Scalable NIC-based Reduction on Large-scale Clusters

Adam Moody, Dhabaleswar K. Panda, Juan Fernandez, and Fabrizio Petrini

Technical Report
OSU-CISRC-5/03-TR27

Scalable NIC-based Reduction on Large-scale Clusters*

Adam Moody[1,2] Juan Fernandez[2] Fabrizio Petrini[2]
Dhabaleswar K. Panda[1]

[1]Department of Computer & Information Science
Ohio State University, Columbus, OH 43210 USA
{moody,panda}@cis.ohio-state.edu

[2]CCS-3 Modeling, Algorithms, and Informatics Group
Computer and Computational Sciences (CCS) Division
Los Alamos National Laboratory, NM 87545, USA
{juanf,fabrizio}@lanl.gov

Abstract

Many parallel algorithms require efficient support for reduction collectives. Over the years, researchers have developed optimal reduction algorithms by taking into account system size, data size, and complexities of reduction operations. However, all of these algorithms have assumed the fact that the reduction processing takes place on the host CPU. Modern Network Interface Cards (NICs) sport programmable processors with substantial memory and thus introduce a fresh variable into the equation. This raises the following interesting challenge: *Can we take advantage of modern NICs to implement fast reduction operations?* In this paper, we take on this challenge in the context of large-scale clusters. Through experiments on the 960-node, 1920-processor ASCI Linux Cluster (ALC) located at the Lawrence Livermore National Laboratory [24], we show that NIC-based reductions indeed out scale host-based algorithms in terms of reduced latency and increased consistency. In particular, in the largest configuration tested —1812 processors— our NIC-based algorithm sums single-element vectors of 32-bit integers and 64-bit floating-point numbers in 73 μs and 118 μs, respectively. These results represent respective improvements of 121% and 39% over the production-level MPI library.

1 Introduction

Many high-performance computing applications depend critically on efficient reduction algorithms. Recent performance evaluation studies show that large-scale scientific simulations spend up to 60% of their time executing reductions [19]. Similar results have been provided by an in-depth analysis of the scientific workload at Lawrence Livermore National Laboratory [10]. Reduction algorithms which minimize latency will thus substantially reduce the overall run-time of such programs.

The problem of developing efficient reduction algorithms has proven to be a rather rich area of research. Reduction collectives involve both communication (data transfer) and processing (data reduction operations), and so efficient implementations must consider characteristics of the network, the processors, and the interplay between the two. In other words, the design space for developing efficient reduction algorithms is quite large. Over the years, many researchers have committed significant time in order to derive optimal and scalable algorithms [2, 3, 4, 5, 15, 18]. These algorithms differ in their assumptions of the underlying system characteristics. During all of this effort, however, designers have commonly assumed processing must be performed by the host CPU.

Network interface cards for modern cluster interconnects, such as the Quadrics Elan [20], provide programmable processors and substantial memory. This added capability allows the host processor to delegate certain tasks to the NIC processor. To differentiate where the task is actually performed the terminology "host-based" and "NIC-based" have been introduced. There are various reasons one may wish to do such a thing, and in this paper we discuss two of them with regard to reduction. Namely, we find that NIC-based reductions can offer both significantly lower latency and better consistency than host-based algorithms.

This paper presents our scientific and technical contributions. We first developed a detailed model to predict the performance of various NIC-based reduction algorithms on Quadrics. A new model was necessary since previously existing ones failed to accurately account for important characteristics of the Quadrics Elan. Guided by this model, we then implemented a NIC-based algorithm that uses emulated floating-point operations in the Quadrics NIC. This algorithm operates without the intervention of the host processors. Finally, we provide an enhanced version of our algorithm when reducing larger vector sizes.

Experimental results show that our NIC-based reduction algorihtm provides reduced latency and increased consistency in the common case. In particular, in the largest configuration tested on the ALC [24] —1812 processors— our NIC-based algorithm sums single-element vectors of 32-bit integers and 64-bit floating-point numbers in 73 μs and 118 μs, respectively. These results represent respective improvements of 121% and 39% over the production-level MPI library. In addition, since the NIC-based algorithm is not subjected to certain host-level interference, we found that the performance of our algorithm is also much more predictable. To the best of our knowledge, our results are the best performance achieved on any large-scale parallel computer, both in terms of latency scalability and consistency.

The rest of this paper is organized as follows. Section 2 outlines the relevant characteristics of the Quadrics network. Section 3 describes important trade-offs involved between implementing host-based and NIC-based collectives, and Section 4 discusses design issues, solutions, and simplifications specific to reductions. Section 5 presents the algorithm and associated model we developed, while Section 6 provides the results we obtained. Finally, some concluding remarks are given in Section 7.

2 The Quadrics Network

We implemented our NIC-based reduction algorithms on the Quadrics network, a modern cluster interconnect technology [20]. Quadrics is based on two building blocks: a programmable network interface card called the Elan [21, 22] and a low-latency high-bandwidth communication switch called the Elite [23].

The Elan resides on the PCI bus and interfaces a processing node, containing one or more CPUs, to the network. The Elan itself is quite powerful. It contains a user-programmable, multi-threaded, 32-bit 100 MHz RISC-based processor and a substantial 64 MB bank of local SDRAM memory, along with an MMU and other sophisticated processing features. All of this hardware is provided at the NIC to aid implementation of higher-level message protocols without requiring explicit intervention from the host CPU. In order to better support this usage model, the processor's instruction set includes extra instructions to construct network packets, manipulate events, and schedule threads. This functionality is used to provide extremely low message processing overhead at the nodes of the network.

The Elan divides messages into a sequence of fixed-length transactions for efficient transfer through the network. The primary communication primitive supported by the network is the Remote DMA (RDMA). RDMAs allow for one-sided data transfer between remote processes, i.e. the remote process need not explicitly participate in the exchange. Transfer operations include PUT, which transfers data to a remote address space, and GET, which acquires data from a remote address space. Both operations can access either host- or NIC-level memory.

The network itself is worm-hole routed and circuit-switched. It consists of Elite switches interconnected in a fat-tree topology [17]. Each Elite provides the following features: 8 bidirectional links supporting two virtual channels in each direction, a full crossbar switch, a raw transmission bandwidth of 400 MB/s (325 MB/s at MPI level) per link with a low cut-through latency of 35 ns, and hardware support for collective communication including barriers and broadcasts.

3 NIC-based vs. Host-based – Pros and Cons

In this paper we show how NIC-based reduction algorithms can outperform host-based versions in two important ways: reduced latency and increased consistency. In this section, we will describe how exactly this is accomplished. We also discuss the major penalties encountered when implementing reductions at the NIC-level, namely, host-NIC synchronization cost and reduced computational performance.

