The message is viewed as a worm that traverses the network in a bit-serial fashion. The header flit or flits, which specify the route that the message will take, proceed first. As the header flits pass through a switch, they set up a connection between the incoming channel on which they arrived, and an outgoing channel, on which they leave. The remainder of the flits contain no routing information. Instead, it is the responsibility of the switch to forward them from the incoming channel to the outgoing channel. As the header flits work their way through the network, the flits following behind become spread out across the network.

In order to keep the size of the switches small in a wormhole router, the capacity of each switch to queue flits is limited to one per incoming channel. Because a switch cannot identify which message a flit belongs to from its contents, the flits of a message must arrive on the incoming channel in a contiguous fashion. Since a switch can only queue one flit of a message, if the header flit of a message cannot advance then the flits following the header must stall.

Wormhole routing has several advantages over the other common method of routing, *store-and-forward* routing. In a store-and-forward router, a switch must store an entire message before it can forward any part of that message along the next edge on its route. Hence, a message can be viewed as making discrete hops, called *message steps* from switch to switch as it traverses its path. If a b-bit message traverses a path of length d, and is never delayed, then it will reach its destination in bd bit steps (assuming each channel can transmit one bit in each bit step). In a wormhole router, however, the header flit does not wait for the rest of the message. The first bit of the header flit arrives at its destination after d steps, and the last bit of the message arrives after $d + b - 1$ steps. The difference in time is due to the better utilization of network edges by the wormhole router; in a wormhole router, multiple edges of the network can simultaneously transmit flits belonging to a single message, whereas in the store-and-forward algorithm only one edge can transmit a flit of the messsage at any time. In addition to reduced latency, wormhole routing also has the advantage that it can be implemented with small, fast switches.

Wormhole routing owes much of its recent popularity to an influential paper by Dally and Seitz [9], which introduced the method. Much of the paper by Dally and Seitz is devoted to the design of wormhole routing algorithms that avoid *deadlock*. A wormhole routing algorithm can deadlock if the header flit of every worm is prevented from moving because the queue that it wants to enter is full. For example, a pair of worms can deadlock if the head of the first worm wants to enter a queue occupied by the tail of the second worm, and vice versa. The solution to this problem in the paper by Dally and Seitz is to allow each physical channel to emulate several virtual channels, and to construct a virtual network in which the worms cannot form cycles. The virtual channels share the physical wire (or wires) provided by the physical channel, but the switch maintains a separate queue for each virtual channel. This solution has also been implemented in hardware. For example, each physical channel of iWarp supports 4 virtual channels, and each physical channel of the J-Machine supports 2 virtual channels [19]. The paper by Dally and Seitz was followed by a large number of papers describing different forms of deadlock free routing for various networks, e.g., [7, 6, 18, 21].

3.1 Rigorous analyses of wormhole routing

Recently there have been a number of breakthroughs in analyzing the time required by wormhole routing algorithms. The results are most easily stated for the case of static oblivious routing. In an *oblivious* routing algorithm, each worm is routed from its source to its destination along a predetermined path, independent of the actions of other worms in the network. In a *static* routing problem, all of the worms to be routed through the network are present at the onset, and no new worms enter the network until all of these worms have reached their destinations. Of course, in real networks worms are not usually routed in batches (i.e., the routing is dynamic rather than static), and the path that a worm takes may depend on the other worms that it encounters en route to its destination (i.e., the routing is adaptive rather than oblivious). Nevertheless, the results derived for static oblivious routing can often been translated to results for dynamic or adaptive routing.

Network independent analyses Perhaps surprisingly, it is possible to design wormhole routing algorithms and to prove lower bounds that can be applied to any network. The performance of a wormhole routing algorithm is described in terms of the following four parameters. The dilation d is the maximum number of network channels that must be traversed by any one worm. The congestion c is the maximum number of worms that must traverse any one channel. The channel capacity q is the number of virtual channels that each physical channel can support, i.e, the number of flits that can be queued for each physical channel. Throughout this section we assume that a physical channel is cable of transmitting only one bit at a time, and must service its virtual channels in a round-robin fashion. Finally, the bit-length, b, is the number of bits in each worm.

The idea of decoupling the path selection process from the scheduling process, and then analyzing the scheduling process alone, was first used by Leighton, Maggs, and Rao [16] in the context of store-and-forward routing. They proved that for any set of messages whose paths are edge-simple (i.e., no path uses the same network edge more than once) and have congestion c and dilation d, there is a store-and-forward schedule that routes the messages in $O(c+d)$ message steps and never stores more than a constant number of messages in the queue at the head of an edge. The $O(c + d)$ bound improves on the naive $O(cd)$ bound, and is optimal. Translated to bit steps, the bound is $O(b(c + d))$, which is optimal for $c > d$ (because some edge must transmit at least bc bits). Note that there is no restriction on the structure of the network, and that the size of the network and number of messages do not appear in the running time. The proof used the Lovász Local Lemma and was nonconstructive.

