NIC-based Reduction on Large-Scale Quadrics Clusters

A Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Master of Science in the
Graduate School of The Ohio State University

By

Adam Moody, B.S.

* * * * *

The Ohio State University

2003

Master's Examination Committee:

Prof. Dhabaleswar K. Panda, Advisor

Prof. Mario Lauria

Approved by

____

Advisor
Department of Computer
and Information Science

© Copyright by

Adam Moody

2003

ABSTRACT

Many high-performance parallel computing applications require efficient reduction collectives. In response, researchers have implemented algorithms considering a range of design factors including data size, system size, and communication characteristics. In all of this research, designers were forced to perform reduction processing on the host CPU, the only processor available. Today, however, network interface cards (NICs) for modern cluster interconnects, such as Quadrics, provide programmable processors with substantial memory, and thus introduce a fresh variable into the equation.

In this thesis, the benefits of NIC-based reduction implementations are investigated, especially in the context of large-scale clusters. Design issues are presented, along with the chosen solutions and alternatives. Performance benefits and penalties are assessed both analytically, through a proposed model, and numerically, through experimental results.

It is shown that large-scale clusters may benefit from NIC-based reductions in the common case. NIC-based reductions avoid inefficiencies between the processors and the network, as well as, between the processors and the operating system that are inherent to host-based reductions. This enables NIC-based reductions to execute with reduced latency on a more consistent basis than their host-based counterparts. Experimental results support this. In particular, in the largest configuration tested

—1812 processors— NIC-based reduction implementations summed 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 host-based implementations.

I dedicate this work to Leslie Lockard and to my parents, Mark and Kathy Moody, who have provided me with unconditional support and constant encouragement.

ACKNOWLEDGMENTS

I would very much like to thank my co-advisors, Dr. Dhabaleswar K. Panda and Dr. Fabrizio Petrini, for their invaluable guidance while directing me along the path of computer science research. They provided the fundamental knowledge which got me started, the advice and suggestions which kept me going, and the motivation and encouragement which helped me finish. I am extremely grateful for their enduring assistance and patience as I made my way.

I would like to thank Dr. Mario Lauria for volunteering to serve as a member on my Master's examination committee.

I thank the entire CCS-3 Modeling, Algorithms, and Informatics Group of the Computer and Computational Sciences Division of Los Alamos National Laboratory. They provided me with the technical advice and assitance, as well as, the financial support which enabled me to do this work. Particularly, I would like to mention Juan Fernandez, who allocated much of his time to orient me, answer my questions, and provide assistance with many of the techincal details of my work. Additionally, I would like to thank Erika Maestas and Tabitha Valdez who patiently initiated and maintained all of the paperwork necessary to ensure my financial support and remote access capabilities.

I would also like to thank the many members associated with the NOWLAB of the Systems Group in the Computer Science Department of the Ohio State University for

sharing their ideas and comments with me. I learned a great deal from our discussions and value the many friendships I made.

Finally, I thank all of my friends and family; especially Leslie Lockard, my parents, Mark and Kathy Moody, my brother Matt Moody, and my grandmothers, Freda Moody and Emily Gambrel. They provided the support and encouragement that I needed to do my work, as well as, the love and friendship that made my time and effort so enjoyable.

VITA

January 15, 1979	Born - Zanesville, Ohio, USA
September 1997 - June 2001	B.S. Computer Science & Engineering, The Ohio State University
September 1997 - June 2001	B.S. Engineering Physics, The Ohio State University
September 2001 - Present	Graduate Research Associate, The Ohio State University

PUBLICATIONS

Adam Moody, Juan Fernandez, Fabrizio Petrini, and Dhabaleswar K. Panda, "Scalable NIC-based Reduction on Large-scale Clusters", In *Proceedings of the Supercomputing Conference*, to be presented, Phoenix, Arizona, November 2003.