3.1 Advantages – Reduced Latency, Increased Consistency

NIC-based collectives can be completed significantly faster than host-based versions. Modern cluster interconnects, such as Quadrics, support very low message latencies; so low in fact, that PCI bus transaction time is substantial compared to the latency between nodes. By implementing collective communications at the NIC, as opposed to the host, many extraneous PCI bus transactions can be eliminated. This can significantly reduce the total operational latency.

Collective operations, by their very nature, require a series of related messages to be exchanged between nodes involved in the collective. In host-based implementations, the host must handle each of these messages. In order to do so, each message must be relayed between the host and the network via PCI bus transactions. NIC-based implementations, on the other hand, handle messages immediately at the NIC, avoiding most of these trips through the PCI bus. In fact, NIC-based implementations suffer from such costs only at the very beginning and very end of the operation. If there are a lot of messages in between, the total savings can amount to a lot. This means that NIC-based collectives can scale substantially better than host-based versions as the size of the cluster increases.

Thus far, the majority of NIC-based research has taken focus on this advantage [6, 7, 8, 9, 13, 14]. In the process of further investigating how this established advantage extends to the realm of reductions, we found a new and much more significant advantage that NIC-based collectives provide when running on large-scale systems.

NIC-based collectives show dramatically reduced latency and increased consistency over host-based versions when used in very large-scale clusters. It happens that process interference at the host level turns out to be a major problem on large clusters. To demonstrate this, observe Figure 1. This figure shows the latency measured for a barrier and a reduction when using both one and two processes per node. Note the dramatic latency deviation for each operation when two processes are used on each node, as opposed to just one.

In this system, there are two physical processors per node. When the collective involves only one process per node, there is a spare processor on which the node may run various system threads. However, when both processors are used by the collective, at least one of the processes is forced to share its processor with the system threads. This interference is responsible for the drastic drop in performance.[19]

Basically, the problem arises since host-based processes in charge of handling intermediate messages during the collective may be subject to process swapping. Unlucky intermediate nodes may be swapped just before processing an incoming message. In this case, the collective will stall until the process is

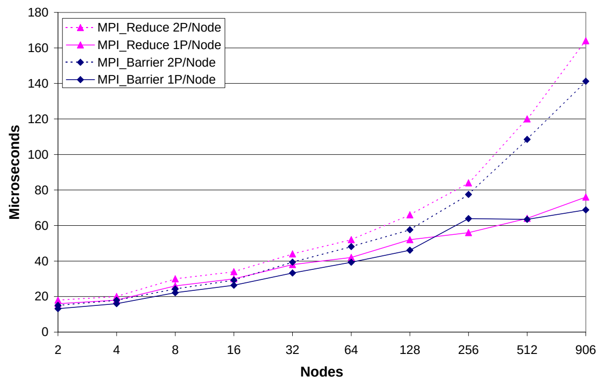

Figure 1: MPI Barrier and Reduce Latencies

swapped back to handle the message. This leads to poor performance, much like problems observed in job scheduling on large systems when using a local scheduling approach. The problem tends to manifest itself on large systems more so than on small systems, because larger collectives require larger algorithmic tree structures. Larger trees in turn require more intermediate nodes. Thus, there are simply more chances that some intermediate processes will be interfered with.

In addition to increased latency, one may immediately understand that this is a rather non-deterministic phenomenon, which leads to a large variance in operational latency from one collective invocation to another. Thus, the same process swapping problem simultaneously increases average latency and decreases operational consistency.

As host-level process swapping is inherently a host-based problem, NIC-based algorithms can avoid it altogether. As a result, NIC-based collectives can complete with drastically better latency and in a more consistent fashion.

3.2 Disadvantages – Overhead, Slower NIC Processor

Even though the NIC carries out the actual collective in the NIC-based implementations, the host must communicate to the NIC, among other information, what operation is to be done, which data is to be processed, and when the operation is to start. Also, the NIC must notify the host of the operation's completion. This process is termed host-NIC synchronization.

Host-NIC synchronization introduces some overhead which must be compensated before NIC-based collectives can be beneficial with respect to latency. As currently implemented, this host-NIC synchronization adds 2 to 3 microseconds of overhead to the total operational latency. However, it should also be noted that this overhead can be largely avoided by overlapping it with other operations, and is thus really of minor concern.

The most important issue to be considered is that of the NIC processor. The user-programmable processor on the NIC is considerably slower than the host processor (as much as 25 times slower on the machines we used). Different processing requirements by different algorithms and different operations make this a very significant difference. Basically, this difference places a limit on the complexities of the algorithms and operations which may benefit from NIC-based implementations. To make matters more complicated, a substantial lack of processing functionality typically exists as well. For example, there is no hardware-based floating-point support on the Quadrics Elan. The limitations of the NIC CPU proved to be the toughest design issue we encountered in our work.

4 Design Issues and Initial Observations

We extend NIC-based collectives to the realm of reductions. Reductions are computationally intensive collectives, and as a result, the slower and less functional NIC CPU becomes a limiting factor. In this section, we probe the sensitivity of the Quadrics Elan to computational requirements, and make note that, fortunately, the common case in many programs does not require large amounts of computation. Thus, even with limited processing power, NIC-based reductions present a viable option.

4.1 Complications – Processing Speed and Capability

As noted above, NIC CPUs are typically much slower than the CPU available at the host level, often by an order of magnitude or so. In addition, NIC CPUs provide less functionality. Knowing these limitations, most of the research in NIC-based work so far has concentrated on collectives which involve little processing. Collectives such as barriers, broadcasts, multicasts, and gathers, simply require intermediate nodes to pass on the received message as is, with perhaps minor data restructuring. Because so little processing is required, these algorithms incur little penalty by running on slower processors, and the overall results have been quite positive.

The success obtained by simpler NIC-based collectives inspired us to investigate more complicated cases, namely reductions. Our design goals were to support NIC-based implementations of the standard MPI reduce and allreduce collectives for 32- and 64-bit integer and floating-point data types, each having min, max, and sum operations.

