NIC-based Reduction Algorithms for Large-scale Clusters

Fabrizio Petrini[1], **Adam Moody**[2], **Juan Fernández**[3], **Eitan Frachtenberg**[1] **and Dhabaleswar K. Panda**[4]

[1]Computer and Computational Sciences (CCS) Division,
Los Alamos National Laboratory, NM 87545, USA
[2]Integrated Computing and Communications Department,
Lawrence Livermore National Laboratory, CA, 94550, USA
[3]Computer Engineering Department,
University of Murcia, 30071 Murcia, Spain
[4]Department of Computer & Information Science,
The Ohio State University, Columbus, OH 43210, USA

Abstract: Efficient algorithms for reduction operations across a group of processes are crucial for good performance in many large-scale, parallel scientific applications. While previous algorithms limit processing to the host CPU, we utilize the programmable processors and local memory available on modern cluster network interface cards (NICs) to explore a new dimension in the design of reduction algorithms. In this paper, we present the benefits and challenges, design issues and solutions, analytical models, and experimental evaluations of a family of NIC-based reduction algorithms. Performance and scalability evaluations were conducted on the ASCI Linux Cluster (ALC), a 960-node, 1920-processor machine at Lawrence Livermore National Laboratory, which uses the Quadrics QsNet interconnect. We find NIC-based reductions on modern interconnects to be more efficient than host-based implementations in both scalability and consistency. In particular, at large-scale—1812 processes—NIC-based reductions of small integer and floating-point arrays provided respective speedups of 121% and 39% over the host-based, production-level MPI implementation.

Keywords: cluster computing; reduce; allreduce; Quadrics QsNet; NIC-based operations; collective communication.

Reference to this paper should be made as follows: Petrini, F., Moody, A., Fernández, J., Frachtenberg, E. and Panda, D. K. (2005) 'NIC-based Reduction Algorithms for Large-scale Clusters', *International Journal of High-Performance Computing and Networking*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Fabrizio Petrini is a member of the technical staff of the CCS3 group of the Los Alamos National Laboratory (LANL). He received his PhD in Computer Science from the University of Pisa in 1997. **Adam Moody** is a member of the Integrated Computing and Communications Department at Lawrence Livermore National Laboratory. He received his BS and MS degrees from The Ohio State University in 2001 and 2003, respectively. **Juan Fernández** is an assistant professor of the Computer Engineering Department at the University of Murcia. He received his BS and MS degrees from the University of Murcia in 1995 and 1997, respectively. **Eitan Frachtenberg** is a Director's postdoctoral fellow at LANL. He received his BS, MS, and PhD degrees from the Hebrew University of Jerusalem in 1993, 2001, and 2003, respectively. **Dhabaleswar Panda** is Professor of Computer Science and Engineering and leads the Network-Based Computing Research Group at the Ohio State University. He received a PhD in Computer Engineering from the University of Southern California in 1991.

1 Introduction

Reduction collectives are essential components of many high-performance computing (HPC) applications. Recent performance evaluation studies show that large-scale scientific simulations spend up to 60% of their time executing reductions [22]. In-depth analysis of the scientific workload at Lawrence Livermore National Laboratory shows similar results [12]. Consequently, faster reduction algorithms can substantially shorten the run times of many scientific applications.

Development of efficient reduction algorithms has proven to be a rich area of research. Reduction collectives entail both communication (data transfer) and processing (data reduction operations), and therefore efficient implementations must consider the characteristics of the network, the processor, and the interactions between them. Over the years, many researchers have dedicated significant effort to derive optimal and scalable algorithms [1, 2, 3, 4, 5, 8]. However, with respect to the underlying system characteristics, all of this work commonly assumed reduction processing must be performed by the host CPU.

Network interface cards (NICs) for modern cluster interconnects, such as the Elan3 used in Quadrics QsNet [20], provide programmable processors and ample memory. This added capability allows more functionality to be delegated to the NIC processor. The terms *host-based* and *NIC-based* are used to indicate where functionality is implemented. The focus of this paper is on the implementation of NIC-based reduction. That is, we examine the process of delegating both the communication and the data-processing tasks of reduction collectives to the network interface card.

This article makes the following contributions. First we discuss the benefits of NIC-based reduction, and describe the design issues and solutions we developed. We then present a detailed model to analyze and predict the performance of reduction algorithms on the Quadrics network. Finally, we present experimental evaluations to validate our analytical model and examine the scalability of our algorithms.

From this work we show that NIC-based reduction exhibits better scalability and improved consistency over host-based algorithms. This is especially true for classes of reductions that are frequently used in large-scale, parallel scientific applications. For example, for summation of single-element vectors of 32-bit integers and 64-bit floating-point values over 1812 processors of the ASCI Linux Cluster (ALC) [26], NIC-based reduction was, respectively, 121% and 39% faster than with the production-level, host-based MPI library. Moreover, the standard deviations in timings for the NIC-based case were as much as two orders of magnitude smaller than those for the host-based case.

2 Related Work

Huang and McKinley were possibly the first to realize the potential of NIC-based collectives [16]. They examined the use of implementing broadcast and barrier operations on Asynchronous Transfer Mode (ATM) network adapters to avoid the excessive processing overhead incurred in the software protocol stack. To maintain portability to even the most limited ATM devices, Huang and McKinley placed rigid restrictions on the processing and memory requirements of their algorithms. Specifically, their algorithms were table-driven and performed a small number of arithmetic and logical operations on a few scalar variables. Even with such limitations, these NIC-based collectives scaled significantly better than host-based versions.

Modern cluster interconnects removed the software bottleneck from the protocol stack by using zero-copy, user-level protocols. However, the same interconnects have also dramatically reduced wire and switch latencies in the network, so that now just the cost of transferring data between the host CPU and the network interface contributes significant overhead. At the same time the processing capability and memory available on network interface cards have increased. Consequently, NIC-based collectives are still valuable and their implementation is now more practical, which leads researchers to investigate ever more complex NIC-based operations and algorithms.

Several recent studies have considered NIC-based multicast algorithms [6, 11, 14, 18, 28, 30]. Multicast is a complex operation which must accommodate varying message sizes and destination sets, and address flow control, acknowledgment collection, and reliability. These studies take different approaches to meeting these requirements, still the authors generally conclude that modern NICs are capable of executing multicast more efficiently than host-based implementations.

The work most closely aligned with our own is that of Buntinas and Panda [10]. They investigated the potential of NIC-based reduction on clusters connected with Myrinet [7]. In particular, they modified the network drivers to implement binary AND and OR operations, and integer and floating-point addition of a single 64-bit value via binomial trees. For these cases they found that NIC-based reduction has better scalability than host-based reduction, and yields performance gains in clusters with as few as 8 nodes. While their implementation provides some additional flexibility, they leave the investigation of more complicated reductions as future work.

This paper picks up where their work left off. We investigate the use of different communication patterns and a range of data sizes for an expanded set of reduction operations, as well as an optimization for multi-element data vectors. In addition, we propose an accurate parameterized model which can be used to analyze and select the best implementation for a given instance of a reduction. Finally, we test our implementations at a dramatically increased scale, running on a machine using as many as 906 nodes.

3 Motivation and Background

NIC-based collectives have both advantages and disadvantages over traditional host-based approaches. This section first discusses the relevant benefits and challenges of NIC-based approaches. We then detail our particular design goals and development environment.

3.1 Benefits of NIC-based Reduction

On modern interconnects, NIC-based collectives are significantly faster than host-based versions. Efficient collective implementations typically require a set of nodes to exchange a series of related messages. In host-based implementations, each message in the series must be passed between the host processor and the network interface via IO-bus transactions. NIC-based implementations, on the other hand, handle messages immediately at the NIC, eliminating transfers through the IO bus. Since IO-bus transfers constitute a significant fraction of the overhead in modern cluster interconnects, and because collectives involving many processes entail many messages, NIC-based collectives scale substantially better than host-based versions as the size of the cluster increases. To date, most NIC-based research has focused on this advantage [6, 9, 10, 11, 14, 16, 18, 28, 30].