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in High-Performance Computing: Prof. D.K. Panda

TABLE OF CONTENTS

Page

Abstract ii

Dedication iv

Acknowledgments v

Vita vii

List of Figures xi

Chapters:

1. Introduction 1

    1.1 More Speed – Fast Networks, User-Level Protocols, and RDMAs . 2
    1.2 More Functionality – Programmable NICs and Hardware Support for Collective Communications 5
    1.3 Reduction – A Collective Communication Operation 7
    1.4 Problem Statement 8
    1.5 Solution Approach 10
    1.6 Thesis Overview 10

2. Background & Motivation 11

    2.1 The Network – The Quadrics Interconnect 11
        2.1.1 The Network Interface Card – The Elan 11
        2.1.2 The Network Switch – The Elite 12
    2.2 Motivation for NIC-based Collectives 13
        2.2.1 Elimination of PCI-bus Trasactions 13
        2.2.2 Avoidance of Host-level Process Interference 14

2.3 Related Work 17
2.4 Thesis Contributions 18
2.5 Summary 19

3. Design Issues & Solutions 20

3.1 NIC Processor Capability 20
3.2 NIC Processor Speed 21
3.2.1 Serial Reduction 22
3.2.2 Simple Operations and Small Data Sizes 24
3.2.3 f-nomial Trees – Generalized Binomial Trees 27
3.2.4 Vector Split Optimization 31
3.3 Host-NIC Synchronization Overhead 34
3.3.1 Minimizing the Overhead 34
3.3.2 Hiding the Overhead 36
3.4 Summary 38

4. Analytical Models 40

4.1 The Model Parameters 40
4.2 Modeling f-nomial Trees 44
4.2.1 Full f-nomial Trees 45
4.2.2 Finding the Optimal Degree 46
4.2.3 Refining the Model 49
4.3 Summary 51

5. Experiments 52

5.1 Procedural Details 52
5.2 Validation 55
5.2.1 Validating the Vector Split Optimization 55
5.2.2 Validating the Host-NIC Synchronization Overhead 56
5.2.3 Validating the Model 58
5.3 Elimination of PCI-bus Transactions 59
5.3.1 Reduced Latency 59
5.4 Avoidance of Host-level Process Interference 61
5.4.1 Increased Consistency 61
5.4.2 Reduced Latency 63
5.5 Summary 65

6. Conclusions & Future Work 67

6.1 Conclusions 67
6.2 Future Work 69

Bibliography 74

LIST OF FIGURES

Figure **Page**

2.1 MPI Barrier and Reduce Latencies 15

3.1 Serial Reduction Latencies 23

3.2 Profile of MPI_Allreduce Operations in SAGE 26

3.3 4-nomial Reduction over 16 Nodes 30

3.4 f-nomial Trees of Varying Degrees covering 16 Nodes 31

3.5 Vector Split Optimization 33

3.6 Host-NIC Synchronization Overhead 37

4.1 Model of Serial Reduction Latency 43

4.2 Model of f-nomial Reduction Latency 46

4.3 Plot of $f \cdot (\ln f - 1)$ and $L/(r + c) - 1$ for $L = 2.10$ μs, $r = 0.42$ μs, and various c 48

4.4 Application of Reduction Latency Model to 16-Node 3-nomial Tree . 50

5.1 f-nomial Split on Various Vector Sizes for 64-bit Floating-Point Addition over 512 Nodes 56

5.2 NIC-based and Host-based Latencies for Single-Element 32-bit Integer Addtion using Binomial Trees 57

5.3 Predicted and Measured NIC-based Latencies for 64-bit Floating-Point Addition over a 31-Node f-nomial Tree 59

5.4 NIC-based and Host-based Latencies for Various Operations using Binomial Trees 60

5.5 Reduction Latency Distributions for Single-Element 64-bit Floating-Point Addition over 900 Nodes 62

5.6 Host-based and NIC-based Reduction Latencies for Single-Element Vectors 64

CHAPTER 1

INTRODUCTION

As market pressures in the personal computer industry continually act to provide improved processor performance at decreased cost, more and more of those in demand of high-performance computing systems find that cluster computing provides an economical solution. There are three basic components required to construct a high-performance computing system (a.k.a., a supercomputer): processors, an operating system, and a communication network. By connecting a collection of commodity processors running a commodity operating system over a commodity network, clusters enable powerful parallel-processing machines to be built very affordably. Because of this, cluster supercomputers continue to gain widespread use in industrial and governmental computing labs.