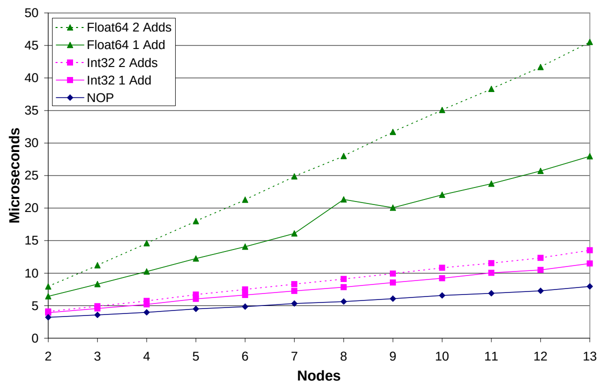

Figure 2: Serial Reduction Latency

The first problem we encountered is the fact that the Elan CPU has no hardware support for floating-point operations. Thus, we were required to emulate floating-point operations in software with integer instructions. Of course, this isn't the first time such a problem has been posed, and fortunately others have worked hard to provide sophisticated software libraries to serve as a solution. In particular, we tackled this problem by porting SoftFloat [25] to the Elan, an IEEE 754 compliant floating-point package written by John R. Hauser, which is freely available to the public domain.

After providing floating-point capability, we investigated the communication and computation characteristics of the Elan. This was accomplished by implementing a very simplistic version of reduce. Basically, a group of N nodes performs a reduce by designating one of the nodes as the root, which is solely responsible for receiving and reducing all of the data. After a synchronization phase, all non-root nodes simultaneously send their data to a corresponding RDMA buffer at the root. Upon receiving all of the messages, the root performs the reduction operation on them in serial order. We will refer to these results at later points in the paper, so it is convenient to provide a name to this algorithm. We simply call it the "serial reduction" algorithm.

Serial reduction tests involving 2-13 processors for various reduction operations and data sizes produced Figure 2. There are a couple important features to take note of.

First, regardless of the operation, all of the curves closely follow a linear trend as the number of nodes is increased. Such a tight trend makes it very easy to model performance, as latency can be predicted using

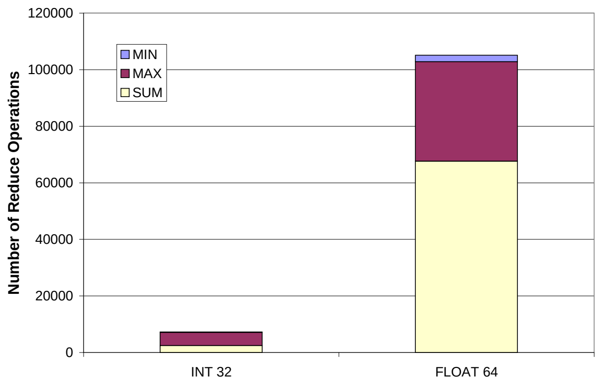

Figure 3: MPI_Allreduce Operator Distribution for SAGE

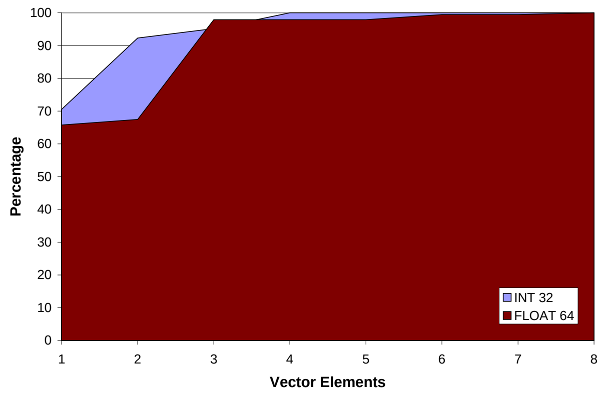

Figure 4: MPI_Allreduce Vector Size Distribution for SAGE

only a couple model parameters. We address this issue in more detail in Section 5, but basically, the intercept is related to the message latency, while the slope represents the reception and reduction time required by the message.

Second, it is more relevant at this time to take note of the reduction latency sensitivity to the operation being performed. Simpler operations scale considerably better than more complicated ones. Even fast operations are rather sensitive to small changes in data size. As could be expected, floating-point operations are especially slow since they must be implemented in software on an already slow processor. In fact, the time to perform a single 64-bit floating-point addition is comparable to the message latency between nodes.

Certainly then, it will be essential to consider both communication and computation costs when designing efficient NIC-based reduction algorithms. It is also clear that NIC-based reductions, even for very simple operations, will perform with reasonably low latency only for small data sizes. Nevertheless, it turns out that even while this is a rather stringent restriction on the class of problems where NIC-based implementations may be valuable, a large majority of the problems posed by practical programs falls within this class.

4.2 Simplifications – Small Data Sizes

Reductions involving simple operations on small data sizes are the common case in many scientific applications. To show this, we profiled the MPI allreduce operations performed during the execution of SAGE [16]. Sage is a program representative of the typical scientific applications running on large-scale, ASCI-class parallel machines. The results are shown in the following figures.

Figure 3 shows the distribution of operation types. First, note that only a few simple types of operations are used by SAGE: minimum, maximum, and sum. Typical reduction operations thus require limited processing. Second, note that floating-point operations far outnumber integer operations. This strongly suggests that, if no hardware-based floating-point support is provided on the NIC CPU, the emulation software should be highly optimized in order to reduce costs in the common case. Though noting this, we made no direct optimizations to SoftFloat for our tests.

Equally important is Figure 4 which shows the cumulative distribution of the data sizes for different reduction operations using both integer and floating-point data types. Direct observation makes a striking point: 97% of all reductions use 3 or fewer elements and 100% use 8 or fewer.

These observations are key. Typical reductions involve simple operations on small vectors, which is the same class of reductions for which one may benefit from NIC-based implementations. In other words, NIC-based reduction implementations will benefit the common case the most. Thus, given the substantial benefits previously mentioned, NIC-based reduction implementations promise to be quite valuable to typical programs, even while considering the limitations imposed by the NIC processor.

5 The Model and the Algorithms

Over the years, many efficient reduction algorithms have materialized, stressing the importance of these collectives. However, a large majority of the existing algorithms are based on models which make assumptions that do not hold when considering NIC-based reductions. In this section, we point out the major problems with the standard models, introduce a model which addresses these problems, and then present an efficient reduction algorithm based on this new model, along with one important optimization.

5.1 Problems with the LogP and Postal Models

Most proposed reduction algorithms are primarily based on one of two simple parallel performance models: LogP [11] or postal [1] [2, 3, 4, 5, 15, 18]. Unfortunately, neither the LogP nor the postal model accurately captures the communication/computation characteristics of Quadrics-based systems, without making significant modification to the models themselves. As a result, the algorithms designed to be most efficient or optimal on systems closely aligned to these models may no longer be the most efficient or optimal when implemented with the Quadrics network.

