

Towards Scalable Multicomputer Communication Through Offline Routing*

Martin C. Herbordt

Department of Electrical and Computer Engineering
Boston University, Boston, MA 02215
Contact Author EMail: herbordt@bu.edu

Paul Swarztauber

Computational Science Section, Scientific Computing Division
National Center for Atmospheric Research, Boulder, Colorado 80305

*This work was supported in part by the National Science Foundation through CAREER award #9702483, by a grant from the National Center for Atmospheric Research funded through the National Science Foundation, and by a grant from the Compaq Computer Corporation.

Abstract: Even with MPP architectures moving towards larger nodes with more powerful processors, communication delay remains a critical factor in application performance. We examine the use of scheduled or off-line routing to provide large-scale parallel computers with efficient communication. Off-line algorithms enable nearly 100% wire utilization, priority ordering of packets, and extremely efficient switching. Moreover, optimal schedules are known for many of the most time-consuming communication tasks, the ones that dominate a number of grand challenge codes. Communication tasks executing under these optimal schedules have excellent complexity *independent of the number of processors*. They also often have the property that elements arrive at their destinations at regular intervals, typically averaging one element per cycle per node. This allows computation to be *chained* with communication with the result that communication contributes only trivially to computation time. This is in stark contrast to the actual timings that are obtained on current multicomputers where communication dominates. We show here that these results diminish by only small factors even after accounting for realistic wire delay, pin bandwidth, pin-out, switching latency, and network dimensionality. We also present a preliminary comparison with an on-line network showing that the performance improvement—not including chaining—is a factor of between 4 and 13 for small networks, depending on implementation and configuration.

1 Introduction: Objectives and Significance

Many critical computational applications such as weather and climate modeling require substantial global communication. It is often the case however, that when these codes are run on current MPPs, they achieve efficiencies in the low single-digit percentages as communication increases to 90% or more of the total execution time. The prime culprit is network contention, and a vast amount of work has been done to address it; broad areas include switch design, routing algorithms, and latency hiding. Although much progress has been made, the problem is still so serious that users will often run codes which yield far less precise results, but which require only local rather than global communications. In other cases, they scale down the computation to avoid using more than a single multiprocessor node. (For a very recent summary of the state of computing in meteorology see [20].) Current technological trends indicate that this problem will only get worse: processor speeds are currently several orders of magnitude faster than communication speeds and the gap is widening.

We believe that the method that has the best chance of addressing the problem of contention in global communication is the set of techniques that have variously been called offline routing and schedule communication. In particular, optimal schedules exist for many if not most of the global communication patterns in question. These schedules enable nearly 100% wire utilization,

priority ordering of packets, and extremely efficient switching. The potential performance benefits of this approach are tremendous. Not only could there be a drastic reduction in duration in the “communication phases,” but communication latency could be hidden to such an extent that even problematic communication tasks could look like just some other vector operations.

We describe a sample result from the theory of scheduled communication. Let an $N \times N$ matrix (where $N = P$) be mapped to P processors connected by a hypercube. Then this matrix can be transposed in $N/2$ cycles independent of P . The most remarkable fact about this and other optimal schedules, however, is that not only do elements arrive at their destinations at a rate of one per cycle, but that the variance of this rate is often small. As a consequence, applications can be created (or recoded) that pipeline, or chain, computation and communication. This makes latency due to communication negligible. We call the property of regular arrivals of one element per cycle **$O(1)$ communication**. We refer to computers that implement this mechanism as **Vector Multiprocessors**.

Note that $O(1)$ communication does not imply that a vector multiprocessor can access random remote elements in a single cycle; e.g. $\log_2 P$ hops may be required on a hypercube. However, for a very broad class of communication tasks, hypercube multi-port algorithms can deliver $\log_2 P$ elements every $\log_2 P$ cycles, effectively delivering one element per cycle, after perhaps a small number of start-up cycles. The usefulness of the proposed method is precisely in that this serves the needs of the most time-consuming applications: they communicate not through random remote access, but rather through chainable collective communication.

What we show in this article is that the vector multiprocessor is much more than a theoretical construct: it is the realistic basis for a scalable communication network. In later sections we show that the seemingly unrealistic requirements of the theoretical model for scheduled communication have practical instantiations that are not only plausible but comparatively straightforward. In particular, the theoretical results of the kind just described diminish by only small factors even after accounting for wire delay, pin bandwidth, pin-out, switching latency, and network dimension. We also show that the switch can be built (i) with sufficient operating frequency to keep the internals off the critical path, (ii) with a latency of only a few cycles, and (iii) that any amount of on-chip buffering does not affect this timing. In addition, we compare off-line with on-line switching (in a limited domain) and find that off-line routing improves communication performance by factors of between 6.5 and 13 depending on network configuration.

The overall significance of this research is to improve performance of grand challenge applications, particularly weather and climate modeling, by endowing large-scale parallel computers with $O(1)$ communication. Achieving this goal will require continued work in several areas, including algorithms, switch and interface design, system software, and applications. However, the availability

of technologies such as FPGAs with high-speed serial channels [39], standardized interface ports and transmission mechanisms (e.g. Infiniband [16]), and high-quality design software, can provide the basis for a proof-of-concept prototype switch. We sketch some of these proposals later in the paper.

In the next section we provide further motivation by describing the overall problem and outlining the general requirements of a solution. There follows a review of optimal communication algorithms and their underlying assumptions. In Section 4 we begin extending these ideas to more realistic models. We sketch the switch design in Section 5. In Section 6 we discuss solutions to many of the remaining practical issues. In Section 7 we present some preliminary performance results.

2 Motivation: A Sample Application and its Requirements

We motivate the use of scheduled communication with an example from climate modeling. This computation is representative of a wide class of physical simulations.

The preferred method of computational climatology involves three-dimensional time-dependent general circulation models (GCMs). The quality of the simulation depends largely on the maximum practical spatial and temporal resolution of the GCM, which in turn is determined by the available computational resources. Although much useful climate science has been produced to date, it is generally agreed that many orders of magnitude more computing could be usefully applied. Even today, however, GCM performance is seriously limited by communication constraints (as described previously).