Another advantage of NIC-based collective operations, which has thus far been overlooked, is that they perform more consistently than host-based implementations. The host CPU is typically required to multitask processes other than the application's, such as operating system daemons and resource management threads. Unfortunately, to service another process, the operating system may deschedule application processes at critical times. Descheduling a process involved in a collective delays the completion of the operation. This interference is stochastic, and the chances for such delays worsen with increases in both the frequency of collectives and the number of processes involved. This effect is particularly dramatic in large-scale clusters and has been shown to cause a slowdown of 50% or more in tightly-synchronized applications [22]. The NIC, on the other hand, is essentially dedicated to the application and so avoids most of the interference associated with multitasking. Thus NIC-based collectives are able to execute with more consistent times than host-based collectives.

3.2 Challenges of NIC-based Reduction

Even though the NIC carries out the actual collective in NIC-based implementations, the host must communicate to the NIC, among other information, what operation is to be done, which data are to be processed, and when the operation is to start. Also, the NIC must notify the host of the operation's completion and deliver any final results. Such *host-NIC synchronization overhead* diminishes the gains provided by implementing NIC-based collectives. However, this overhead is relatively small and is not of major concern.

A more important issue to consider is that of the NIC processor's capability. The user-programmable processor on the NIC is considerably slower than the host processor (more than 25 times slower on ALC). This difference limits the complexity of the collectives and algorithms that may benefit from NIC-based implementations. To complicate matters further, the NIC processor typically lacks functionality present in the host processor. For example, there is no hardware-based floating-point support on the Quadrics Elan3. The limitations of the NIC processor proved to be the most challenging issue encountered in our work.

3.3 Targeted Design Goals

Given the NIC processor limitations, much of the research in NIC-based work so far has concentrated on collectives which involve little processing. Collectives such as barriers, broadcasts, and multicasts simply require intermediate nodes to pass on the received message as is, possibly with minor data restructuring. Because so little processing is required, these algorithms incur little penalty from running on slower processors, and the overall results have been quite successful. This success inspired us to investigate more complicated cases, namely reductions.

Our design goal was 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 minimum, maximum, and summation operations. We seek to improve the *reduction latency*, by which we mean the time from when the first process enters the operation to when the final result is delivered to its final destination.

3.4 Targeted Development Environment

We implemented NIC-based reduction on the Quadrics QsNet network, a modern cluster interconnect technology [20]. QsNet is based on two building blocks: a programmable network interface card called the Elan3 [23, 24] and a low-latency high-bandwidth communication switch called the Elite [25].

The Elan3 resides on the PCI bus and provides an interface between the network and a processing node that contains one or more CPUs. The Elan3 provides a user-programmable, multi-threaded, 32-bit, 100 MHz RISC-based processor; 64MB of local SDRAM, an MMU, and other sophisticated processing features. The purpose of all this hardware is to enable the implementation of higher-level message processing protocols without requiring explicit intervention from the host CPU.

The Elan3 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 underlying network is circuit-switched and uses source-based wormhole routing. It consists of Elite switches connected in a fat-tree topology [19]. An Elite provides eight bidirectional links, each with a raw bandwidth of 400 MB/s (325 MB/s at the MPI level) and a full crossbar switch with a low 35ns cut-through latency.

The Elite switch also provides hardware support for collective communication, including barriers and broadcasts, which is remarkably fast and scalable [21]. In fact, the cost to broadcast a message to all nodes is comparable to the cost of sending it to just one. This broadcast hardware support makes the implementation of the allreduce algorithms trivial—an efficient reduce, followed by a broadcast from the root, provides an efficient allreduce. For this reason we will focus our attention on just the reduce phase in the rest of the paper.

4 Design Issues and Solutions

While host-NIC synchronization overhead poses some concern, the primary challenges faced when developing NIC-based reduction are the limitations of the NIC processor. In this section, we describe the issues we encountered along with the solutions we developed to overcome them.

4.1 Host-NIC Synchronization Overhead

The host must perform several tasks to delegate a reduction operation to the NIC. This includes writing the application data to and reading the final result from NIC memory; informing the NIC processor of what operation to perform, what data type to use, and the number of vector elements to process; and providing the NIC with lists of communication partners and intermediate data buffers. The host must also instruct the NIC when to start the operation, and the NIC must notify the host of the operation's completion. Such host-NIC synchronization introduces overhead, but its cost can be minimized and/or hidden.

By eliminating redundant information, we minimized host-NIC synchronization overhead by reducing the amount of physical data written to the NIC. Where possible, we referred to collections of data items using a single index parameter. For example, we grouped intermediate data buffers into *channels*, which enabled the host processor to refer to a set of buffers through a single channel number. A similar technique was used with the data structures that list the communication partners. Typically, applications repeatedly iterate over a limited set of communication patterns. Thus we used some portion of NIC memory as a cache so the host processor could refer to communication data structures previously copied to the NIC with a simple cache-line number.

Additionally, many data items assume limited ranges, so only a few bits are needed to encode their value. For example, 8 or 16 different channels and 16 or 32 cache slots will often be more than enough. We packed the channel and cache-line numbers along with the type of collective, e.g. reduce or allreduce, the reduction operation, e.g. 32-bit integer addition or 64-bit floating-point maximum, and the vector size into a single 32-bit value using bit masks.

We also worked to hide the host-NIC synchronization overhead. The benefits of NIC-based reduction are gained when a node must receive, process, and then send a message. In the case of a reduction tree, this corresponds to just the intermediate nodes. At the leaves, which only send data, and the root, which only receives and processes data, NIC-based reduction provides no benefit, and the associated host-NIC synchronization only gets in the way. By using NIC-based reduction only during intermediate steps, host-NIC synchronization costs can be largely removed from the critical path. We found that this technique can cut host-NIC synchronization overhead by more than half.

4.2 NIC Processor Functionality

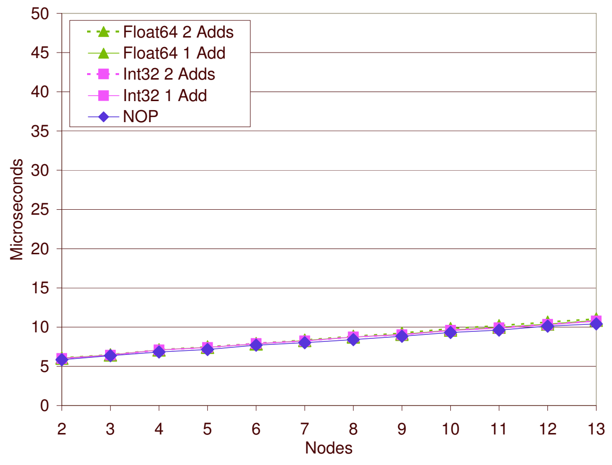
The first major obstacle to satisfying our design goals was the lack of hardware support for floating-point operations on the Quadrics Elan3 processor. The NIC processor offers only integer instructions so floating-point operations must be emulated in software. Efficient emulation of floating-point operations meeting all of the various representations, rounding methods, and exceptions standardized in the IEEE 754 floating-point specification is not trivial. To solve this problem, we ported the SoftFloat [15] library to the Elan3. SoftFloat implements IEEE-754-compliant floating-point operations via integer and bit-wise logic instructions. It is designed to be efficient, and it is open source and freely available.

4.3 NIC Processor Speed

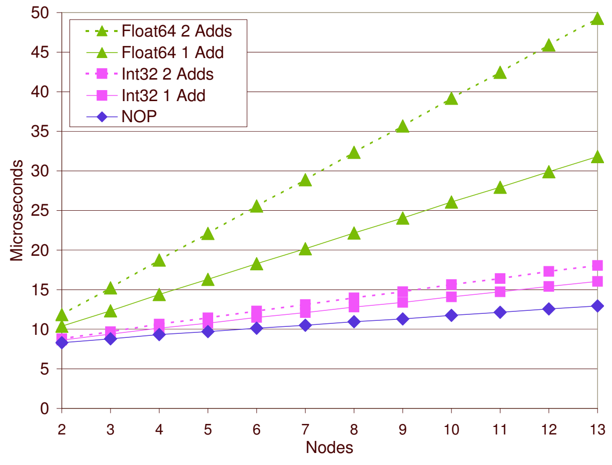
The biggest obstacle faced in designing NIC-based reductions is the speed of the NIC processor. Direct comparison of the clock rate of the Elan3 processor, at one-hundred megahertz, to that of a typical host processor in the multi-gigahertz range shows an order of magnitude difference in processor speeds. The gap is wider when executing floating-point operations, which must be emulated in software.