Problems arise in two important respects. The familiar LogP and postal models each implicitly assume that: 1) the send/receive costs of the underlying system are symmetric, and 2) reduction costs are negligible compared to communication costs. In particular, the LogP model reserves the 'o' parameter to simultaneously represent both the time it takes a sender to send a message as well as the time it takes a receiver to receive one. The postal model normalizes its sole parameter 'λ' to this symmetrical cost. Additionally, neither model explicitly provides a parameter to represent computation time. These prove to be substantial limitations. Both assumptions break down for NIC-based reductions on Quadrics, which involve threads running on relatively slow Elan processors connected to a fast worm-hole routed, circuit-switched network.

Worm-hole routed, circuit-switched systems, such as Quadrics, lead to asymmetrical send and receive costs when sending small messages. This occurs since the sender must wait for a message to worm its way through the network to the receiver, establish a circuit, and then tear it down before sending another message. This process is limited primarily by the latency of the network. The receiver, on the other hand, is free to receive messages as fast as it can pull them off the wire; a process limited by the bandwidth. For small messages, this means that a receiver is able to receive more messages than a sender can send in a given time. While this asymmetry is most prevalent for small messages, as previously noted, reductions which involve vectors of just a few elements are arguably the common case in practical programs. Thus, it is critical that we choose a model which explicitly accounts for this asymmetrical behavior when designing our algorithms.

In addition, unlike host-based algorithms which are largely communication bound, especially for small vector sizes, NIC-based implementations may be either communication or computation bound. For example, on the Quadrics Elan, the cost to perform a single floating-point 64-bit addition is comparable to the network latency. The much slower and less functional NIC CPU can pay a considerable price in computation costs when the vector size is increased even by a single element. This implies that while host-based algorithms may be designed quite successfully by neglecting computation costs altogether, efficient NIC-based implementations are forced to consider such costs. Thus, it is critical that the model explicitly account for reduction costs.

Parameter	Meaning
L	message latency
r(M)	receive cost of a message of size M
c(M, OP)	computation cost of a message of size M, dependent on the operation OP
P	number of nodes
C(OP)	constant due to initial overhead, in general dependent on the operation OP

TABLE 1: Model Parameters

5.2 The Model

Observations of the serial reduction data from the previous section suggest a very simple model. Namely, take note of the linearity of the latency curves. Essentially, the intercept represents message latency while the slope contains information about the receive and reduction costs of a message. These serial algorithms will be the building blocks of any more sophisticated algorithms. So by accurately modeling these building blocks, one can piece together a model for more sophisticated algorithms. In other words, essentially just the slope and intercept of these lines are sufficient to quite accurately predict performance of any proposed algorithms.

With these observations, we define our model as given in Table 1. We will typically suppress the functional parameters M and OP from the various terms. Note with this model it is simple to describe the latency curves from the serial reduction data as:

$$Time \approx C + L + (P - 1) \cdot (r + c)$$

This expression is shown pictorially in Figure 5.

To assign numerical values to the parameters, we extracted the values of r and c from the serial reduction data for various values of M and OP. The terms L

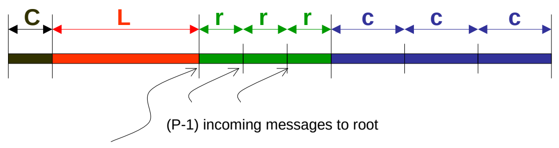

Figure 5: Serial Reduction Latency Model

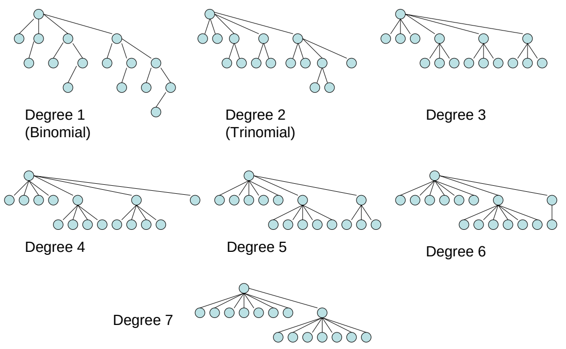

Figure 6: F-nomial Trees of Varying Degrees

and C were fit to the data, and P is obviously given for a particular problem. In passing, we will note that while, in general, r is dependent on the message size, it turns out to be constant for cases we are interested in. This is because we focus on reductions involving vector sizes of a few elements, say up to 8, which typically fit into a single 64-byte fixed-length packet on the Quadrics network. Thus whether we are working with single-element vectors or 8-element vectors, the receive time is the same.

The proposed model parameters also suggest the general form of efficient algorithms. Again looking at the serial reduction data, note that for small messages, the latency L is significantly more than the receive time, r. Thus, due to the circuit-switched nature of the network, the sender may only send a message every L units of time, while the receiver can receive one in every $r \ll L$ units. This is the asymmetrical communication characteristic previously discussed. As a result, nodes in efficient algorithms will tend to receive more often than they send, leading to a class of tree-shaped algorithms. Given that efficient algorithms will take the form of trees, we implemented f-nomial tree algorithms, feeling they were a good balance between structural simplicity and optimality.

5.3 F-nomial Trees – Generalized Binomial Trees

F-nomial trees are generalized binomial trees, which are more familiar structures. Here we will describe f-nomial trees starting from a quick review of the operation of binomial trees. Also, although reduction trees will collapse in on themselves, it is easier to describe the functionality of a tree as it expands. For convenience then, say we are attempting to broadcast a message from the root to all nodes in the tree.

The operation of binomial trees can be described as follows. The algorithm can be broken into distinct phases. At the start of the first phase only the root has a copy of the message to be broadcast. During each phase, each node which has a copy at the start of the phase sends to another node which doesn't. In this way, the number of nodes that have copies of the message doubles after each phase. The algorithm stops once all nodes have received the broadcast message. In a binomial tree then, the number of nodes the message can reach in a given number of phases, grows as a power of 2 (hence the prefix "bi") with the number of phases.

An f-nomial tree generalizes this algorithm by having each node with a copy of the message at the start of a phase send to (f-1) others who don't, as opposed to just one. Thus, the number of nodes the message can reach grows as a power of f with the number of phases. This is the structure of the algorithm we implemented; only remember the tree collapses rather than expands.

Figure 6 shows some example f-nomial trees of varying degrees which cover 16 nodes. In general, the lower the degree, the taller the tree. Each level of the tree corresponds to a communication phase, while the width of each level determines the amount of computation any one processor is required to do. Efficient algorithms will tend to balance the cost of communication and computation. Communicationally bound reductions will favor wide trees to minimize the number of tree levels, and hence, the number of communication phases. Computationally bound reductions, on the other hand, will fair better with tall trees which better parallelize the processing. Thus, the best choice for the degree of the tree depends on the relative costs established by a particular problem.