GCMs have well understood computing characteristics, several of which can be used to specify an effective, scalable, communication mechanism. One characteristic is that communication is predictable, consisting largely of global communication with fixed patterns (including all-to-all) and occurring during spectral transformations. Another is that the computations run in phases with global communication alternating with “Z-axis” code with little overlap possible. A third is that the communication mechanism must scale to at least several thousand compute nodes [13]. The fact that the codes are generally highly tuned means that these characteristics are not likely to change through further improvements in communication/computation overlap or other similar coding optimizations.

The constraints just outlined both motivate us and enable us to address the large-system communication problem with mechanisms based on scheduled routing. The requirement for virtually unlimited computation points to a large P and therefore to methods that scale independent of P . The fact that the codes are already well-tuned and run in phases means that the applications will actually benefit from the increased communication performance; i.e. the benefit will not get

lost in either other inefficiencies or through latency hiding. And the predictable static (or at least semi-static) communication patterns means that the use of preloaded schedules is viable.

Although our expertise in computational applications does not extend far beyond climate/weather modeling, the computing characteristics just outlined do not appear to be unique to CGMs, being shared (at least) by some computational chemistry codes. They are certainly shared by the many applications in which the FFT or similar transformation is the bottleneck, and by communication bound applications involving matrix transposes, all-to-all communication, or communication of other types outlined below.

3 Theoretical Models, Results, and Their Implications

In this section we describe a number of results that demonstrate the basis for fully scalable communication and present the theoretical machine model on which they are based. In the following sections we show that this machine model is plausible and how it can be instantiated.

3.1 Overview of Theoretical Communication Results

The primary theoretical result is that, assuming a problem size appropriate for the number of nodes in the machine, **communication is algorithmically negligible**. In particular, for a large class of critical communication patterns, $N \times N$ data sets can be routed in precisely N time steps. Moreover, **an average of 1 element arrives at each destination every cycle**. Sometimes the result is even better. For example, given a P node processor and an $N \times N$ matrix where $N > P$, the matrix can be transposed such that, on average and with little variance, 2 transposed elements arrive at each of the P destinations every cycle.

What we mean by algorithmically negligible is that, if such a communication is integrated into a computation, a vector chain of compute-communicate-compute can be formed. In this chain, elements are passed from computation to communication and back to computation units with single cycle latency. **Note that this result is independent of the number of nodes P .**

This result is a theoretical basis for the thesis that we can solve larger problems with larger parallel computers with *no slowdown due to the amount of parallelism P , only due to the size of the problem N* . Note that there are no hidden constants other than the time required for the element transfer operations themselves. This and other details are described next.

3.2 Theoretical Communication Result Details

The complexities of several common communication tasks are given in Table 1 for a hypercube network and two communication models: *element-based* and *packet-based* [5, 12, 18, 19, 29, 34]. It is assumed that the problem size increases with the number of processors, which is consistent with

the evolution of computations. This is referred to as the scaled speedup model. The theoretical switch requirements are described in the next subsection.

Element-based communication model: Each node has the capability of simultaneously transferring one element per cycle to each of its $\log_2 P$ neighbors with a cycle time $= \tau$.

Packet-based communication model: Each node has the capability of packaging into packets all of the elements destined for each of its $\log_2 P$ neighbors. Communication time is given as a function L and U^{-1} , the inter-node latency and bandwidth, respectively.

task name	description	communication model	
		element-based	packet-based
<i>Transpose</i>	$x(i, j) = y(j, i)$	$\tau N/2$	$UN/2 + L \log_2 N$
<i>Broadcast</i>		$\tau \lceil N/\log_2 N \rceil$	$U \lceil N/\log_2 N \rceil + L \log_2 N$
<i>ColumnPermute</i>	$x(i, j) = y(i, p(j))$	τN	$UN + 2L \log_2 N$
<i>ReverseArray</i>	$x(i) = y(N - 1 - i)$	τN	$UN + L \log_2 N$
<i>ParallelGather</i>	$x(i) = y(2^n i + 2^m)$	τN	$UN + L \log_2 N$
<i>IndexSyntax</i>	$x(i) = y(ai + b)$	$2\tau N?$	$2UN + 2L \log_2 N?$

Table 1: Hypercube communication requirements for common tasks.

Communication Task Definitions:

1. **Transpose.** An $N \times N$ array is distributed column(row)-wise in $P = N$ processors.
2. **Broadcast.** Each processor has a single element that must be sent to all other processors. This communication task is required by a distributed matrix-matrix or matrix-vector multiply.
3. **Column permute.** Each processor contains a column of an $N \times N$ array. The task is to permute and redistribute the columns.
4. **Reverse array.** An array x_i with N elements is distributed N/P elements per processor. The task is to reverse and redistribute the array.
5. **Parallel gather.** The array x_i with $N = 2^{m+n}$ elements is uniformly distributed across $P = 2^l$ processors. The task is to gather every 2^n th element, 2^m th times and uniformly redistribute across the P processors.
6. **Index syntax.** A related task is to redistribute an array that is mapped by index syntax $x(i) = y(ai + b)$. This task could be useful for the automatic parallelization of any program originally written for a vector uniprocessor. The result in Table 1 is known to be true in practice but unproven in general, which is indicated by the question marks.
7. **General permutation.** Although not included in Table 1, the general one-to-one communication task is discussed in [35].

3.3 Communication and Application Examples

We illustrate one of the results for the element-based model. Consider the task of transposing an 8×8 array $X(i, j)$ that is distributed columnwise on an 8 processor hypercube. The de Bruijn algorithm [34] is given in Table 2 for processor $p = 5$ (where 5 is arbitrary). Columns are in de Bruijn order as prescribed by the transpose algorithm. The contents of the node following each communication cycle are in columns 1 through 4. Each column differs from the one on its left by three elements that are received on its three ports.