To gain intuition on the communication and computation characteristics of the Elan3 and host processors, we first implemented a simplistic reduce algorithm referred to as *serial reduction*. In this algorithm the root of the reduction is solely responsible for receiving and reducing all of the data. In a group of P nodes, the $(P-1)$ non-root nodes simultaneously send their data to a corresponding RDMA buffer at the root. The root waits until it has received all of the messages and then reduces the data in serial order. The reduction is completed when the root signals the group with a hardware-based broadcast.

Serial reduction tests involving 2-13 nodes for various reduction operations and data sizes produced Figure 1. Figure 1(a) shows the host-based serial reduction latencies, while Figure 1(b) shows the NIC-based times. Each figure provides the latencies of 32-bit integer addition and 64-bit floating-point addition for one-



(a) Host-based



(b) NIC-based

Figure 1: Serial reduction latencies

and two-element vectors. Also shown is a NOP operation which performs no computation. Since the latency for a NOP serial reduction consists only of communication costs, it represents a lower bound for the reduce algorithm with respect to computation.

In comparing these two figures, it is immediately obvious that NIC-based reductions depend significantly on the reduction operation as well as the reduction vector size, while host-based reductions are largely independent of both. In the NIC-based graph, simple operations scale considerably better than more complicated ones: compare integer addition to floating-point addition. Also, even fast operations are sensitive to small changes in data size: observe integer addition for one- and two-element vectors. Each curve in the host-based graph, on the other hand, lies on or just above the NOP curve, implying that computation is insignificant compared to communication.

That the NIC processor is slower than the host processor was already understood, but it is now clear that this difference is substantial, since computation costs may be comparable to communication costs. While efficient host-based reductions may be designed considering only communication, designs for NIC-based reduction are more complicated because they must also account for computation.

4.4 Simple Operations and Small Data Sizes

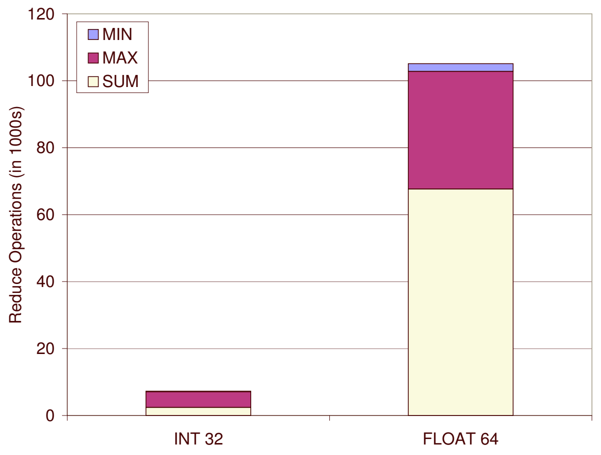
NIC-based reductions will perform well only for simple operations and small data sizes. The slow NIC processor will become overwhelmed if given anything more complicated. This is a tight constraint on the class of reductions where NIC-based implementations may be valuable. However, a large majority of the reductions posed by practical programs fall within this class.

Reductions involving simple operations on small data sizes are the prevalent case in many scientific applications. Researchers have verified this claim across a collection of large-scale scientific programs covering a range of application domains [29]. That collection includes linear systems solvers, simulators for gas dynamics, particle and photon transport, and shock-wave analysis. In further support of this, we profiled the MPI allreduce operations performed during the execution of SAGE [17]. SAGE is representative of scientific applications running on large-scale parallel clusters in the ASC program. The results are shown in Figure 2.

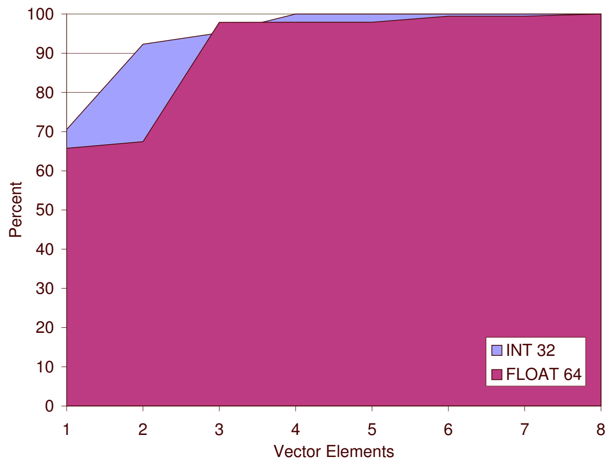
Figure 2(a) shows the distribution of reduction operator types. Note that only two simple data types are used by SAGE: 32-bit integers and 64-bit floating-point numbers. Additionally, only a few simple types of operations are used, namely, minimum, maximum, and summation. Typical reductions thus require limited processing functionality.

Equally informative is Figure 2(b), which shows the cumulative distribution of the data sizes for both integer and floating-point data types. The data reveal that 95% of all reductions use three or fewer elements and all reductions use eight or fewer.

Together, these two figures imply that typical reductions involve simple operations on small vectors. Thus, while the NIC processor provides limited capability, it is the common case reduction which stands to benefit from NIC-based implementations.



(a) Operator type distribution



(b) Vector size distribution

Figure 2: Profile of MPI_Allreduce operations in SAGE

4.5 f-nomial Trees – Generalized Binomial Trees

Communication structures in efficient reduction algorithms tend to balance message processing time with message latency. For simple operations and small data sizes, message processing time in host-based reductions is dependent only on the communication costs, while it is affected by the communication costs, the reduction operation, and the vector size in NIC-based implementations. Thus, while a single communication structure will suffice for efficient host-based reductions, the slow NIC processor demands a range of communication structures for efficient NIC-based reductions. For this work, we chose to implement f-nomial trees (a.k.a. k-nomial trees), since they provide such a range of communication structures by generalizing binomial trees, a well-known reduction communication structure.

Binomial trees are commonly used in reduction algorithms because they have two useful properties, 1) they have a regular structure, so they are easy to implement; and, 2) they keep many nodes involved throughout the collective, so they are well parallelized. In fact, binomial trees are known to be optimal communication structures for reduction in synchronous networks [2], i.e., those in which the sender and receiver incur the same cost for message transfer (latency plus processing). f-nomial trees generalize binomial trees to add a third valuable property: they provide a range of communication structures, so one may selectively balance message processing time against message latency.

While the goal is not to dwell on presentation of a new algorithmic communication structure, f-nomial trees are somewhat uncommon so some discussion is called for. Here we describe f-nomial trees starting from a quick review of the operation of binomial trees, from which the generalization is trivial. Also, although messages in reduction trees collapse to the root node, it is easier to describe the structure of a tree as it expands. For convenience of description assume the goal is to broadcast a message from the root to all nodes in the tree.

The operation of binomial trees can be described as follows. Starting from the root, the broadcast message is distributed through a series of communication phases. During each phase, each node with a copy of the message sends it to another node which does not have a copy, so by the end of each phase, the number of nodes holding a copy of the message is doubled. Thus, in a binomial tree, the number of nodes the message can reach grows as a power of 2 (hence the prefix "bi") with the number of phases.

An f-nomial tree generalizes this algorithm so that, during each phase, each node with a copy of the message sends to $(f-1)$ others who do not, as opposed to just one. For instance, during the first phase, the root sends the broadcast message to $(f-1)$ children. By the end of the first phase, the root and its $(f-1)$ children all hold a copy of the message, for a total of f nodes. In the second phase, each of these f nodes becomes a parent to $(f-1)$ children who have yet to receive the message. By the end of the second phase, the message spreads from the f parent nodes to each of their $(f-1)$ children, reaching a total of $f + f(f-1) = f^2$ nodes. In general, the number of nodes the message can reach grows as a power of f with the number of phases.

Our reduction algorithm is based on this communication structure. However, since we implement reduction rather than broadcast, messages collapse to the root rather than expand away from it.