For the special case of leveled networks, Leighton, Maggs, Ranade, and Rao [15] presented a simple randomized on-line store-and-forward algorithm. In a *leveled* network with depth L, each node is labeled with an integer between 0 and L (its level), and each edge with its tail on level i, $0 \leq i < L$, has its head on level $i + 1$. The algorithm in [15] routes any set of N messages in a leveled network with depth L in $O(c+L+\log N)$ message steps. Translated to bit steps, the time is $O(bc + bL + b \log N)$, which is optimal for $c \geq L + \log N$.

Ranade, Schleimer, and Wilkerson [22] were the first to state non-trivial wormhole routing results in terms of these four parameters. They showed that on any network, any set of b-bit messages whose paths have congestion c and dilation d can be routed in $O(bcd)$ bit steps. The $O(bcd)$ bound improves on the naive $O((b + d)cd)$ bound. They did not consider the case $q > 1$.

Ranade et al. also showed how to construct a network with $q = 1$, and a set of b-bit messages whose paths have congestion c and dilation d in which the optimal wormhole routing schedule requires $\Omega(bcd)$ time. Note that the time for the optimal store-and-forward algorithm is at most $O(b(c + d))$, which can be much smaller than bcd when c and d are roughly equal.

Cypher et al. [8] showed that there is a wormhole routing algorithm that takes time $O(bcd^{1/q} + (d+b) \log N)$ time, for any $q \leq \log N$, where N is the number of nodes in the network. Cole, Maggs,and Sitaraman [4] also independently showed that it is possible to route any set of b-bit messages whose paths are edge simple and have congestion c and dilation d in

$(b+d)c(d\log d)^{1/q}2^{O(\log^*(c/d))/q}$ bit-steps. (They did not provide an on-line algorithm, however.) What is interesting about these time bounds is that they exhibit a non-linear dependence on q, the number of virtual channels that each physical channel can support. In particular, an increase in q results in superlinear speedup. This effect was first observed by Koch [11] in the context of circuit-switching on the butterfly.

In addition to these upper bounds, Cole, Maggs, and Sitaraman also proved a nearly matching lower bound. For any values of c, d, q, and b, where $c, d \geq q+1$ and $b = (1+\Omega(1))d$, they showed how to construct a network and a set of paths with congestion c and dilation d and a set of b-bit messages that require $\Omega(bcd^{1/q})$ bit-steps to route. The lower bound implies that in order for a wormhole routing algorithm to outperform the optimal store-and-forward algorithm, q must be at least $\Omega(\log d)$.

Wormhole routing on butterfly networks Wormhole routing on butterfly networks has been studied extensively, and in general wormhole routing algorithms on these networks are not as fast as store-and-forward algorithms. As in the case of general networks, however, the performance of wormhole routing algorithms on butterfly networks has a non-linear dependence on q, both in the case of static routing [4, 8, 11] and dynamic routing [24]. By making q large enough, the algorithms can be made competitive with store-and-forward algorithms.

Early work focused on the case $q = 1$. Kruskal and Snir [13] showed that if each input in an N-input butterfly node sends a message to a randomly-chosen output, then the expected number of worms that reach their destinations without ever being delayed is $\Theta(N/\log N)$. In Problem 3.285 of [14], Leighton describes an algorithm for solving a random routing problem in which each input has one b-bit worm to send. The algorithm runs in $O((b+\log N)\log N)$ bit steps. In Problem 3.286, he shows that the algorithm can be converted to one that routes any permutation using Valiant's idea [26] of first routing to random intermediate destinations. For the interesting case of $b = O(\log N)$, the time is $O(\log^2 N)$ bit steps. The algorithm can easily be generalized to the case in which input has k worms to send, and runs in $O(k(b+\log N)\log N)$ bit steps. For the interesting case of $b = O(\log N)$ and $k = \log N$, the time is $O(\log^3 N)$ bit steps. Felperin et al. [10] independently discovered an $O(\log^4 N)$ bit-step algorithm for solving a random problem for the case $b = O(\log N)$ and $k = \log N$, and then Ranade et al. [22] discovered an $O(\log^3 N \log\log N)$ bit-step algorithm.

Ranade et al. also proved a lower bound of $\Omega(\log^3 N/(\log\log N)^2)$ on the number of bit steps required for a wormhole routing algorithm to route a $\log N$-relation when $b = \log N$ and $q = 1$. This lower bound is nearly tight. Note that several known store-and-forward routing algorithms run in $O(\log^2 N)$ bit steps (see Leighton[14]). Hence, the store-and-forward algorithms are faster. The switches executing the store-and-forward algorithms, however, must each have the capacity to queue an entire message, i.e., $O(\log N)$ bits.