T_0	T_1	T_2	T_3	T_4
$x_{4,5}$	$x_{5,4}$	$x_{5,4}$	$x_{5,4}$	$x_{5,4}$
$x_{6,5}$	$x_{4,7}$	$x_{5,6}$	$x_{5,6}$	$x_{5,6}$
$x_{2,5}$	$x_{6,1}$	$x_{4,3}$	$x_{5,2}$	$x_{5,2}$
$x_{3,5}$	$x_{3,5}$	$x_{7,1}$	$x_{5,3}$	$x_{5,3}$
$x_{0,5}$	$x_{0,5}$	$x_{0,5}$	$x_{4,1}$	$x_{5,0}$
$x_{7,5}$	$x_{7,5}$	$x_{7,5}$	$x_{7,5}$	$x_{5,7}$
$x_{1,5}$	$x_{1,5}$	$x_{1,5}$	$x_{1,5}$	$x_{5,1}$
$x_{5,5}$	$x_{5,5}$	$x_{5,5}$	$x_{5,5}$	$x_{5,5}$

Table 2: Contents of processor $p = 5$ during the 4 cycles of an 8×8 transposition.

To further dramatize the significance of these communication algorithms, we provide an example of their use in computing the Fast Fourier Transform (FFT). Let a sequence of $N = P^2$ elements be distributed across a P processor hypercube. Let the communication system be element-based with τ the elemental transfer time between registers in neighboring modules and η the time required for a floating point operation. The unordered FFT of a sequence distributed in this manner requires a single transposition [36]. The complexity of the FFT is $T_h = 5\eta\sqrt{N} \log N + \frac{\tau}{2}\sqrt{N}$ where the first term on the right is the computational time and the second term is the communication time. Therefore as $N = P^2$ increases, communication is asymptotically negligible compared to computation. This is in stark contrast to the actual timings that are obtained on current multicomputers where communication dominates. Note that this result assumes no overlap between communication and computation.

3.4 Theoretical Hardware Requirements

The theoretical hardware model upon which the preceding complexity results and example are realized is now described. The P nodes consist of a processor with memory and are connected with an n -dimensional hypercube. For element-based networks, each node also contains the following.

1. A programmable communication unit that orchestrates communication via $n = \log_2 P$ bidirectional channels. Data elements do not require headers as their routes and schedules have been predetermined.

2. A register, called the ξ -register, that interfaces to the network and is analogous to a vector register. The ξ -register has n locations. The i th register slot is connected to a port attached to the i th neighboring module.
3. Network support for simultaneous bidirectional transfers through all ports.
4. An input and an output buffer to inject packets into and receive packets from the ξ -register. These buffers must be large enough to handle the maximum transfer occurring during any single cycle. For the transpose, the average transfer is two per cycle while the the maximum is somewhat higher. For the 8×8 case the maximum is 3. Although the maximum value has not been derived analytically, it has an upper bound of $\log_2 P$; experimental results indicate that it is smaller.

One of many ways to complete the element-based implementation is sketched in Table 3. The ξ -register size is $n = \log_2 P$ while the input and output buffer sizes are 3. A cycle has two phases, internal and external. During the external phase, the input buffer is loaded from the processor/network interface, the output buffer unloaded to the same, and the ξ -register contents are exchanged with the neighbors. During the internal phase, elements are moved from the input buffer to either the ξ -register or to the output buffer; one or more elements of the ξ -register are moved to the output buffer; and the ξ -register contents are reordered.

	T_0	T_1 external	T_1 internal	T_2 external	T_2 internal	T_3 external	T_3 internal	T_4 external	T_4 internal
ξ_0	$x_{4,5}$	$x_{5,4}$	$x_{4,7}$	$x_{5,6}$	$x_{4,3}$	$x_{5,2}$	$x_{4,1}$	$x_{5,0}$	
ξ_1	$x_{6,5}$	$x_{4,7}$	$x_{6,1}$	$x_{4,3}$	$x_{7,1}$	$x_{5,3}$	$x_{7,5}$	$x_{5,7}$	
ξ_2	$x_{2,5}$	$x_{6,1}$	$x_{3,5}$	$x_{7,1}$	$x_{0,5}$	$x_{4,1}$	$x_{1,5}$	$x_{5,1}$	
input buffer		$x_{3,5}$ $x_{5,5}$		$x_{0,5}$		$x_{7,5}$ $x_{1,5}$			
output buffer			$x_{5,4}$ $x_{5,5}$		$x_{5,6}$		$x_{5,2}$ $x_{5,3}$		$x_{5,0}$ $x_{5,7}$ $x_{5,1}$

Table 3: Contents of processor $p = 5$ during the 4 cycles of an 8×8 transposition.

Tasks other than transpose can require a slightly more complex implementation. Some communication schedules require that data elements be sent to the output buffer, only to be re-input on a later cycle. Because of the complexity of processor/network interfaces, however, a preferable implementation includes an additional buffer (e.g. called *temp*) which holds data that have arrived from elsewhere, but that are not to be transmitted until a later cycle. Note that this requires a $3 \log P \times 3 \log P$ crossbar, plus whatever additional support is necessary to deal with recycling elements.

We now summarize the single cycle operations on the hardware just outlined that are integral to the theoretical model. Other similar sets of operations also work. During the external phase:

- The network interface gets elements from the output buffer that have arrived at their destination.
- The network interface puts elements into the input buffer that will be needed during the next internal phase.
- Elements in the ξ -register are transmitted and received by the corresponding nodes.

During the internal phase:

- Elements from any location of the input buffer are transferred to any location in the ξ -register.
- Elements from any location of the input buffer are transferred to any location in the output buffer.
- Elements from any location of the input buffer are transferred to any location in the ξ -register.
- Elements from any location of the ξ -register are transferred to any location in the ξ -register.
- Additional transfers through the temp register may be required.

An alternative implementation of the element-based model has N -sized ξ -registers and results in an implementation that looks more like that in Table 2. Although simpler conceptually, the reordering step is much more complex; we therefore believe that the previous implementation is more realistic.

The instantiation of the packet-based model is similar to the element-based model, but with an added requirement. Whereas in the element-based model only one element per cycle is sent to each neighbor, in the packet-based model all elements in the node that go through each neighbor are bundled together into a packets and transmitted. The transmission time is measured in terms of latency and bandwidth of the communication. Packet formation is assumed to take place in a single cycle.