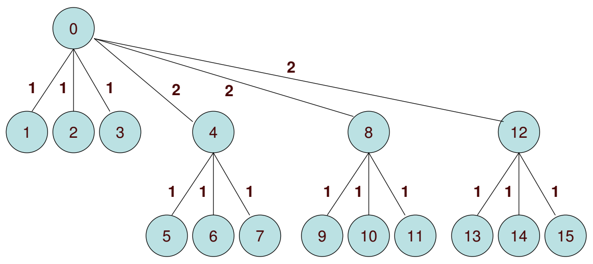


Figure 3: 4-nomial reduction over 16 nodes

As a concrete description of an f-nomial reduction, consider Figure 3, which shows a graph representing a 4-nomial tree covering a set of 16 nodes. In this example, the goal is to reduce data distributed among the 16 nodes and place the result at the root node 0 using a 4-nomial-tree communication structure. The arcs in the graph connect communication partners and are labeled with the phase number in which the corresponding communication takes place; all messages travel upward from children to parents. During the first phase of the 4-nomial algorithm, parent node 0 receives and reduces $(4-1) = 3$ messages from nodes 1, 2, and 3; while likewise, nodes 4, 8, and 12 simultaneously receive and reduce data from their own three children. At the end of the first phase, the distributed data has been partially reduced and localized to the four parent nodes 0, 4, 8, and 12. The algorithm completes after the second phase when node 0 receives and reduces the partial results from the three, now child, nodes 4, 8, and 12. Thus, in two communication phases, the 4-nomial tree is able to perform a reduction over $4^2 = 16$ nodes.

f-nomial trees offer a range of communication structures to select from through choice of the degree of the tree f. For example, Figure 4 shows f-nomial trees of various degrees, all which cover 16 nodes. This flexibility allows one to trade off between communication and computation costs. Each level of the tree corresponds to a communication phase, while the width is related to the amount of computation any one processor is required to do. Communication-bound reductions will favor wide trees in order to minimize the

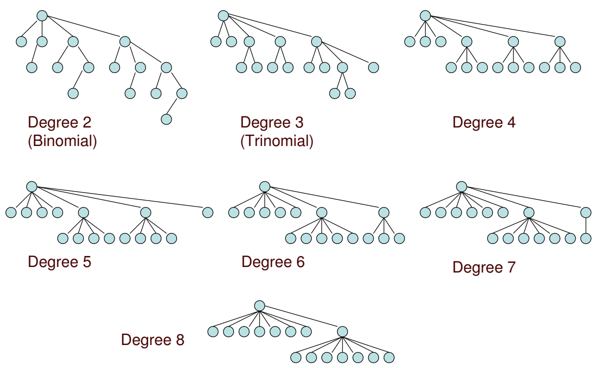


Figure 4: f-nomial trees of varying degrees (16 nodes)

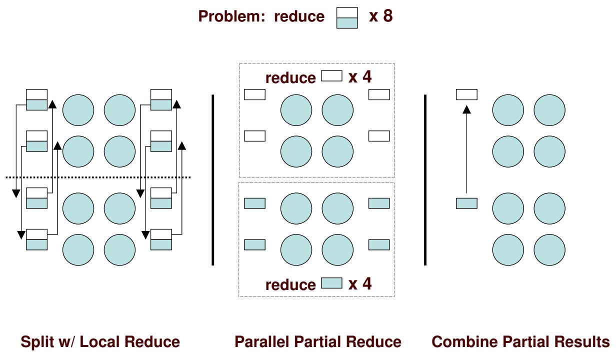


Figure 5: Vector split optimization

number of tree levels, and thus the number of communication phases. Computation-bound reductions will fare better with tall trees which better parallelize the processing. The best choice for the degree of the tree depends on the relative costs established by a particular problem. Section **??** will illustrate how to choose the optimal degree analytically.

4.6 Vector Split Optimization

NIC processors are slow, so it is desirable to keep as many of them working as possible in order to utilize their collective processing power. Often it is worthwhile to do a little extra communication in return for a substantial reduction in computation. In other words, computationally intensive NIC-based reductions should be highly parallelized.

For multi-element vectors, we can increase parallelism through an optimization proposed by Van de Geijn [27]. Basically, the idea is to split the reduction vector and distribute the pieces to distinct groups of nodes. The groups then reduce the pieces in parallel and combine the results to form the fully-reduced vector in the last step. Presented with this optimization, there are two options to reduce multi-element vectors: 1) reduce one large vector serially through a single tree, or 2) reduce smaller pieces of the vector in parallel with many trees. The second approach requires extra communication to distribute and recombine the pieces of the vector. However, if computation is expensive, significant savings are gained by processing the pieces in parallel.

As an example, which is diagrammed in Figure 5, assume we would like to employ this optimization to reduce a two-element vector across eight nodes. The vector elements are shown as small rectangles located adjacent to circles representing the nodes on which they reside. As indicated by the dotted line that bisects the circles horizontally in the left section of the figure, the group of eight nodes is first split into two groups of four. Then the top element of the vector is distributed to the top group of four nodes and the bottom element to the bottom group. To do this, nodes pair up with a partner in the opposite group and send it the appropriate vector element, as represented by the arrows in the diagram. The nodes then reduce the element received from their partner with their local copy of the corresponding element. At this point, the top group contains all information about the top element, and the bottom group contains all information about the bottom element. Once this distribution is complete, the two groups simultaneously perform groupwise reductions on the element assigned to them. This is represented by the dotted boxes shown in the middle section of the figure. Finally, as shown in the right section of the figure, the two fully-reduced elements are recombined to produce the fully reduced, two-element vector.

This optimization was added to the basic f-nomial algorithm to create a new algorithm we call *f-nomial split*. At the beginning the vector is recursively split in half a specified number of times, with the pieces being distributed among the appropriate number of groups. The f-nomial tree algorithm is then used within each of the groups to reduce the smaller pieces in parallel. The root of the f-nomial tree in each group will receive a fully-reduced piece of the vector, which is then sent to the primary root of the overall reduction during the last step. The improvement due to this optimization proved to be dramatic and is discussed in Section 6.2. Essentially, it allows NIC-based reductions to scale substantially better than they otherwise would have for larger vector sizes.

5 Analytical Models

In this section, we apply analytical models to the design of efficient NIC-based reductions. Simple model parameters are introduced and used to describe quantitative differences between host-based and NIC-based reductions. Then the model parameters are applied to f-nomial reductions in order to find the best degree f to use for a given reduction problem.

5.1 The Model Parameters

The sharp linear trend observed in Figure 1 permits accurate modeling of serial reduction latencies using just a slope and intercept. Furthermore, the serial reduction algorithm will serve as the basic building block to more sophisticated tree-based algorithms. Given an accurate model for the building blocks, one can piece together a model for more sophisticated algorithms. Thus, the slope and intercept of the serial reduction latency curves are sufficient to quite accurately predict the performance of any other proposed algorithm.

Continuing in this direction, it is instructive to define the slope and intercept in terms of more meaningful parameters. To account for the linear trend, we recall the implementation of the serial reduction algorithm: all nodes simultaneously send their data to the root, which receives all, and then reduces all messages in serial order. Since the nodes send to the root simultaneously, all messages worm their way through the network to the root in parallel. Hence, regardless of the number of nodes involved, the cost of message latency is suffered only once. On the other hand, the root receives and reduces each message serially, which introduces reception and reduction cost on a per node basis. With these observations, we defined the model parameters as listed in Table 1. Throughout the rest of this paper, the functional parameters M (message size) and OP (reduce operation) will typically be suppressed from the various terms.

This model modifies the LogP model [13] to better address the needs of this work. The parameter r is used in place of o, the cost to receive a message, and the parameter g is represented as $(r+c)$, the time required to fully process a message. While parameter o simultaneously represents both send and receive overhead in LogP, it is renamed r for clarity since it is only used to

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

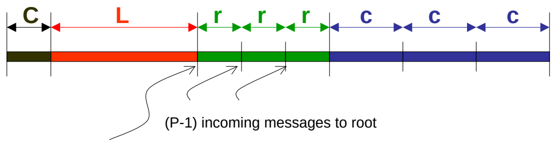


Figure 6: Model of serial reduction latency

account for receive overhead here. Also, the parameter g is split to separate that part of g which is dependent solely on message size, $r(M)$, from that part which is also dependent on the reduction operation, $c(M, OP)$. The message latency, the reception costs, and the reduction costs may all differ between host-based and NIC-based implementations. These redefinitions allow one to explicitly account for those differences with dedicated parameters. Additionally, since r and c may be general functions of the message size, one may better model nonlinearities, such as data packetization and caching, which are relevant for small data sizes.