Now we apply the model to this algorithm. Since the

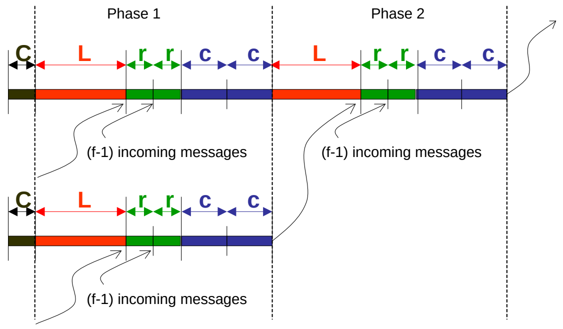

Figure 7: Multi-phase Reduction Latency Model

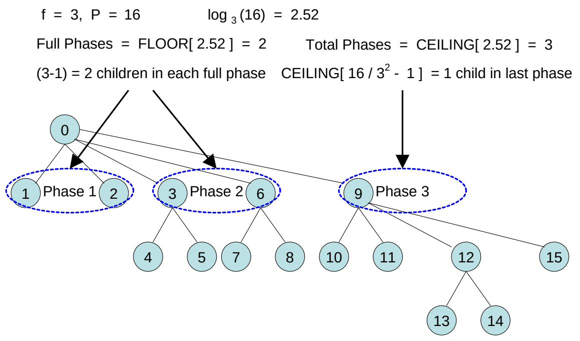
f = 3, P = 16
log 3 (16) = 2.52
Full Phases = FLOOR[2.52] = 2
Total Phases = CEILING[2.52] = 3
(3-1) = 2 children in each full phase
CEILING[16 / 3² - 1] = 1 child in last phase
Phase 1
Phase 2
Phase 3

Figure 8: Derivation of Model of F-nomial Reduction Latency

root node in an f-nomial tree is involved in each step of the algorithm and is the only node to receive messages in the final step, we can predict the latency of the entire operation by focusing on the work the root node must do. An f-nomial tree contains roughly $log_f(P)$ phases, during each of which the root has roughly (f-1) children (roughly, since this assumes a full tree). Each phase will be of the linear, building-block, form of the serial reduction data previously discussed. Thus one can arrive at the following expression as a quick analysis of the time required for an f-nomial algorithm to complete:

$$Time \approx C + [L + (f-1) \cdot (r+c)] \cdot log_f(P)$$

Application of the model for an intermediate phase is shown pictorially in Figure 7. In this algorithm, the initial overhead, C, is encountered as a one time cost. Then there are $log_f(P)$ phases each of which consists of (f-1) children who send to the root at the same time. All of these messages worm their way in parallel to root and simultaneously suffer the latency, L, before arriving. Finally, the root must receive and reduce each of the (f-1) messages before moving to the next phase.

This simplistic expression does not accurately account for trees with a number of nodes other than an integer power of the degree f. When the number of nodes, P, is not an integer power of the degree, f, more careful analysis will show that:

$$\begin{aligned} Time \approx\ & C + L \cdot \lceil PHASES \rceil + \\ & (r+c) \cdot \{(f-1) \cdot \lfloor PHASES \rfloor + \\ & \lceil P/f^{\lfloor PHASES \rfloor} - 1 \rceil\} \\ & \text{where } PHASES = log_f(P). \end{aligned}$$

Here, $PHASES$ represents roughly the number of phases in the f-nomial tree. In particular $\lceil PHASES \rceil$ is the total number of phases, while $\lfloor PHASES \rfloor$ is the total number of full phases, i.e. those involving a full set of (f-1) children. The L term represents the total latency cost incurred from each phase of the tree. The (r+c) term accounts for the time to pull each message from the network and perform the reduction, which in turn is broken into two terms itself. The $\lfloor PHASES \rfloor$ term counts the number of children we process due to full phases, while the ceiling term counts the number of children in the last step, which may be fewer than a full set. An example is given in Figure 8 for a 16-node tree to demonstrate how the various terms refer to the tree. This more detailed model was found to be impressively accurate. Verification of this model is presented in the experimental section.

5.4 Vector Split Optimization

The slower and less functional NIC CPU is quite sensitive to the vector size of the reduction, especially for floating-point operations which must be emulated in software. To reduce this cost, one would like to heavily parallelize the computation. In other words, we would often like to keep as many of the NIC processors working as possible. To do so, we are often willing to suffer a little extra communication cost in favor of a substantial reduction in computation cost.

For multi-element vectors we can use an optimization to increase parallelism, proposed by Van de Geijn in [12]. Basically, the idea is to split the vector and assign the different pieces to different groups of nodes. The groups then reduce the distributed pieces in parallel and recombine the vector in the last step. As an example, say we would like to reduce a two-element vector over 8 nodes. Presented with this optimiza-

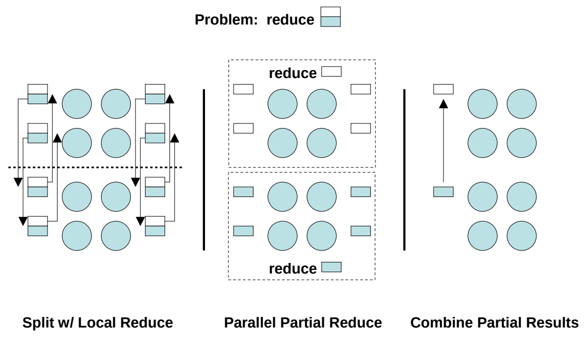

Figure 9: Vector Split Optimization

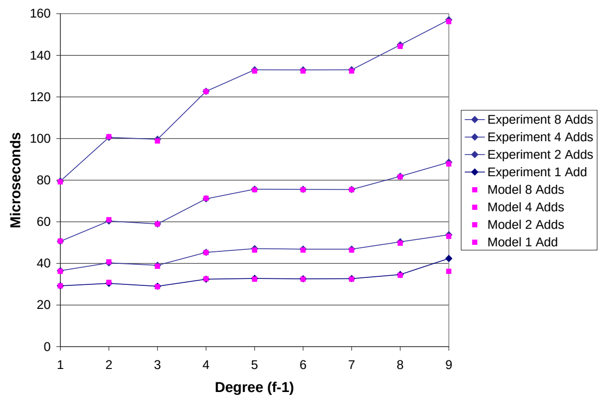

Figure 10: Prediction and Actual Latencies for Float-64 Addition on 31 Node F-nomial Tree

tion, we now have two options: 1) perform a straightforward two-element reduction via an 8-node f-nomial tree or, as shown in Figure 9, 2) divide the 8 nodes into two groups of 4, assign the top piece of the vector to one group and the bottom piece to the other, then perform two single-element reductions via 4-node f-nomial trees in parallel, and finally recombine the reduced vector pieces at the end. In the second approach, we suffer from one extra communication step to recombine the vector pieces at the end, however if computation is expensive, we save significantly on reduction costs during each phase of the reduction tree. For very large trees, which require many phases, this savings can quickly amount to a lot.