For $q > 1$, a non-linear dependence on q has been observed. The first result is due to Koch, who showed that if each input in an N-input butterfly network sends a worm to a randomly-chosen output, then the expected number of worms that reach their destinations without ever being delayed is $\Theta(N/(\log N)^{1/q})$. Next, Cypher et al. [8] found an algorithm for routing k worms from each input of an N-input butterfly to randomly-chosen outputs in $O(kb\log^{1/q} N + \log(kN)\log N)$ time. They also proved a lower bound of $\Omega(kb\log^{1/q} N(\log\log N)^{-2/q})$ bit steps. Cole, Maggs, and Sitarmanan [4] independently discovered a randomized algorithm that routes a k-relation on the inputs and outputs of an N-input butterfly in $O(b(k+\log N)(\log^{1/q} N)\log\log Nk)$ bit-steps. Cole et al. also proved an $\Omega(bk(\min\{b, \log N\})^{1/q}/\log\log N)$ lower bound that holds for a broad class of algorithms. Scheideler and Vöcking [24] proved that the non-linear dependence on q holds for the case of dynamic wormhole routing on butterflies as well.

For $b = \log N$, $k = \log N$, and $q = \log\log N$, the time for the wormhole routing algorithms drop to $O(\log^2 N)$. Hence, a wormhole routing algorithm can achieve the same time bound as the store-and-forward algorithm, while queuing at most $O(\log\log N)$ bits at the end of each edge.

3.2 Empirical comparisons

Due to the relatively small number of parallel machines and wormhole routers that have been built, it is difficult to draw general conclusions about their performance relative to that of store-and-forward routers from the benchmarks. It is worth noting, however, that the most powerful routing network – by far – is a store-and-forward router. This network belongs to the Cray T932, and has a network bisection bandwidth 50 GWords/sec. The closest competitor among wormhole routers is that of the Cray T3E, which has a bisection bandwidth of 15.25 GWords/sec. Furthermore, the machines with the best across-the-board benchmark performance, the Cray C916 and Cray T916, both have store-and-forward routers.

3.3 Summary

The analytical results in this section demonstrate that there are situations in which store-and-forward routing algorithms outperform wormhole routing algorithms. The performance of wormhole routing networks, however, can be improved by adding support for virtual channels. In practice, the most powerful routing networks are store-and-forward networks, and the machines that use these networks also have the best across-the-board performance on the benchmarks.

4 The PRAM model

The PRAM is an unrealistically ideal model of computation.

The model most frequently used by the designers of parallel algorithms is the Parallel Random-Access

Machine (PRAM) model. In this model, a set of processors share a global memory system, and every basic CPU operation, including arithmetic and logical operations and accesses to the global memory system, require a single time step.

The PRAM model has been heavily criticized as being unrealistically ideal, primarily because no shared memory system can perform a memory access as fast as a processor can execute a local arithmetic or logical operation.

In order to understand the usefulness of the PRAM model, it helps to examine the issue of work-efficiency. The efficiency of an algorithm is determined by the total number of operations, or *work* that it performs. On a sequential machine, an algorithm's work is the same as its time. On a parallel machine, the work is simply the processor-time product. Hence, an algorithm that takes time t on a P-processor machine performs work $W = Pt$. In either case, the work roughly captures the actual cost to perform the computation, assuming that the cost of a parallel machine is proportional to the number of processors in the machine. We call an algorithm *work-efficient* (or just efficient) if it performs the same amount of work, to within a constant factor, as the fastest known sequential algorithm. For example, a parallel algorithm that sorts n keys in $O(\sqrt{n}\log n)$ time using $\sqrt{n}$ processors is efficient since the work, $O(n \log n)$, is as good as any (comparison-based) sequential algorithm. However, a sorting algorithm that runs in $O(\log n)$ time using n^2 processors is not efficient. The first algorithm is better than the second – even though it is slower – because its work, or cost, is smaller. Of course, given two parallel algorithms that perform the same amount of work, the faster one is generally better.

The notion of work-efficiency leads to the following observation:

> A model can be useful without mimicking any real or even realizable machine.

Instead, it suffices that any algorithm that runs efficiently in the model can be translated into an algorithm that runs efficiently on real machines. For example, any algorithm that runs efficiently on a P-processor PRAM can be translated into an algorithm that runs efficiently on a (P/L)-processor machine with latency L (provided that there is no gap or overhead). In the translated algorithm, each of the P/L processors emulates L PRAM processors. The latency is "hidden" because a processor has useful work to perform while waiting for a memory access to complete. Although the translated algorithm is a factor of L slower than the PRAM algorithm, it uses a factor of L fewer processors, and hence is equally efficient.