With this model it is simple to describe the linear form of the serial reduction latency curves:

$$T_{serial}(P) \approx C + L + (P-1) \cdot (r + c).$$

This expression is shown pictorially in Figure 6, which depicts a time-line of the events required for the root node of the serial reduction to receive and reduce $(P-1)$ vectors.

To assign numerical values to the parameters, the values of r and c were extracted from the serial reduction data for various values of M and OP. The terms L and C were fit to the data, and P is given for a particular problem. Note that while r is dependent on the message size in general, it turns out to be constant for the cases we are interested in—reductions involving vector sizes of only a few elements, say up to eight, which all fit into a single 64-byte, fixed-length RDMA transaction on the Quadrics network. Thus, whether the problem involves one-element vectors or eight-element vectors, the receive time is the same—the cost to receive one 64-byte packet.

To provide some context of typical model parameter values, communication and initialization values are given in Table 2 and computation values are listed in Table 3.

Parameter	Value
L	2.90
r	0.42
C	2.70

(a) Host-based

Parameter	Value
L	2.10
r	0.42
C	6.20

(b) NIC-based

Table 2: Comm and Init Parameter Values (μs)

Operation	Number of elements			
	1	2	4	8
Int32 Max	0.03	0.03	0.07	0.13
Int32 Add	0.02	0.03	0.06	0.13
Float64 Max	0.04	0.07	0.14	0.28
Float64 Add	0.02	0.06	0.12	0.16

(a) Host-based

Operation	Number of elements			
	1	2	4	8
Int32 Max	0.27	0.46	0.84	1.60
Int32 Add	0.25	0.44	0.76	1.44
Float64 Max	0.67	1.27	2.44	4.80
Float64 Add	1.50	2.95	5.80	11.56

(b) NIC-based

Table 3: Computation Parameter Values (μs)

These numbers demonstrate many of the design issues previously mentioned. First, the message latency L for NIC-based reductions is less than that for host-based reductions. This highlights the savings in PCI-bus transaction costs. Second, the overhead C is higher for NIC-based reductions due to host-NIC synchronization costs. Finally, the computation costs are much higher for the NIC-based reductions.

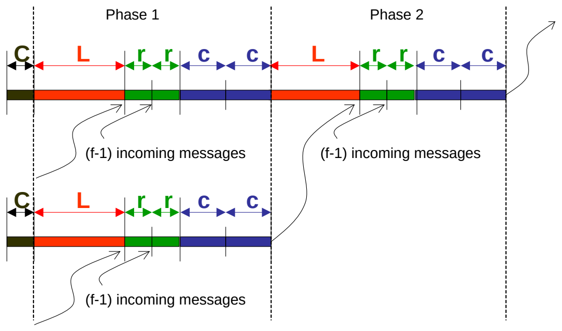


Figure 7: Model of f-nomial reduction latency

5.2 Modeling f-nomial Trees

For a given problem we would like to be able to choose the best f-nomial tree analytically, so now we apply our model to the proposed algorithm. Since the root node of an f-nomial tree is involved in every phase of the algorithm, the latency of the entire operation may be predicted by focusing on the work the root node must do. Assuming a full tree, an f-nomial tree generates $\log_f P$ phases, during each of which the root has $(f-1)$ children. Each phase will be of the linear, building-block form of the serial reduction algorithm previously discussed. In other words, the critical path consists of a series of $\log_f P$ serial reductions, each involving f nodes. Thus, inserting $T_{serial}(f)$ as derived in the previous section, and adjusting for initial overhead, one arrives at the expression

$$\begin{aligned} T^{full}_{fnomial}(P,f) &\approx C + T_{serial}(f)\cdot\log_f P \\ &\approx C + [L + (f-1)\cdot(r+c)]\cdot\log_f P \end{aligned}$$

as a model of the f-nomial reduction latency.

An example application of the model to intermediate phases is shown pictorially in Figure 7. In this figure the two horizontal time-lines represent two intermediate parent nodes in the f-nomial tree, the bottom node being one of the children of the top node. To start, the initial overhead, C, is encountered in parallel across all nodes as a one time cost. Then, after waiting for time L, the two parent nodes each receive and reduce the data from their $(f-1)$ children of the first phase. Starting the second phase, the bottom node, now a child to the top node, immediately sends its partial result to its parent. Again, after time L, the top node receives and reduces the data from its $(f-1)$ children of the second phase. The reduction continues as the top node, now a child to some higher node, sends its partial result to its parent to begin the third phase, which is not shown.

Given the model for $T^{full}_{fnomial}(P,f)$, it is straightforward to compute the optimal degree f to use for a particular problem. Basically, the goal is to find that value of f which minimizes the expression

$$T^{full}_{fnomial}(P, f) \approx C + [L + (f-1) \cdot (r+c)] \cdot \log_f P.$$

To do so, first the derivative of $T^{full}_{fnomial}(P, f)$ is taken with respect to f:

$$\begin{aligned}
&\frac{\partial}{\partial f} T^{full}_{fnomial}(P, f) \\
&\approx \frac{\partial}{\partial f}\{C + T_{serial}(f) \cdot \log_f P\} \\
&= \frac{\partial T_{serial}(f)}{\partial f} \cdot \log_f P + T_{serial}(f) \cdot \frac{\partial \log_f P}{\partial f} \\
&= \frac{\partial[L + (f-1) \cdot (r+c)]}{\partial f} \cdot [\ln P / \ln f] \\
&\quad + [L + (f-1) \cdot (r+c)] \cdot \frac{\partial[\ln P/\ln f]}{\partial f} \\
&= [(r+c)] \cdot [\frac{\ln P}{\ln f}] \\
&\quad + [L + (f-1) \cdot (r+c)] \cdot [-\frac{\ln P}{f \cdot \ln^2 f}] \\
&= (r+c) \cdot \frac{\ln P}{\ln f} - [L + (f-1) \cdot (r+c)] \cdot \frac{\ln P}{f \cdot \ln^2 f}.
\end{aligned}$$

Then this expression is set equal to zero and f is isolated:

$$(r+c) \cdot \frac{\ln P}{\ln f} - [L + (f-1) \cdot (r+c)] \cdot \frac{\ln P}{f \cdot \ln^2 f} = 0.$$

$$\begin{aligned}
(r+c) \cdot \frac{\ln P}{\ln f} &= [L + (f-1) \cdot (r+c)] \cdot \frac{\ln P}{f \cdot \ln^2 f} \\
f \cdot \ln f \cdot (r+c) &= L + (f-1) \cdot (r+c) \\
f \cdot \ln f &= L/(r+c) + (f-1) \\
f \cdot \ln f - f &= L/(r+c) - 1 \\
f \cdot (\ln f - 1) &= L/(r+c) - 1.
\end{aligned}$$

The above expression gives the best value of f to use given L, r, and c. Since this is a transcendental expression, f must be solved for numerically by finding the intersection of $f \cdot (\ln f - 1)$ with the function $L/(r+c) - 1$. In our case, L and r are constants, and c will be determined by the operation and data size of a particular problem. After setting $L = 2.10$ μs and $r = 0.42$ μs, values corresponding to NIC-based reduction, we plotted the intersection of these two functions for various values of c in Figure 8,

Only integers $f \geq 2$ produce valid f-nomial trees. For intersection points which are between two integers, one must choose the best of the two. For the values used for L and r, note that the best degree may fall

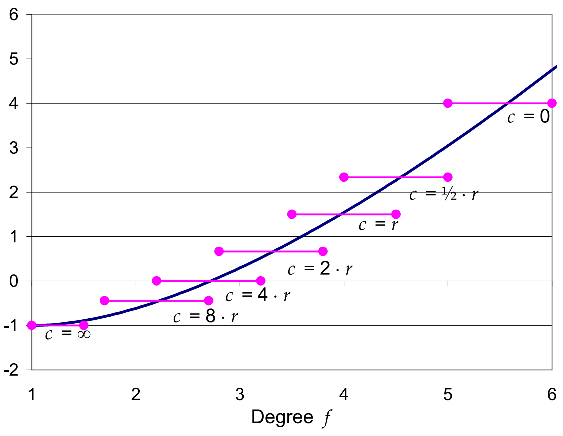


Figure 8: Plot of $f \cdot (\ln f - 1)$ and $L/(r+c) - 1$

anywhere in the range $[2, 6]$ depending on the value of c. The upper bound is reached somewhere between 5 and 6 when $c = 0$. Note when $f = 6$, a parent node receives 5 messages so that the reception costs accumulate to exactly balance the message latency, $5 \cdot r = L$. As computation cost increases, the best degree decreases.