4 Extensions to More Realistic Models

In this section we continue justifying the viability of scheduled communication by giving results for more realistic machine models. In the next section we address many of the remaining issues in creating a physical implementation. One concern typical of theoretical models, the mapping of data to processors, is already much more accurate here than it was in models popular through the early '90s. Rather than mapping a small number of elements from each parallel variable to each processor, the model used here realistically assumes that the square-root of the total be so assigned. Other issues are now addressed.

4.1 Wire Latency

The theoretical model assumes similar time for wire and switching latency. In reality, wire latency can be significantly greater. We introduce a new model, the *streaming element-based model* to refer to the case where there are one or more elements in flight between nodes. The timing properties of this model are virtually identical to those of the packet-based model. Latency L is the time it takes to transmit the first element to its destination processor; all subsequent transmissions arrive at time U where U^{-1} is the bandwidth. As in the packet-based model (and element-based), wire utilization is nearly 100% and the elements are priority ordered. As a consequence, the performance of the streaming element-based and packet-based models are nearly identical. The streaming element-based model, however, does not require single cycle packet formation.

A moderate increase in wire latency therefore has little effect on performance, especially as P increases. Assume a wire latency of 10, a switch latency of 1, and channel bandwidth of 1 element per cycle. Then a transpose takes 208 cycles rather than 128 for $P = 256$, 612 cycles versus 512 for $P = 1024$, and 2168 cycles versus 2048 for $P = 4096$. See Figure 1 for more examples.

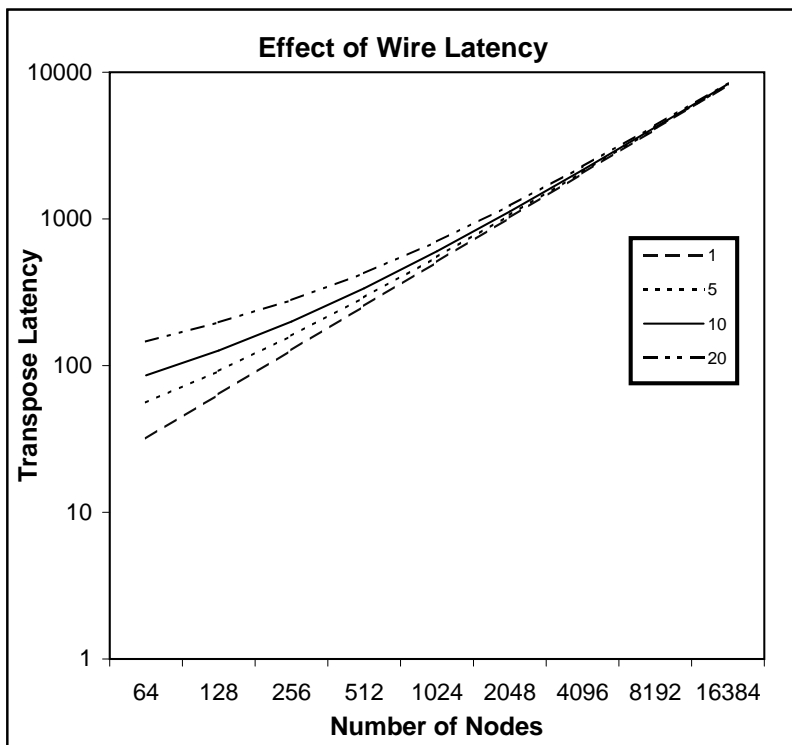


Figure 1: Graph showing performance of matrix transform accounting for number of processors and wire latency.

4.2 Bounding the Number of Ports

Unbounded ($\log_2 P$) ports is unrealistic: here we examine the performance for bounded degree networks. Even in the strictest comparison, where port bandwidth is held constant as the number of dimensions is increased—clearly an impractical case—meshes of 3 and 4 dimensions yield quite respectable performance. The key result is from Bertsekas and Varvarigos [5] who developed a class of optimal algorithms for isotropic communication tasks on networks that are symmetric about any node. For example, the time required for the previously defined transpose problem on a d -dimensional torus is $P^{\frac{1}{d}}(UP/8 + Ld/2)$. Sample numbers are plotted in Table 4. Further results are graphed in Figure 2.

dimensions	number of nodes				
	64	256	1024	4096	16384
2	2.2	2.6	4.3	8.2	16.1
3	1.1	1.0	1.4	2.0	3.3
4	.80	.66	.76	1.0	1.4
$\log_2 P$	1.4	.81	.60	.53	.51

Table 4: Performance of the streaming element model in cycles per arrival per node for an $N \times N$ matrix transpose as the number of dimensions and nodes is varied. Bandwidth is 1 and latency is 10.

We find these results quite impressive: for a three dimensional network with 4096 nodes and a realistic delay model, a matrix can be transposed such that an element arrives at every destination on every other cycle. When communication is integrated with applications, as has been studied by one of the PIs [36], the results are definitive. In particular, performance comparable to a hypercube can be obtained with a 2-D torus if $P \leq 256$, a 3-D torus if $P \leq 10^4$ processors, and a 4-D torus if $P \leq 10^6$.

4.3 Bounding the Number of Pins

Unbounded port bandwidth is also unrealistic: we now integrate a pin constraint into the previous model and recompute performance with respect to number of nodes and dimensions. We assume switch chips with an aggressive, but plausible, number of data transfer pins, say 640. We also assume that the elements have 64 bits. As before, wire latency is 10 cycles and switch latency 1 cycle. Table 5 shows a summary of the results. Further results are graphed in Figure 3.