This optimization was pre-pended to the f-nomial algorithm to create a new algorithm we call “f-nomial split”. During the beginning, the vector is split in halves continuously until the pieces consist of just single elements. The f-nomial tree algorithm is then used to reduce the single-element vectors. As discussed, this is done in parallel over multiple sub-trees. The root of each of these sub-trees will receive a fully reduced piece of the vector, which is then sent to the primary root of the overall reduction tree in the last step. The improvement due to this optimization proved to be dramatic, and is discussed in the experiment section. Basically, it allows the NIC-based reductions to scale substantially better than they otherwise would have for larger vector sizes.

6 Experiments

In this paper, we aim to highlight the attractive advantages NIC-based reductions achieve over host-based versions in large-scale systems. We developed our algorithms and our initial performance evaluation on the ‘crescendo’ cluster at LANL, a 32-node 64-processor cluster based on 1 GHz Pentium IIIs and the Quadrics network. We completed our scalability analysis on the ALC cluster [24] located at the Lawrence Livermore National Laboratory. ALC uses 960 dual-processor nodes with 2.4 GHz Xeons and the Quadrics network.

To begin, we will first verify the accuracy of the newly proposed model. Then, we show results indicative of the reduced latency and increased consistency we observed using f-nomial NIC-based reductions. To end, we present the benefits obtained with the vector split optimization.

6.1 Model Verification

Before running tests on large-scale systems, we wanted to inspect the accuracy of the model. We extracted the model parameters from the serial reduction data as previously mentioned and applied them to various f-nomial trees for different reduction problems. To provide some confidence in this model, in Figure 10, we show the predicted and measured latencies for 64-bit floating-point addition on a 31-node system using vectors sizes of 1, 2, 4, and 8 elements. There are a few items of interest here.

First, as one might guess, we were of course quite pleased to see how well the model aligns with actual measurements. Because the model fits the data so closely, this allows one to make theoretical estimates of the behavior of various reduction algorithms with a good deal of confidence. Thus, in future reduction algorithm design, one has a detailed model by which one may be able to consider and eliminate many design choices without the need to run extensive tests.

Second, it is also quite important to note how significant the computational costs are. For example, note that the latency required to reduce an 8-element vector across 31 nodes is more than three times the latency required for a single-element vector. Clearly then, any well-designed algorithms must absolutely consider computation costs. To make the point once more, this brightly highlights the issue of the difference in processor speeds discussed earlier. This issue is the most limiting impedance which NIC-based reduction implementations encounter.

Finally, note that because of the high susceptibility to computation costs, the degree of the f-nomial tree may make a significant difference in the latency of the reduction. Intuition suggests that expensive computation should be spread among as many processors as possible, implying that efficient algorithms will tend to produce low-degree trees for problems that require much computation. Reassuringly, that is what is observed in the plots. Small vectors, which require less processing time, lead to curves that are essentially flat for the degrees tested, while larger vectors tend to heavily favor lower-degree trees. On the other hand, for reduction operations simpler than floating point addition, it pays more dividends to use higher-degree trees to save on the relatively more costly communication. Once again, we point out here that, because the host processor is so much faster, such drastic latency variation would not be observed as the degree of the tree is varied in host-based reductions. This is why many previous reduction algorithms so successfully get away with neglecting computation costs.

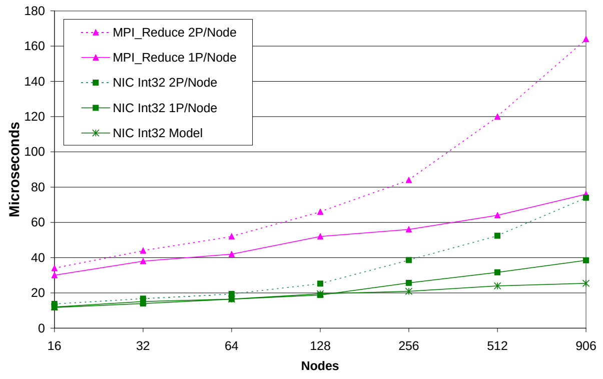

Figure 11: Host-based and NIC-based Reduce Latencies for INT32

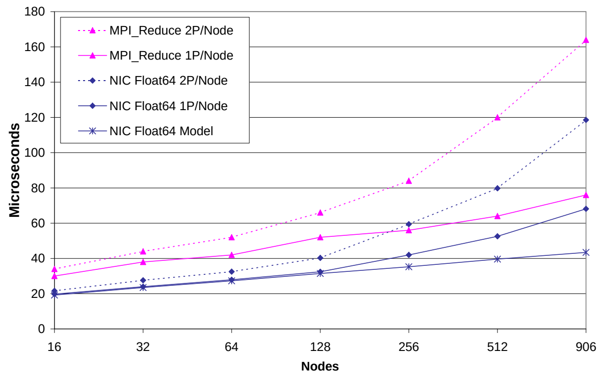

Figure 12: Host-based and NIC-based Reduce Latencies for FLOAT64

6.2 Reduced Latency

We timed the latencies for host-based and NIC-based reduction over a variety of operations and data sizes. We used the MPI reduce collective for our host-based tests. When taking measurements, we found a large variance in the host-based latencies from one iteration to another. To compensate, we plotted the average latency recorded over 100,000 iterations. We show the single-element vector results we obtained for host-based and NIC-based 32-bit integer addition in Figure 11 and 64-bit floating-point addition in Figure 12. Since the host-based latencies are only slightly affected by the type of operation being computed, the provided curve for floating-point addition is representative of other operations as well.