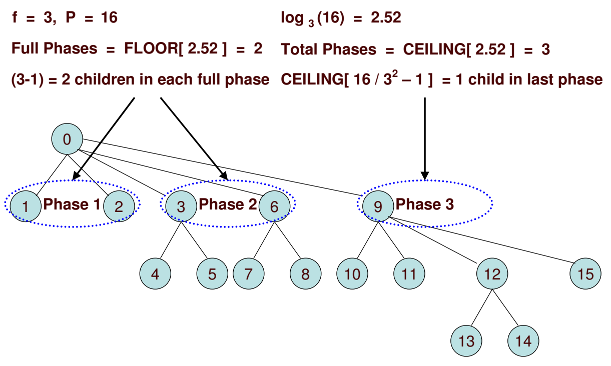
It is interesting to consider the range $[1, 2]$. Values of f smaller than 2 do not produce meaningful f-nomial trees. However, if we choose some arbitrary f in this range, say 1.5, to substitute back in $T^{full}_{fnomial}(P, f)$, we arrive at

$$\begin{aligned}
&T^{full}_{fnomial}(P, 1.5) \\
&\approx C + [L + ((1.5) - 1) \cdot (r+c)] \cdot \log_{(1.5)} P \\
&= C + [L + 0.5 \cdot (r+c)] \cdot \log_{1.5} P
\end{aligned}$$

When compared to binomial trees, this value of f produces trees which have more communication phases, since $\log_{1.5} P > \log_2 P$, in return for reduced amount of reception and computation costs, $0.5 \cdot (r+c)$ instead of $(r+c)$. Thus, trees in this range do more communication to save on computation. This is the range in which optimizations like the vector split are valuable.

5.3 Refining the Model

The expression for $T^{full}_{fnomial}(P, f)$ was derived assuming a full tree, i.e., assuming $\log_f P$ is an integer. The expression for an arbitrary number of nodes is more complex. When the number of nodes is not an integer power f the root may not have a full set of children during the final phase. In this case, the root still incurs the message latency cost L while waiting for the data of the last phase to arrive, however, there will be fewer than the full set of $(f-1)$ messages to receive



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
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Figure 9: Application of reduction latency model

and reduce. A more detailed analysis will show that

$$\begin{aligned} T_{fnomial}(P,f) \approx\; & C + L \cdot \lceil \log_f P \rceil + \\ & (r+c)\cdot(f-1)\cdot\lfloor \log_f P \rfloor + \\ & (r+c)\cdot\lceil P/f^{\lfloor \log_f P \rfloor} - 1\rceil. \end{aligned}$$

Here $\log_f P$ represents roughly the number of phases in the f-nomial tree. In particular $\lceil \log_f P \rceil$ is the *total* number of phases, while $\lfloor \log_f P \rfloor$ is the number of *full* phases, i.e., those in which the root has a full set of $(f-1)$ children. The term involving L accounts for the message latency cost incurred from each phase of the tree. The last two $(r+c)$ terms together sum the reception and reduction costs incurred for processing each child. Of these two terms, the first counts the number of children processed in the full phases, while the second counts the number of children in the final phase, if less than a full set. An example given in Figure 9 demonstrates how these terms apply to a 16-node, 3-nomial tree.

With this expression for $T_{fnomial}(P,f)$ it is nontrivial to express the best degree f in terms of the other model parameters. However, in practice the best degree tends to be small, so a small set of values may be evaluated numerically to find the best one. This approach is illustrated graphically when the model is validated in Section 6.3.

6 Experiments

Various versions of the f-nomial algorithm were implemented for experimental purposes. Results from these tests are presented in this section to validate design choices and to illustrate the benefits of NIC-based reduction. The algorithms were developed and initial performance evaluations were taken on the "crescendo" cluster at Los Alamos National Laboratory, which consists of 32 dual-processor nodes with 1.0 GHz Pentium IIIs and the Quadrics QsNet network. Scalability analysis was performed on the ALC located at Lawrence Livermore National Laboratory. The ALC uses 960 dual-processor nodes with 2.4GHz Xeons and the Quadrics QsNet network.

6.1 Implementation and Testing Details

This section provides details about the implementation and testing methods relevant for proper interpretation of the results given in the following sections.

First, each node implements NIC-based reduction using a single thread running on a single NIC, regardless of the number of local host processes. When multiple host processes are involved on a single node, the host processor is used to first reduce the local data vectors in shared memory before initiating the NIC-based portion of the algorithm. In NIC-based reduction, one accepts the increased computational cost associated with performing reduction processing on the slower NIC processor in return for elimination of extraneous data transfers between the host and network. However, if a collection of data is already located in host memory, one may as well use the faster host processor to reduce it. In addition to the obvious computational savings, less data needs to be sent through the PCI-bus.

Second, for timing purposes, a barrier was inserted between each of the NIC-based reductions in order to serialize consecutive invocations. Since QsNet provides a hardware-based barrier mechanism, such barriers keep the distributed nodes very tightly synchronized. Although such synchronization is not required for reduction, the measurement procedure is simplified since there is no need to worry about pipelining effects due to nodes starting the next operation before the previous one has completed.

Third, for host-based reduction we used the reduce collective from the vendor-provided, production-level MPI library. The MPI implementation internally delegates the work to a reduction function, called elan_reduce(), provided in the lower-level Quadrics Elan library [23]. The Elan algorithm, in turn, performs a reduction via a 4-ary tree followed by a hardware-based broadcast of the result. This trailing broadcast simultaneously serves as a global synchronization step and acts to extend the reduce into an allreduce. Thus, the elan_reduce() function implements allreduce rather than reduce, as used in the NIC-based

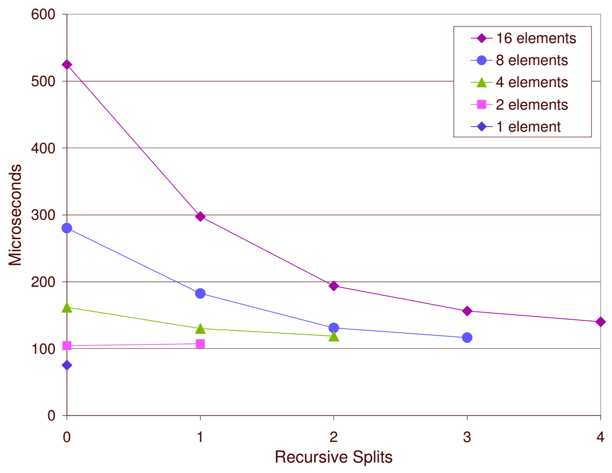


Figure 10: f-nomial split on various vector sizes

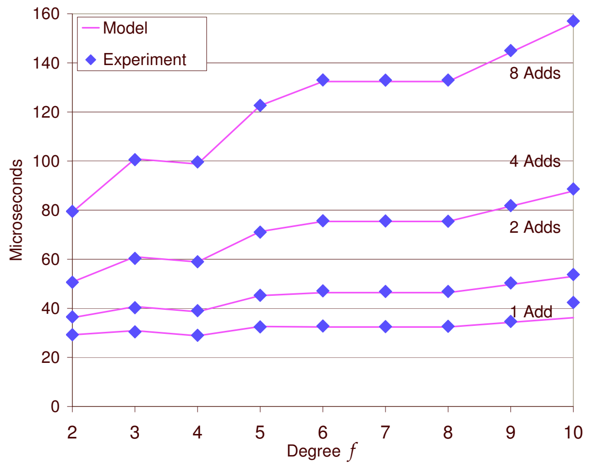


Figure 11: Predicted and measured NIC-based latencies

reduction. Even so, the tests remain fair because the cost of the barrier inserted between each of the NIC-based reductions offsets the cost of the broadcast that completes each of the host-based reductions.