The key observation is that for three and four dimensional networks, the reduction in the number of arrivals per cycle as P increases remains small. In particular, for plausible numbers of dimensions, the numbers are still remarkable: just over one arrival every other cycle for $d = 3$ and $P = 1024$ and just under one arrival every third cycle for $d = 4$ and $P = 16384$. Recall that in our

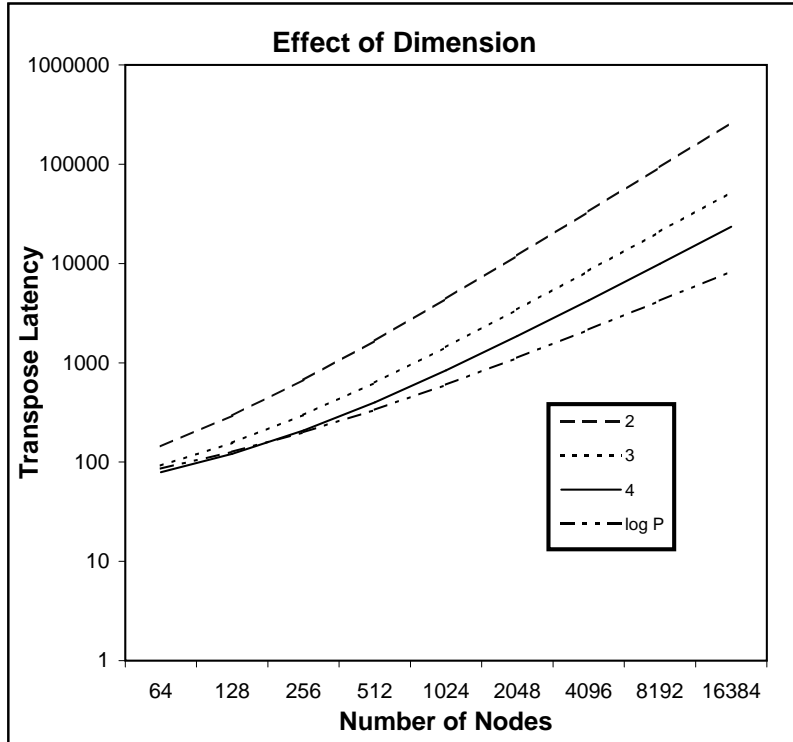


Figure 2: Graph showing performance of matrix transform accounting for number of processors and dimensions. Wire latency is 10 times switch latency.

calculations, the data set size increases with P . Thus in the final example of routing among 16K nodes, the data set is $16K \times 16K = 256M$ 8 byte elements for a total of 2 gigabytes.

4.4 Implication of Gigabit Serial Connections

The latest communication technology favors the use of very high speed serial links, which require only single pin connections (e.g. [38]). The number of ports per switch is likely to remain bounded in the near term, however, if only because of difficulty in routing a large number of gigabit streams through a chip. Still, it is always possible to cascade switches to create a “switching fabric” (e.g. [7]) of almost arbitrary dimension. This does not greatly change the relative benefit of scheduled communication, however: higher dimensional online networks still suffer serious performance degradation from congestion while the scheduled communication model simply reverts back to the result in Subsection 6.1.

5 Implementing the Switch

Another critical question is whether the somewhat abstract theoretical switch described in the previous section can be implemented with (i) operating frequency sufficient to keep the switch

dimensions	number of nodes				
	64	256	1024	4096	16384
2	2.5	3.0	5.1	9.8	19.3
3	1.7	1.6	2.2	3.3	7.6
4	1.6	1.3	1.5	2.0	2.8
optimal (d)	1.6(4)	1.2(6)	1.2(8)	1.3(9)	1.5(10)

Table 5: Same as Table 4 except that here the number of signal pins per node is fixed. Note that the optimal number of dimension is no longer a hypercube.

internals off the critical path, (ii) critical path latency of only a few cycles, and (iii) on-chip storage that is both adequate and whose access does not affect (i) and (ii). In this section we outline how all three conditions are met.

The key result is that although some communications require moderately complex hardware, optimal scheduled communication has the remarkable property that this complexity has only a trivial effect on the critical path timing. In particular, (i) critical path packets never need to be delayed and (ii) this is achieved with the addition of only one extra level of logic on the critical signal path. This is in contrast with on-line routing where there is invariably a trade-off between improved packet selection and added complexity (channel selection, header comparisons, etc.) in processing packets potentially on the critical path [30].

All optimal schedules run the switch at full utilization. That is, on every cycle, one element arrives at every input network port and one element exits on every output network port. For many schedules (see e.g. Table 2), every element that arrives at a network input register is immediately routed to a unique network output register. These types of schedules are trivially supported with a crossbar (e.g. comprised of MUX's) with data flowing through the data inputs and the schedule feeding through the select inputs.

For other schedules, however, individual packets are delayed by some number of cycles, although the ports are still running at full capacity. These schedules are supported by adding (packet-wise) parallel-input/serial-output delay queues to the output ports. A MUX selects between the output of the main crossbar and the delay queue. Because the schedule is optimal, at most one packet will be written into any particular queue slot during any cycle. Since every slot could be written from any input port, this requires a MUX for every slot of every output queue. This is reasonable for a small number of slots (like having a small switch). Also, since the queued packets are not needed for at least one cycle, the queue latency is completely hidden.

It is possible that there exist communication patterns whose optimal schedules require packet delays larger than, say, four cycles. To support these communications without a quadratic increase in switch complexity, another level of packet storage can be added. One example is a random access

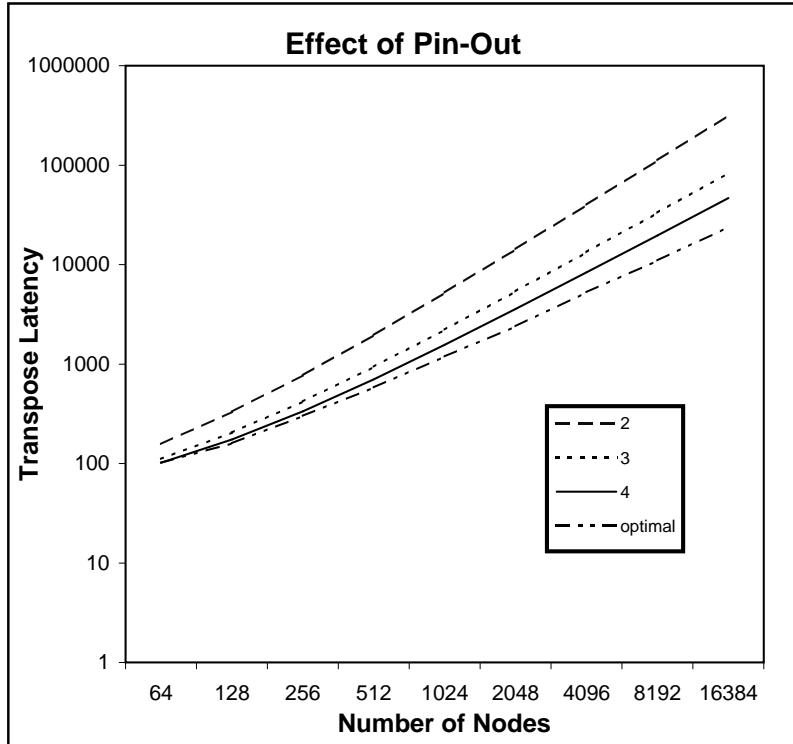


Figure 3: Graph showing performance of matrix transform accounting for number of processors, dimensions, and pin-out. Wire latency is 10 times switch latency.