As popularity for cluster computing systems grows, the research community works to make them more efficient. For a supercomputer to operate efficiently, each of its constituent components must operate efficiently as an individual unit, and the integration of those components must operate efficiently as a collective system. In the past, single companies, such as Cray Inc., commonly constructed complete supercomputer systems for its customers. In this business model, the company had intricate knowledge and control over the design of the components, so it could develop them

within the context of one another for efficient integration. Today, under the paradigm of cluster computing, industry still ensures the efficiency of each of the commodity components used to build clusters. However now, those components are developed independently of one another, and the non-trivial task of integration has been largely left as an open problem for the research community.

This thesis contributes to the evolving solution for efficient component integration. In particular, this work shows that novel use of modern cluster interconnect technology during reduction collective communication operations improves overall system performance by avoiding inefficiencies residing between the processors and the network, as well as, between the processors and the operating system. The remainder of the chapter briefly presents the major architectural modifications leading from early cluster systems to the current state of the art, as well as, the issues this work addresses and the approach it takes to do so.

1.1 More Speed – Fast Networks, User-Level Protocols, and RDMAs

The earliest clusters were constructed by connecting workstation PCs through basic networking technologies like Ethernet. While a large amount of processing power could be aggregated through such collections of workstations, the existing slow, bus-based interconnects limited its utilization. In addition to raw processing power, parallel systems often demand low latency and high bandwidth communication between distributed processors. However, slow interconnects naturally lead to high latency and low bandwidth. Additionally, bus-based interconnects, which serialize communication, amplify latency and bandwidth problems whenever multiple nodes

wish to send data simultaneously. This early cluster networking hardware failed to meet the communication requirements of high-performance processing applications.

Thus, the first optimizations came in the form of faster interconnects. While some improvement was acheived by using faster versions of old bus-based networks, the most progress was acheived by employing fast, switch-based interconnects like Asynchronous Transfer Mode. Faster interconnects reduce latency and increase bandwidth, while switch-based interconnects allow multiple nodes to send data simultaneously. Today, cluster interconnects support very low latency and very high, full-bisection bandwidth communication between nodes.

With the newly developed interconnects, communication bottlenecks shifted from networking hardware to networking software. Workstations are designed to provide protected access to system resources, such as the communication system, which must be shared among multiple user processes that may be multitasked on the machine. In early workstations, the operating system was used to enforce this protection; only the kernel was able to access hardware directly. A user application wishing to access the communication system had to call upon the operating system to do so on its behalf. This required context switches between the user and the kernel processes, as well as, memory copies to move message data to and from buffers located in user memory and throughout the kernel's communication protocol stack. Compared to the raw performance available through the new networking hardware, the old networking software used in early clusters added substantial communication overhead.

Thus, the next optimization came in the form of improved communication software. *User-level protocols*, such as EMP[29], Elan[25], FM[22], and VIA[13], were

developed to provide user processes with protected access to the communication system without invoking the kernel. Protection is implemented outside of kernel space in shared libraries directly accessible to user processes. Additionally, these protocols reduce the number of data copies required during communication through the use of *data descriptors*. Data descriptors are small data structures associated with message data that list its location, its size, and other relevant attributes. The communication software copies these structures for internal processing rather than the data itself, which is less costly since the descriptors are often smaller than their associated data. Today, clusters support user-level protocols that employ reduced data copying strategies.

Further enhancement came with the addition of *Remote Direct Memory Access* (RDMA) protocols. RDMAs allow for one-sided data transfer between remote processors, i.e. the remote processor need not explicitly participate in the exchange. Transfer operations include PUT, which transfers data to a remote process address space, and GET, which acquires data from a remote process address space.

RDMAs further improved the latency and bandwidth of a cluster computing system. Without RDMA capability, a local processor expecting to receive data from a remote processor must post a receive descriptor listing, among other information, the size and location of the destination buffer. The receive descriptor informs the communication system about what to do with the data when it arrives. Since distributed processors execute asynchronously in a cluster, incoming data may arrive before the receiving processor posts the corresponding receive descriptor. It this case, the communication system may select one of two options: 1) store the data in a temporary location and copy it to its destination buffer once the descriptor is posted, or 2) toss

the data out and instruct the sender to resend it at a later time, hopefully sometime after the descriptor has been posted. The extra data copy in the first option increases latency, while the extra data transfer in the second option decreases bandwidth. With RDMA, however, the received data is written directly to its destination buffer regardless of the state of the receiving processor. Whenever the receiving processor is lagging, better latency and bandwidth is acheived using RDMA.