Finally, when taking measurements, we found a large variance in the reduction latency from one invocation to another, especially for host-based reductions. Unless otherwise stated, the reported reduction latency as the average latency over 100,000 iterations.

6.2 Validating the Vector Split Optimization

By increasing parallelism in NIC-based reductions the vector split optimization can save significant computation costs at the expense of additional communication. This section validates this claim.

The performance of the NIC-based f-nomial split algorithm for 64-bit floating-point addition on 512 nodes was measured for various vector sizes. The results are shown in Figure 10, where the horizontal axis represents the number of recursive splits the vector undergoes before its pieces are reduced through f-nomial reduction. One split implies that the vector is broken into halves, two splits implies quarters, and so on. Data points are not shown if the corresponding reduction vector contains fewer elements than pieces implied by a given number of splits.

The effect of the vector split optimization for multi-element vectors is quite pronounced. After three recursive splits, the 8-element latency is improved by nearly a factor of three, while for four recursive splits, the 16-element case is over three times faster. The trend suggests that the larger the vector, the greater 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 140μs for a 16-element reduction may still be much more than what a host-based implementation could provide. And, interestingly, one may note that the latency for a two-element vector actually increases slightly after one split. This of course will happen if the total savings in computation is less than the added communication cost of the distribution and recombination steps. However, the crossover point can be computed to always choose the better of the two options. Van de Geijn discusses the details in [27].

6.3 Validating the Model and f-nomial Trees

NIC-based reduction latency is dependent on the degree of the f-nomial tree used. In order to pick the best f-nomial tree, we would like to rely on our model. This section illustrates the accuracy of the model and the impact of changing f for a given problem.

Figure 11, shows predicted and measured NIC-based f-nomial reduction latencies as a function of the degree f. The plots correspond to 64-bit floating-point addition on a 31-node system using vectors sizes of 1, 2, 4, and 8 elements. Here, the refined f-nomial tree model from Section 5.3 uses the NIC-based parameter values given in Section 5.1, which were derived from serial reduction tests on crescendo.

The figure indicates that the model predicts NIC-based reduction latencies with high accuracy. For instance, when choosing among NIC-based f-nomial trees, the model correctly indicates that a degree of 4 is best for 64-bit floating-point addition of 1-element vectors, and a degree of 2 is best for 2, 4, and 8-element vectors. Although not shown here, the model also accurately predicts host-based reduction performance. This allows us to extrapolate algorithm scalability and consider trade-offs between design choices analytically. This is important because the problem parameter space is large and opportunities to run tests on large-scale clusters are rare.

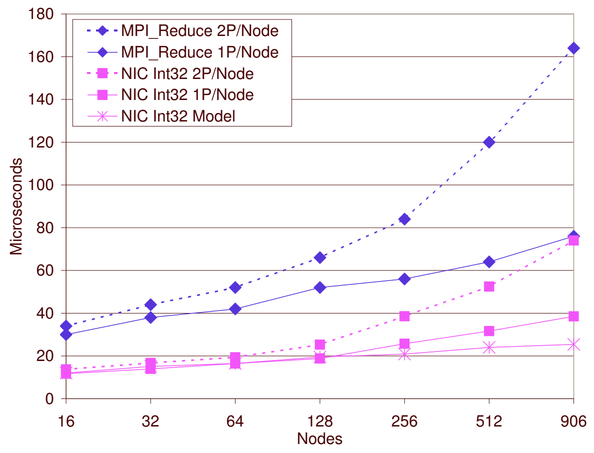
Note how the degree of the f-nomial tree affects reduction latency. Small vectors, which require less processing time, lead to curves that are essentially flat for the degrees tested, while larger vectors tend to heavily favor lower-degree trees: compare the one-element curve to the eight-element curve. Additionally, although not shown, reduction operations simpler than floating-point addition strongly favor higher-degree trees. Such variation does not exist in host-based reduction, where message processing time is effectively independent of both the vector size and the type of computation being performed.

6.4 Latency Measurements

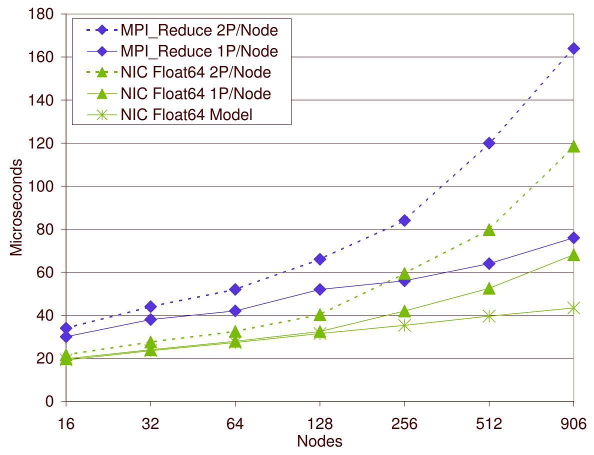
We timed the latencies for host-based and NIC-based reduction over a variety of operations and data sizes, using both one and two processes per node. We found that NIC-based reductions are capable of completing with lower latencies than host-based versions. To illustrate this point we show the single-element vector results obtained for host-based and NIC-based 32-bit integer addition in Figure 12(a) and 64-bit floating-point addition in Figure 12(b). In all measurements we consider a 4-nomial tree, which provides the best performance for the configurations used in the experiments.

The NIC-based implementations scale considerably better than the host-based ones. Indeed, as one may infer from the 32-bit integer addition plot, our NIC-based implementation was able to perform simple integer reductions in about half the time it took the host to do so. Furthermore, even with the cost of emulating floating-point addition on a much slower processor, the NIC-based implementation was able to substantially outperform the host-based reduction. When reducing over 906 nodes, we were able to obtain latencies as low as 40μs for integer operations and a slightly higher time of 65μs for floating-point. In the largest configuration tested—1812 processors—our NIC-based algorithm summed single-element vectors of 32-bit integers and 64-bit floating-point numbers in 73μs and 118μs, respectively. These results represent respective improvements of 121% and 39% over the host-based, production-level MPI library.

We also note that system noise does have an effect in NIC-based reductions. This is apparent when comparing the latencies recorded for the case of two processes per node to those obtained for one process per node, and when comparing the performance predicted by the model to the actual measurements. The NIC-based implementation is subject to host-level process interference during the time it takes the host processes to initiate the reduction operation. Once initiated, however, the NIC-based algorithm avoids process interference throughout the execution of the reduction.



(a) 32-bit integer addition



(b) 64-bit floating-point addition

Figure 12: Host-based and NIC-based reduction latencies

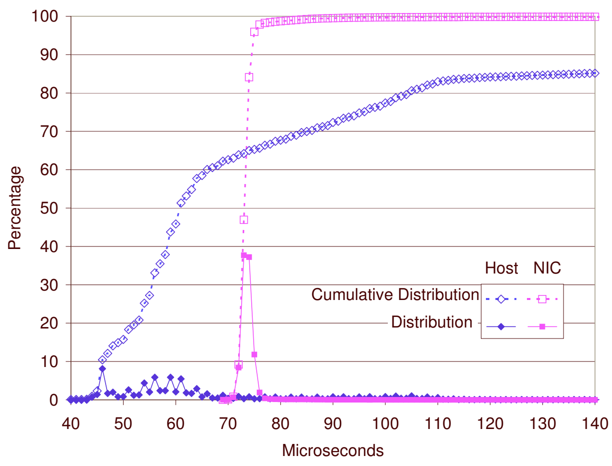


Figure 13: Reduction latency distributions over 900 nodes

As a result, our NIC-based reduction implementation is only marginally affected by the system noise when compared to the host-based results.