As the figures show, we note that the NIC-based curves scale considerably better than the host-based results. Indeed, as one may infer from the 32-bit integer addition curve, our NIC-based implementation was able to perform simple integer reductions in about half the time it takes the host to do so. Further, even while incurring the expensive cost of emulating floating-point addition on a much slower processor, our NIC-based implementation was able to substantially out race the host-based reduction. With some optimization to the emulation software, this gain could be even further improved. We will acknoledge that, when reducing with floating-point operations, the very best host-based latency recorded was better than the best NIC-based times. However, for simpler operations, like integer addition, the NIC-based implementations gained enough in PCI transaction savings to out perform the host even on its best run. At any rate, it is clear to see that, in many cases, NIC-based reductions can complete with extremely low latencies. When reducing over 906 nodes, we were able to obtain latencies as low as 40 μs for integer operations and a slightly higher time of 65 μs for floating-point.

Further, we note that the NIC-based reduction latencies involving one process per node scale dot-for-dot with the times predicted by the model up through 128 nodes, at which point the measured times break away cleanly. This deviation is due to the testing environment in which we recorded our results; it is not due to an inherent fault in the model. The synchronization method we used in between reduction iterations changed from a hardware-based barrier to a software-based barrier at this point due to the manner in which nodes were allocated to us. The model could be adjusted to account for this difference, however, we thought it to be quite instructive to observe the kind of performance one could expect to see if allocated nodes appropriately. The model suggests that, provided with hardware-based barriers, extremely low-latency reductions may be achieved. Namely, we have all indications that 32-bit integer addition can be completed in under 25 μs and 64-bit floating-point addition in less than 45 μs, even for clustes as large as 900 nodes.

Finally, we should point out the latency deviation in the NIC-based results when two processes are invovled on each node. Unfortunately, the curves follow the same trend noticeable in the host-based results which we were trying to avoid. Again, we blame this occurance on the synchronization method used. We rather naively used a host-based barrier in between our NIC-based reductions during testing. As a result, our NIC-based timings were subject to the same type of host-level problems as the host-based implementations. We intend to fix these problems by implementing our own NIC-based synchronization scheme for future tests, however, we did not get the opportunity for another test slot on the Livermore machine before writing the paper.

6.3 Increased Consistency

We just mentioned how the host-based MPI reduce latencies varied substantially depending on the system environment. The best times we observed, when the system was unloaded and noise-free, were about 3 times better than the times observed when other jobs were running on the system. The NIC-based results were quite steady in either case. This is related to the consistency advantage we have noted for NIC-based reductions.

To brighten the point a little more, Figure 13 shows a distribution graph of the latencies recorded for the NIC-based and host-based 64-bit floating-point addition of a single-element vector. Though at first glance, the NIC-based reduction appears to take more time than the host-based reduction, one must look deeper into the numbers. The point to be made is the much tighter variance which surrounds the sharp spike of NIC-based latencies. Host-based latencies, on the other hand, are spread quite smoothly across a wide range of values. Indeed, a very large number of host-based latencies extend far past the right-hand limit of the distribution graph. To be precise, 97% of the NIC-based reductions fall with a spread of only 4 μs, while for host-based reductions, only 57% fall within

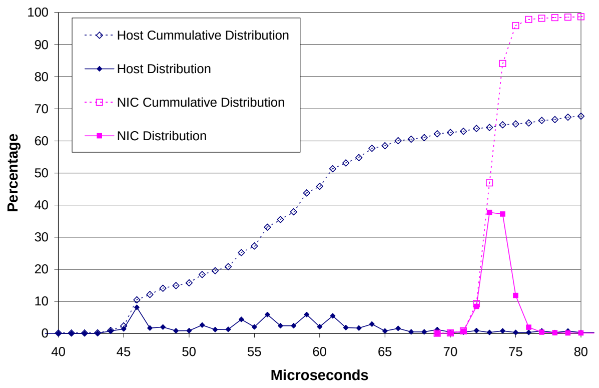

Figure 13: Reduction Latency Distributions for Single-Element Float-64 Addition on 900 Nodes

a spread of 20 μs. In fact, after adding it all up, the average host-based latency comes in around 95 μs, while the NIC scores a substantially lower 75. This noticeably large contrast in consistency is quite indicative of the non-deterministic effect that process swapping imposes on host-based reduction implementations. As expected, NIC-based reductions are more consistent and scalable than host-based versions on large-scale systems.

6.4 Split Optimization

Earlier we noted that, while NIC-based reductions can provide reduced latency and increased consistency, they are especially sensitive to computational cost due to the slow NIC CPU. The vector split optimization is a way to counteract this shortcoming by increasing parallelism when reducing multi-element vectors.

We measured the performance of the f-nomial split algorithm for 64-bit floating-point addition on 512 nodes using various vector sizes. The results are shown in Figure 14. The value of the vector split optimization is quite pronounced. After 3 splits, the 8-element latency is improved by nearly a factor of 3, while for 4 splits, the 16-element case is over 3 times faster. The trend obviously suggests the larger the vector, the better the benefit.

Although the vector split optimization enables NIC-based reductions to scale better than they otherwise would have, there is still a limit on the performance it can achieve. Note that a latency of 140 μs for a 16-element reduction may still be much more than what a host processor can churn out. And interestingly, one may carefully note that the latency for a 2-element vector actually increases slightly after one split. This of course will happen if the total savings in computation over the height of the tree is less than the added communication cost of the recombine step. However, the

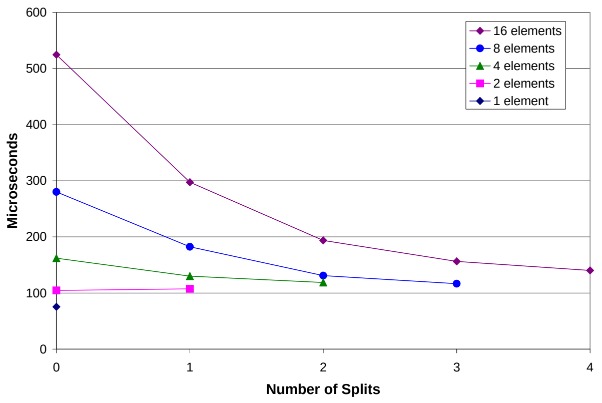

Figure 14: F-nomial Split on Various Vector Sizes for Float-64 Addition on 512 Nodes

cross-over point can be computed so as to always pick the better of the two options. Van de Geijn discusses the details in [12].

7 Conclusions and Future Work

In this paper we showed that NIC-based collectives out perform host-based versions in two important ways: 1) reduced latency via fewer PCI transactions and 2) reduced and more consistent latency via decreased susceptibility to process swapping. While these are attractive advantages, they don't come to us for free, namely, one must deal with host-NIC synchronization overhead and perform processing on a much slower and less functional processor.