store for which the output port maintains a queue of pointers. Long-delay packets are shunted there and retrieved into the delay queue as their transmission time approaches. This and other still-lower levels of storage can be constructed in slower more compact technologies such as SRAM or even DRAM. The latency variation between levels is handled with the standard technique of trading off time for path width.

The main point with building switches for scheduled routing is therefore that, since the schedule is optimal, delays are necessarily only added for packets off of the critical path. Delay queues and slow, compact storage are only used when a packet is not needed for a while, not because undesired congestion has occurred and a large number of packets suddenly needs to be stored.

The switch is programmed with microcode instructions whose fields consist of the selects for the multiplexors and delay queues. For a three dimensional switch, the microcode word size is less than 128 bits. As with packet storage, microcode sequencing is entirely off of the critical path: since the execution is entirely ordered, the microcode execution can be pipelined almost to an arbitrary depth.

The investigators and one of our students Kurt Olin (also a Principal Engineer at Compaq Computer Corporation) have conducted an initial timing estimate of this design. Assuming 32 bit

datapaths and three dimensions and using the LSI Logic G12 .18 micron standard cell process, we obtained an operating frequency of nearly 1GHz.

6 System Issues

In this section we discuss practical issues in implementing systems based on programmable switches. We concentrate especially on stream alignment and on interface latency. The former is perhaps the most critical technical question that needs to be addressed. The latter is an obvious necessary condition for low-latency communication.

6.1 Overview of Potential System Implementations

Basing systems on an existing standard such as InfiniBand allows us to leverage commercial technology for compute nodes, node interfaces, transmission, software, IP blocks, and switching. InfiniBand is a complex standard with multiple levels of implementation possible. Useful for our purpose is the raw packet format, which retains the software, mechanical, and electrical characteristics, but is meant for interfacing with networks running other protocols. In particular, use of this format will allow us to use commercial transceivers and a standard API while giving us the flexibility to experiment with the switch architecture.

At least two system implementations are immediately viable: one is based on standard server modules, another on systems-on-a-chip. Clustering servers for HPC is a mainstream technique with thousands in use. Development continues with increasingly powerful and high density nodes as well as faster interconnects (e.g. the Green Destiny cluster [37]). The breakthrough with InfiniBand, however, is that it standardizes a low-latency network interface. Rather than plugging into the I/O bus, the NIC connects directly to the “north bridge” (Intel terminology) or system controller, off the system bus [17]. There are already commercially available Host Channel Adapters providing exactly that capability [23].

The QCDOC architecture [9, 8] is an ideal template for an implementation where a system-on-a-chip combines the compute node with the switch. QCDOC was developed to address the lack of support in available systems for fine-grained, low-latency, Euclidian communication. The first generation consisted of PowerPC boards with a simple communication ASIC (basically a set of FIFOs). The latest version takes advantage of IBM IP blocks to implement the entire node on a single ASIC. These blocks include the processor, memory, memory controllers, buses, and off-node links. Only the communication controller is custom logic.

There are at least two other system implementations of interest. One is to put the entire multicomputer onto a single chip or substrate. This is a topic already investigated by one of the PIs (see e.g. [1] and the brief description in Section 1). Placing 256 processors on a single substrate

will be viable in near-term technology. In this scenario, a switch interface can be integrated with the processor core at a very low level, for example with the SPARC, through the coprocessor interface. Long term, the most interesting implementation is perhaps in a vector multicomputer. As described earlier, one of the eventual goals of this project is chained communication and computation. Part of this goal includes creating direct interfaces between the network and interleaved memory and between network and vector processor.

6.2 Software Issues

Scheduled communication is only useful if the network is the bottleneck. This has often not been the case but, fortunately, much emphasis has been given in recent years to reducing software latency between when the communication request is made in the application and when the data actually reach the hardware interface. Techniques include avoiding the operating system, using memory mapped I/O, and single copy. Although these ideas are not new (see e.g. [10, 14]), they are now being firmly integrated into mainstream HPC [27, 3]. They have also been used by one of the PIs and his students in network performance studies [11, 21, 22]. In particular, we have created and used an MPI interface that passes the data directly through to the network interface. The Infiniband standard supports VI-like queues which fit naturally into this framework.

At a higher level, we assume a collective communication model such as is provided by MPI directives. In doing so, we support the major climate and weather modeling codes (and many others including the NAS benchmark suite). Our programmer's model is similar except that a switch schedule must be specified when scheduled communication is requested. Communication has both synchronous and asynchronous forms: the former for implementation with a barrier model, the latter for chaining computation with communication.

6.3 Interconnection Issues

Transmission. Perhaps the greatest advantage of using Infiniband is that any reasonably sized system can be connected using only off-the-shelf channels and wires.

Flow Alignment. The scheduled communication model requires that data flowing through each switch be aligned. Using high-speed serial transmission, however, precludes global synchronization. In any case, global synchronization, although possible [25], is quite challenging and best avoided. Instead we rely on a variation of a technique called retiming [6]. In retiming, it is assumed that various signals need to be aligned (e.g. in a bit-parallel transmission) and that a delay of a few cycles is acceptable. One further assumes that data rates are nearly identical and that we have a mechanism for centering each bit. The problem is then to find where the bit streams are in relation to one another. Once this has been done, all that is required is to delay all but the lagging stream

by the appropriate amount. The delays are carried out with programmable FIFOs. In Bolotski's work, the delays were calculated by using a reference bit stream. Here, another mechanism is preferable: each packet is tagged with a sequence number. The switch only needs to be sure that it processes packets with the same sequence number at the same time. Note that in our version of retiming, what is being aligned is not the bits in each stream, but rather entire packets after they have been read into the switch.