6.5 Consistency Measurements

Because NIC-based reductions avoid much of the process interference that host-based implementations are subject to, NIC-based reductions execute with more consistent latencies than host-based implementations. In our tests, the host-based latencies varied substantially from one invocation to another. The best time recorded for an individual invocation was about three times better than the average. The NIC-based results, on the other hand, were quite consistent

To further illustrate this point, Figure 13 shows a distribution graph of the latencies recorded for NIC-based and host-based 64-bit floating-point addition of a single-element vector over 900 nodes. Unlike measurements for the average reduction latency, to obtain this distribution, 100,000 reduction invocations were timed individually, and the resulting set was binned to yield a histogram.

Note that the NIC-based latencies are largely contained within a sharp spike, while the host-based latencies are spread across a wide range of values. To be precise, 97% of the NIC-based reductions fall with a spread of only 4μs, while for host-based reductions, only 57% fall within a spread of 20μs. Indeed, a substantial percentage of host-based latencies extend far beyond the right-hand limit of the graph. After discarding the highest 1% of the samples, the statistics in Table 4 were calculated.

Note the drastic, two order-of-magnitude difference in the standard deviations (SD). This large difference

Reduction	Average (μs)	SD (μs)
host-based	89.30	65.26
NIC-based	73.67	0.29

Table 4: Reduction latency statistics over 900 nodes

in consistency is indicative of the nondeterministic effect that process interference imposes on host-based reduction implementations. Even when the lowest recorded host-based latency is faster than the lowest recorded NIC-based latency, the spread in host-based latencies often pushes its average higher. In this case, for example, the average host-based latency, at 89μs, is substantially higher than the NIC-based average, at 74μs. By avoiding host-level process interference, we found that NIC-based reductions are able scale significantly better than host-based versions.

7 Conclusions

Modern cluster interconnects provide programmable processors and local memory on the network interface card (NIC). We successfully exploited these features in the Quadrics QsNet to implement reduction algorithms on the NIC, as opposed to the host processor where reductions are traditionally performed. The biggest challenge we faced was the slow speed and limited functionality of the NIC processor. Overcoming these obstacles involved designing a family of algorithms for a range of problem configurations, deriving a communication and computation model to select from among them, and implementing IEEE-compliant floating-point operations on an integer-only processor. We illustrated how NIC-based reductions gain efficiency over host-based versions by eliminating data transfers between the host and the network, as well as by avoiding host-level process interference. We found that NIC-based reductions outperform host-based versions in two important ways: reduced latency and increased consistency.

Our experimental results demonstrate low latency and impressive scalability. In the largest configuration tested—1812 processors—our NIC-based algorithm summed single-element vectors of 32-bit integers and 64-bit floating-point numbers in 73μs and 118μs, respectively. These results represent respective improvements of 121% and 39% over the host-based, production-level MPI library. In addition, the standard deviations in timings for the NIC-based reductions were as much as two orders of magnitude smaller than the host-based equivalents.

ACKNOWLEDGEMENTS

We would like to thank Robin Goldstone, Jim Garlick, Moe Jette, and Ryan Braby at Lawrence Livermore National Laboratory, and David Addison at Quadrics, for providing us with the opportunity to run experiments on the ASCI Linux Cluster. We would also like to thank our anonymous reviewers for their time and many instructive comments.

This work was partially supported by the U.S. Department of Energy through Los Alamos National Laboratory contract W-7405-ENG-36 and by the Spanish MCYT under grant TIC2003-08154-C06-03.

REFERENCES

1 V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.T. Ho, S. Kipnis, and M. Snir. CCL: A Portable and Tunable Collective Communication Library for Scalable Parallel Computers. In *Proceedings of the 8th International Parallel Processing Symposium*, pages 835–844, Cancun, Mexico, April 1994.

2 A. Bar-Noy, S. Kipnis, and B. Schieber. Optimal Computation of Census Functions in the Postal Model. *Discrete Applied Mathematics*, 58(3):213–222, April 1995.

3 M. Barnett, R. Littlefield, D.G. Payne, and R.A. van de Geijn. Global Combine on Mesh Architectures with Wormhole Routing. In *Proceedings of the 7th International Parallel Processing Symposium*, pages 156–162, Newport Beach, California, April 1993.

4 M. Barnett, L. Shuler, S. Gupta, D.G. Payne, R.A. van de Geijn, and J. Watts. Building a High-Performance Collective Communication Library. In *Proceedings of the Supercomputing Conference*, pages 107–116, Washington D.C., November 1994.

5 M. Bernaschi and G. Iannello. Collective Communication Operations: Experimental Results vs. Theory. *Concurrency: Practice and Experience*, 10(5):359–386, April 1998.

6 R. Bhoedjang, T. Ruhl, and H. Bal. Efficient Multicast on Myrinet Using Link-Level Flow Control. In *Proceedings of the International Conference on Parallel Processing*, pages 381–390, Minneapolis, Minnesota, August 1998.

7 Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawick, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, January 1995.

8 J. Bruck, L. de Coster, N. Dewulf, 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):256–265, March 1996.

9 D. Buntinas and D.K. Panda. Fast NIC-Based Barrier over Myrinet/GM. In *Proceedings of the International Parallel and Distributed Processing Symposium*, San Francisco, California, April 2001.

10 D. Buntinas and D.K. Panda. NIC-Based Reduction in Myrinet Clusters: Is It Beneficial? In *Proceedings of the Workshop on Novel Uses of System Area Networks*, pages 22–33, Anaheim, California, Febuary 2003.

11 D. Buntinas, D.K. Panda, J. Duato, and P. Sadayappan. Broadcast/Multicast over Myrinet using NIC-assisted Multidestination Messages. In *Proceedings of the Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing, High Performance Computer Architecture Conference*, pages 115–129, Toulouse, France, January 2000.

12 M. Collette. LLNL User Briefings. In *ASCI Q LANL/HP Technical Quarterly Meeting*, Santa Fe, New Mexico, March 2003.

13 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 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, San Diego, California, May 1993.

14 M. Gerla, P. Palnati, and S. Walton. Multicasting Protocols for High-Speed, Wormhole-Routing Local Area Networks. In *Proceedings of the ACM SIGCOMM Symposium*, pages 184–193, Stanford, California, August 1996.

15 J.R. Hauser. SoftFloat.

16 C. Huang and P.K. McKinley. Efficient Collective Operations with ATM Network Interface Support. In *Proceedings of the International Conference on Parallel Processing*, volume 1, pages 34–43, Bloomingdale, Illinois, August 1996.

17 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 the Supercomputing Conference*, Denver, Colorado, November 2001.

18 R. Kesavan and D.K. Panda. Optimal Multicast with Packetization and Network Interface Support. In *Proceedings of the International Conference on Parallel Processing*, pages 370–377, Bloomingdale, Illinois, August 1997.

19 C. E. Leiserson. Fat-trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

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, January/Febuary 2002.

21 Fabrizio Petrini, Salvador Coll, Eitan Frachtenberg, and Adolfy Hoisie. Hardware- and Software-Based Collective Communication on the Quadrics Network.

In *IEEE International Symposium on Network Computing and Applications 2001 (NCA 2001)*, Boston, MA, February 2002.

22 Fabrizio Petrini, Darren Kerbyson, and Scott Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *Proceedings of SC2003*, Phoenix, Arizona, November 10–16, 2003.

23 Quadrics Supercomputers World Ltd. *Elan Programming Manual*, 2nd edition, December 1999.

24 Quadrics Supercomputers World Ltd. *Elan Reference Manual*, 1st edition, January 1999.

25 Quadrics Supercomputers World Ltd. *Elite Reference Manual*, 1st edition, November 1999.

26 M. Seager. Planned Machines: ASCI Purple, ALC and M&IC MCR. In *Proceedings of the 7th Workshop on Distributed Supercomputing*, Durango, Colorado, March 2003.

27 R.A. van de Geijn. On Global Combine Operations. April 1991.

28 K. Verstoep, K. Langendoen, and H. Bal. Efficient Reliable Multicast on Myrinet. In *Proceedings of the International Conference on Parallel Processing*, volume 3, pages 156–165, Bloomingdale, Illinois, August 1996.

29 J. Vetter and F. Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Fort Lauderdale, Florida, April 2002.

30 W. Yu, D. Buntinas, and D.K. Panda. High Performance and Reliable NIC-based Multicast over Myrinet/GM-2. In *Proceedings of the International Conference on Parallel Processing*, Kahosiung, Taiwan, October 2003.