Many existing reduction algorithms are based on the popular and successful LogP and postal models. Unfortunately, these models do not well account for the asymmetrical communication characteristics in Quadrics, nor for the high computation costs of the relatively slow Elan processor. In response, we presented a simple model which does address these issues. The new model suggests that efficient reduction implementations will fall in the class of asymmetrical tree-shaped algorithms. We then presented the f-nomial tree reduction algorithm, which are generalized binomial trees. We also added the vector split optimization to increase performance when reducing larger vectors.

Experimental data shows that the model we proposed quite accurately predicts the performance of our algorithm. We also found evidence that NIC-based reduction implementations indeed scale better by eliminating many PCI transactions. More dramatically, we show that NIC-based reductions can avoid the costly process-swapping penalties to which the host-based versions are subject to. Finally, we note the value of the vector split optimization for larger reduction sizes in NIC-based algorithms.

The experimental results show low latency and impressive scalability. In the largest configuration tested —1812 processors— our NIC-based algorithm sums single-element vectors of 32-bit integers and 64-bit floating-point numbers in 73 μs and 118 μs, respectively. These results represent respective improvements of 121% and 39% over the production-level MPI library.

Future work will involve exploration of additional algorithms. It is also possible to optimize much of the software performing the reduction on the NIC, especially for floating-point operations. Another important optimization to be tapped is the ability for NIC threads to directly build and send packets on the network. For small messages, one can gain about 33% improvement in message latency by doing so. In addition, the host-based versions do not have access to such benefit, so this would increase the gains obtained by NIC-based implementations during each phase, on top of savings already gained by the elimination of extraneous PCI-bus transactions.

References

[1] A. Bar-Noy and S. Kipnis. Designing broadcasting algorithms in the postal model for message-passing systems. *Math. Systems Theory*, 27(5), 1994.

[2] A. Bar-Noy and S. Kipnis and B. Schieber. Optimal computation of census functions in the postal model. *Discrete Applied Mathematics*, 58, 1995.

[3] M. Barnett, S. Gupta, et al. Building a High-Performance Collective Communication Library. In *Supercomputing*, 1994.

[4] M. Barnett, R. Littlefield, D. G. Payne, and R. V. de Geijn. Global Combine on Mesh Architectures with Wormhole Routing. In *Proceedings of the International Parallel Processing Sympos ium*, pages 156–162, 1993.

[5] J. Bruck, R. Cypher, P. Elustando, A. Ho, C. Ho, V. Bala, S. Kipnis, and M. Snir. CCL: A Portable and Tunable Collective Communication Library for Scalable Parallel Computers. In *Proceedings of the International Parallel Processing Sympos ium*, 1994.

[6] D. Buntinas and D. Panda. Fast nic-based barrier over myrinet/gm. In *Proceedings of the Interna-*

tional Parallel and Distributed Processing Symposium 2001 (IPDPS'01), San Francisco, CA, April 2001.

[7] D. Buntinas and D. Panda. NIC-Based Reduction in Myrinet Clusters: Is It Beneficial? In *Proceedings of Workshop on Novel Uses of System Area Networks*, Febuary 2003.

[8] D. Buntinas, D. Panda, J. Duato, and P. Sadayappan. Broadcast/multicast over Myrinet using NIC-assisted multidestination messages. In *Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing (CANPC '00), High Performance Computer Architecture (HPCA-6) Conference*, Toulouse, France, January 2000. Available from `ftp://ftp.cis.ohio-state.edu/pub/communication/papers/canpc00-nic-multicast.pdf`.

[9] D. Buntinas, D. Panda, and W. Gropp. NIC-based atomic operations on Myrinet/GM. In *SAN-1 Workshop, High Performance Computer Architecture (HPCA-8) Conference*, Boston, MA, February 2002. Available from `ftp://ftp.cis.ohio-state.edu/pub/communication/papers/san-1-atomic_operations.pdf`.

[10] M. Collette. LLNL User Briefings. In *ASCI Q LANL/HP Technical Quarterly Meeting*, Santa Fe, NM, March 2003.

[11] D. E. 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 Symposium on Principles and Practice of Parallel Programming*, pages 1–12, San Diego, CA, May 1993.

[12] R. V. de Geijn. Efficient Global Combine Operations. In *Proceedings of 6th Distributed Memory Computing Conference*, April 1991.

[13] C. Huang and P. K. McKinley. Design and implementation of global reduction operations across ATM networks. In *Proceedings of Third International Symposium of High-Performance Distributed Computing*, San Francisco, CA, August 1994.

[14] C. Huang and P. K. McKinley. Efficient collective operations with atm networks interface support. In *Proceedings of International Conference on Parallel Processing (ICPP'96)*, 1996.

[15] J. Bruck and L. De Coster and N. Dewulf and C. T. Ho and R. Lauwereins. On the Design and Implementation of Broadcast and Global Combine Operations Using the Postal Model. *IEEE Transactions on Parallel and Distributed systems*, 7(3), March 1996.

[16] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings. Predictive performance and scalability modeling of a large-scale application. In *Proceedings of SC2001*, Denver, Colorado, Nov. 10–16, 2001. Available from `http://www.sc2001.org/papers/pap.pap255.pdf`.

[17] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, Oct. 1985.

[18] M. Bernaschi and G. Iannello. Collective Communication Operations: Experimental Results vs. Theory. *Concurrency: Practice and Experience*, 10(5), April 1998.

[19] F. Petrini. Identifying and Eliminating the Performance Variability on ASCI Q. Invited Talk, Lawrence Livermore National Laboratory, Availabe from `http://www.c3.lanl.gov/~fabrizio/talks/asciq_noise.pdf`, March 2003.

[20] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics network: High-performance clustering technology. *IEEE Micro*, 22(1):46–57, Jan./Feb. 2002. ISSN 0272-1732. Available from `http://www.computer.org/micro/mi2002/pdf/m1046.pdf`.

[21] Quadrics Supercomputers World Ltd. *Elan Programming Manual*, 2nd edition, Dec. 1999.

[22] Quadrics Supercomputers World Ltd. *Elan Reference Manual*, 1st edition, Jan. 1999.

[23] Quadrics Supercomputers World Ltd. *Elite Reference Manual*, 1st edition, Nov. 1999.

[24] M. Seager. Planned Machines: ASCI Purple, ALC and M&IC MCR. In *7th Workshop on Distributed Supercomputing (SOS7)*, Durango, CO, March 2003. Available from `http://www.cs.sandia.gov/SOS7/presentations/seager.ppt`.

[25] http://www.jhauser.us/arithmetic/SoftFloat.html.