The question is how long is the longest delay likely to be? Or rather, how small can we make the longest delay? This will determine the length of the alignment FIFOs and the number of bits required for the (recirculating) sequence numbers. Clearly, initial synchronization and a throttling mechanism will be required. Although this is one of the critical questions to be investigated in this research, we believe that the answer will be reasonable, certainly less than, say, fifty 256-bit packets and likely substantially less than that. The resulting added delay for the entire collective communication would then be at most a few microseconds with a goal of a few 10's of nanoseconds.

Topology. Number of dimensions was discussed in the previous section. Under current connection schemes (PCB boards, connectors, cables, etc.), implementation of three and four dimensional networks is straightforward using direct connections; more than that are probably not necessary. In any case, higher dimension networks are can be built using cascaded switches just as in on-line routing.

6.4 Other Issues

Although there has been research in scheduled communication algorithms and programmable switches (e.g. [2, 4, 26, 28]) we are not aware of any recent machines that take advantage of this paradigm. In other sections we have justified the use of programmable switches because of their superior performance; here we address some concerns.

Storing and loading the schedules. Several points make these issues straightforward: (i) communication patterns get reused many times as simulations progress through huge numbers of iterations, (ii) a relatively small number of bits is required for each packet-cycle of the schedule, and (iii) we know far in advance exactly when each switch microinstruction is needed. The first point means that even if one communication is delayed because a schedule must be loaded, many other communications will not be. The second point means that even for the largest MPPs, at least 20 complete schedules can be stored on-chip and several hundred in a neighboring schedule store. The third point means that streaming schedules onto the chip from a schedule store for just-in-time use is also viable.

How useful are the communication patterns for which there are known optimal schedules? The algorithms referenced in Table 1 optimize communication for the most time consuming

parts of weather and climate modeling codes.

Can we find similarly efficient schedules for (i) Other $P (< N)$, (ii) Other topologies, and (iii) other communication patterns? Perhaps even more important than the previous answer is that the communication support provided by this architecture can be used to implement *any* optimal or near optimal communication algorithm. The development of such algorithms will be the subject of research both as a part of this proposal and by the communication sciences community in general.

On-line versus off-line routing – prior knowledge of communication patterns. There will always be some communications whose properties are not known until just before they occur. In many (if not most) computational applications, however, and certainly in the primary weather and climate codes, this is not the case. The programmer knows where the time consuming internode communication occurs, what type it is, and the data set sizes in relation to the overall model. The communication details, especially for the most time-critical operations, can therefore usually be determined by load time if not sooner. Even in the case where they need to be determined at run time, this computation would only take a small fraction of the time it takes to compute all of the start-up parameters (often several minutes for complex codes).

When on-line routing is necessary. Scheduled is obviously not always possible. To support those cases where on-line routing is required, the scheduled communication router should be combined with a conventional router. This is straightforward, requiring only an additional bit in the header and a trivial amount additional routing circuitry (beyond what is already needed for each router). But since the most time consuming communications are likely to be using the scheduled router, the on-line router can be comparatively simple.

7 Preliminary Experimental Comparison with On-line Routing

Some of our previous work has addressed networks optimized for use in single chip multicomputers. Although this is admittedly a restricted domain, it is one for which we can compare on-line with off-line routing modes completely as “apples-to-apples,” i.e. all the way from communication pattern through to silicon simulation.

In [15] we did an exhaustive parameter study accounting for virtual cut-through versus worm-hole, input versus output queuing, number of virtual channels, size of channels, arbitration policies, etc. Because of our domain, we assumed two dimensions, inter-switch latency of 1 cycle, and 32 bit paths. Significantly we accounted for switching latency variations caused by parameter changes and so were able to get a true capacity measure for each design/work-load combination: flits per node per nanosecond.

Although the optimal switch configuration varied slightly with communication pattern, it was possible to determine that the highest performance, irrespective of chip area, would be obtained with a switch with the following characteristics: virtual cut-through, input queuing, two virtual channels per physical channel, and random arbitration. Although increasing queue size increased capacity monotonically, little benefit was obtained in making queues bigger than 2-3 packets, although we did use small packets in our workloads (24 flits).

The results for the matrix transpose for various network sizes are shown in Table 6 and indicated factors of 6 to 13 performance advantage for the offline routing network. Also note that this comparison does not account for the regular and predictable delivery of elements to their destinations by the off-line router, potentially the key advantage of this model as it allows the chaining of communication with computation.

	network configuration		
	8x8	16x16	32x32
on line	729	7346	59038
off line	83	1134	4499

Table 6: On-line versus off-line routing performance in nanoseconds for a transpose workload of N elements per node for $N \times N$ networks.

8 Discussion and Conclusion

We have proposed work on a collection of algorithms and designs that could greatly improve the efficiency of large-scale parallel computers used in grand challenge applications. We are confident that in the applications with which we are most familiar, weather and climate modeling, the increase in effective computing resources available to researchers would be enormous.

Other work has been done previously in hardware and algorithms for scheduled communication [4, 26, 2, 28], much of which has already been described in this proposal and upon which we will build. The work proposed here differs from the previous efforts in two ways. The first is that it concentrates on supporting algorithms that are optimal in that they possess what we call the $O(1)$ communication property. This focus permeates through all aspects of the proposed work: developing further optimal algorithms, supporting these algorithms as efficiently as possible with existing and projected hardware technology, taking advantage of the properties of these optimal algorithms to optimize switch designs, and (in the long term) recoding applications to integrate chained communication and computation. The second is that we base our designs on readily available and more importantly, readily usable technology. In particular we leverage the latest multi-gigabit standard and FPGA technology.

In at least one way, the approach proposed here represents the application of the most fundamental architectural technique: improve application performance by integrating maximal application knowledge with the latest technology.