The notion of a work-efficient emulation is not just theoretical. Indeed, the key to extracting the maximum performance out of a parallel vector machine such as the Cray C916 is to view it as a data-parallel PRAM. In the Cray C916, each processor has a set of 8 128-word vector registers. Although the network has considerable latency, the latency can be hidden by performing operations on entire registers rather than on single words. In particular, the C916 provides vector memory operations called gather and scatter. In a scatter operation, for example, each of the 128 words in a vector register is written to a (potentially) different word in memory. The network bandwidth of the C916 is so large that these write operations can be pipelined at the rate of one write per cycle, i.e., there is no overhead, and no gap. Hence, while waiting for the first write to complete, a processor can perform useful work (and hence hide the latency) by issuing the remaining writes. Of course, in order to make use of this capability, the program must have a set of 128 writes to perform. Thus, although the machine has at most 16 processors, the key to making it run efficiently is to view it as a machine with $16 \cdot 128 = 2048$ virtual processors, each executing the same data parallel program, i.e., as a data-parallel PRAM.

4.1 Summary

The purpose of an abstract model of computation such as the PRAM is to provide a tool for developing efficient algorithms. As such, it is not necessary for a model to mimic any real machine, provided that algorithms that are efficient in the model can be translated to algorithms that are efficient on real machines. For machines with high bandwidths and no overhead, the PRAM model fulfills this role.

Acknowledgements

I would like to thank Guy Blelloch, Gary Miller, and Uzi Vishkin for many helpful discussions.

References

[1] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS parallel benchmarks. Technical Report RNR-94-007, Numerical Aerospace Simulation Facility, NASA Ames Research Center, Moffett Field, CA, March 1994.

[2] A. Bar-Noy and S. Kipnis. Designing broadcasting algorithms in the postal model for message-passing systems. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 13–22, June 1992.

[3] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P.S. Tseng, J. Sutton, J. Urbanski, and J Webb. iWarp, an integrated solution to high-speed parallel computing. In *Proceedings of the 1988 International Conference on Supercomputing*, pages 330–339, November 1988.

[4] R. J. Cole, B. M. Maggs, and R. K. Sitaraman. On the benefit of supporting virtual channels in wormhole routers. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1996. To appear.

[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN on Principles and Practice of Parallel Programming*, pages 1–12, May 1993.

[6] R. Cypher. Minimal, deadlock-free routing in hypercubic and arbitrary networks. In *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, pages 122–129, October 1995.

[7] R. Cypher and L. Gravano. Storage-efficient, deadlock-free, adaptive packet routing algorithms for torus networks. *IEEE Transactions on Computers*, 43(12):1376–1385, 1994.

[8] R. Cypher, F. Meyer auf der Heide, C. Scheideler, and B. Vöcking. Universal algorithms for store-and-forward and wormhole routing. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, May 1996. To appear.

[9] W. Dally and C. Seitz. Deadlock free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C–36(5):547–553, May 1987.

[10] S. Felperin, P. Raghavan, and E. Upfal. A theory of wormhole routing in parallel computers. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 563–572, October 1992.

[11] R. R. Koch. Increasing the size of a network by a constant factor can increase performance by more than a constant factor. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 221–230. IEEE Computer Society Press, October 1988.

[12] R. K. Koeninger, M. Furtney, and M. Walker. A shared MPP from Cray research. *Digital Technical Journal*, 6(2):8–21, Spring 1994.

[13] C. P. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C–32(12):1091–1098, December 1983.

[14] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.

[15] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, July 1994.

[16] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–180, 1994.

[17] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Jill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak. The network architecture of the connection machine CM-5. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 272–285, June 1992.

[18] P. M. Merlin and P. J. Schweitzer. Deadlock avoidance in store-and-forward networks. 1: Store-and-forward deadlock. *IEEE Transactions on Communications*, 28:345–354, 1980.

[19] M. D. Noakes, D. A. Wallach, and W. J. Dally. The J-Machine multicomputer: an architectural evaluation. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 224–235, May 1993.

[20] W. Oed. Cray Y-MP C90: system features and early benchmark results. *Parallel Computing*, 18(8):947–954, August 1992.

[21] G. D. Pifarré, L. Gravano, S. A. Felperin, and J. L. C. Sanz. Fully-adaptive minimal deadlock-free packet routing in hypercubes, meshes, and other networks. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 278–290, July 1991.

[22] A. Ranade, S. Schleimer, and D. S. Wilkerson. Nearly tight bounds for wormhole routing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[23] S. Saini and D. H. Bailey. NAS parallel benchmark results 12-95. Technical Report Report NAS-95-021, Numerical Aerospace Simulation Facility, NASA Ames Research Center, Moffett Field, CA, December 1995.

[24] C. Scheideler and B. Vöcking. Universal continuous routing strategies. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1996. To appear.

[25] B. J. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Real-Time Signal Processing IV*, pages 241–248, August 1981.

[26] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, May 1982.

[27] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.