References

- [1] Adapa, C. Implementation issues in building a multicomputer on a chip. Master's thesis, Department of Electrical and Computer Engineering, University of Houston, 2001.
- [2] Annexstein, F., and Baumslag, M. A unified approach to off-line permutation routing on parallel networks: the interaction of architecture and operating system design. In *Proc. of the 2nd ACM Symposium on Parallel Algorithms and Architectures* (1990), pp. 398–406.
- [3] Athanasaki, M., Sotiropoulos, A., Tsoukalas, G., and Koziris, N. Pipelined scheduling of tiled nested loops onto clusters of SMPs using memory mapped network interfaces. In *Proc. Supercomputing 2002* (2002).
- [4] Beetem, J., Menneau, M., and Weingarten, D. The GF11 parallel computer. In *Experimental Parallel Computing Architectures*, J. Dongarra, Ed. North Holland, Amsterdam, 1987.
- [5] Bertsekas, D., Ozveren, C., and G.D. Stamoulis, P. Tseng, J. N. T. Optimal communication algorithms for hypercubes. *Journal of Parallel and Distributed Computing* 11 (1991), 263–275.
- [6] Bolotski, M. *Abacus: A Reconfigurable Bit-Parallel Architecture for Early Vision*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1996.
- [7] Booth, R. Enabling HPC with scalable InfiniBand switching. In *Intel Developer Forum* (2002).
- [8] Boyle, P.A., et al. Status of the QCDOC project. In *Proceedings of Lattice 2001* (2001).
- [9] Chen, D., et al. QCDOC: A 10 teraflop scale computer for lattice QCD. In *Proceedings of Lattice 2000* (2000).
- [10] Dally, W. J., and et al. The Message-Driven Processor: A multicomputer processing node with efficient mechanisms. *IEEE Micro* 12, 2 (1994), 194–205.
- [11] DeFord, M. Test and integration environment for PCI coprocessor cards. Master's thesis, Department of Electrical and Computer Engineering, University of Houston, 2001.
- [12] Edelman, A. Optimal matrix transposition and bit reversal on hypercubes: All-to-all personalized communication. *Journal of Parallel and Distributed Computing* 11, 3 (1991), 328–331.
- [13] Hammond, S., James, R., and Loft, R. Computational considerations for tera-scale climate modeling. In *Joint US/Japan Workshop on Next Generation Climate Models for Advanced Computing Facilities* (1999).

- [14] Henry, D., and Joerg, C. A tightly coupled processor network interface. In *Proc. Architectural Support for Programming Languages and Operating Systems* (1992), pp. 111–122.
- [15] Herbordt, M. C., Olin, K., and Le, H. Design trade-offs of low-cost multicomputer networks. In *Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation* (1999), pp. 25–34.
- [16] Infiniband Trade Association. *InfiniBand Architecture Specification*, 2002.
- [17] Intel Corporation. Enterprise interconnect technologies. In *Intel Developers Forum* (2002).
- [18] Johnsson, S., and Ho, C.-T. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers* 38 (1989), 1249–1268.
- [19] Johnsson, S., and Ho, C.-T. Optimal communication channel utilization for matrix transpose and related permutations on binary cubes. *Discrete Applied Mathematics* 53 (1994), 251–274.
- [20] Lazou, C. ECMWF workshop user experience with NEC SX-6 and IBM P3/4. *HPCwire* 11.28.02 (2002).
- [21] Lin, C. CAL-Sim: An environment for evaluating multicomputers with respect to MPI applications. Master’s thesis, Department of Electrical and Computer Engineering, University of Houston, 2001.
- [22] Mande, A. Performance prediction of message passing communication in distributed memory systems. Master’s thesis, Department of Electrical and Computer Engineering, University of Houston, 2002.
- [23] Mellanox Technologies, Inc. *Mellanox and HPC Clustering*, 2002.
- [24] Olin, K. High-performance embedded multicomputer networks. Master’s thesis, Department of Electrical and Computer Engineering, University of Houston, 1999.
- [25] Pratt, G., and Chung, C.-P. Distributed synchronous clocking. *IEEE Trans. on Parallel and Distributed Systems* 6, 3 (1995), 314–328.
- [26] Rana, D., and Weems, C. C. The ICAP parallel processor communication switch. In *Proc. of the IEEE Int. Symp. on Circuits and Systems* (1989), pp. 126–129.
- [27] Shivam, P., Wyckoff, P., and Panda, D. EMP: Zero-copy OS-bypass NIC-driven gigabit ethernet message passing. In *Proc. of Supercomputing 2001* (2001).

- [28] Shoemaker, D., Honore, F., Metcalf, C., and Ward, S. Numesh: An architecture optimized for scheduled communication. *Journal of Supercomputing* 10, 3 (1996), 285–302.
- [29] Stout, Q., and Wagar, B. Intensive hypercube communication: Prearranged communication in link-bound hypercubes. *Journal of Parallel and Distributed Computing* 4 (1987), 95–115.
- [30] Stunkel, C.B., et al. The SP2 high-performance switch. *IBM Systems Journal* 34, 2 (1995), 185–204.
- [31] Swarztrauber, P. The vector parallel paradigm. In *Advanced Mathematics: Computations and Applications* (1995), pp. 675–687.
- [32] Swarztrauber, P. MPP: What went wrong? can it be made right? In *Making its Mark: The Use of Parallel Processors in Meteorology*, G.-R. Hoffmann and N. Kreitz, Eds. World Scientific, Reading, UK, 1997, pp. 1–15.
- [33] Swarztrauber, P. *Multipipeline multiprocessor system*. US Patent 5,689,722, 1997.
- [34] Swarztrauber, P. Transposing arrays on multicomputers using de Bruijn sequences. *Journal of Parallel and Distributed Computing* 53 (1998), 63–77.
- [35] Swarztrauber, P. The vector multiprocessor. *International Journal of High Speed Computing* (2000).
- [36] Swarztrauber, P., and Hammond, S. A comparison of optimal FFTs on torus and hypercube multicomputers. *Parallel Computing* (2000).
- [37] Warren, M., Weigle, E., and Feng, W.-C. High-density computing: A 240-processor Beowulf in one cubic meter. In *Proc. Supercomputer '02* (2002).
- [38] Xilinx, Inc. *RocketIO Transceiver User Guide*, 2002.
- [39] Xilinx, Inc. *Virtex-II Pro Platform FPGA User Guide*, 2002.