A consequence of such architectural modifications and optimizations, the most popular cluster interconnect technologies available today, such as Quadrics[23], Myrinet[7], and the InfiniBand Architecture[18], offer communication latencies on the order of microseconds and data transfer rates on the order of gigabits per second. These performance numbers represent 1000-fold improvements over the original Ethernet networks used in the earliest clusters. This has largely removed the bottleneck previously imposed by the communication network, and future interconnects promise even lower latencies and higher bandwidths to keep pace with increasing processor speeds.

1.2 More Functionality – Programmable NICs and Hardware Support for Collective Communications

Distributed nodes executing parallel applications communicate with one another to exchange data and synchronize processing steps. Typically, the more one divides a given problem among a set of distributed processing nodes, the more those nodes must communicate with one another to find a solution. In early cluster systems, the same processors executing the application were also responsible for handling the network traffic. This placed a limit on the processing scalability of the cluster. The larger the set of processors used to solve a problem, the more time those processors spent handling network messages and less time processing the application. These early

cluster systems failed to scale well since application processors became overwhelmed with network processing when many nodes were involved.

Modern cluster interconnects now provide processors on the network interface card (NIC), which may be programmed by the user to handle basic network processing tasks. Tasks delegated to NIC processors are refered to as *NIC-based*, while those left to the host processors are called *host-based*. NIC-level processors are typically much slower, offer much less functionality, and have much less memory than their host-level counterparts. However, NIC processors have direct access to the network link and support instructions to perform specialized network operations. The motivation is to free host processors for application processing by offloading simple but common network processing tasks to limited but specialized NIC processors. Today, cluster interconnects provide such NIC processors to improve system scalability.

In addition to point-to-point communication operations, parallel applications commonly employ *collective communication operations* (*collectives*, for short). Collectives are parallel programming primitives in which a group of distributed processors cooperate to achieve a particular task. Collectives provide basic synchronization, data exchange, and distributed processing capabilities over a group of processors. Being primitives, many parallel applications are built on top of collectives, so it is essential that they execute efficiently. Early clusters implemented collectives in software using the point-to-point mechanisms supported by the hardware. While efficient communication algorithms were applied, these implementations failed to scale as desired.

Thus now, like basic point-to-point communications, modern cluster interconnects provide hardware support for collective communications. Recent interconnect switches are capable of replicating incoming messages to multiple outgoing links,

which assists implementations of collectives that distribute information, such as multicasts, broadcasts, and barriers. Quadrics, a very sophisticated network, also provides acknowledgement aggregation at network switches, which aids development of many collectives that collect information. Such hardware-based and hardware-assisted collectives are much more efficient and scalable than their software-based equivalents.

The recent functionality incorporated into cluster interconnects allows for awesome scalability. Scalability, combined with the affordability of commodity components, has enabled organizations to construct massive cluster supercomputing systems that contain thousands of nodes and tens of thousands of processors that collectively compute tens of trillions of operations per second. Future interconnects promise to deliver even more functionality, while researchers actively explore mechanisms to take further advantage of what currently exists.

1.3 Reduction – A Collective Communication Operation

Reductions are a particular type of distributed processing collective in which participating processors work together to summarize some property of a set of data partitioned among them. As an example, the goal may be to find the maximum value in a distributed set of integers. Other commonly used operations include finding the minimum and computing the sum. There are two common flavors of reductions, *allreduce* and *reduce*, being distinguished by which processors require the summarized result. In an allreduce, all processors are notified of the result, while a reduce provides the result to a single processor called the *root*.

In practice, it is common to need the same summary information about multiple sets of distributed data of the same type, i.e. find the various maximums of several

distributed sets of integers. Reductions handle this by operating on the data sets in a vector fashion. That is, data elements from the various sets are merged into a contiguous group called the reduction *vector*. The vector is transfered as a single unit and operations are performed in a element-wise, vector fashion to maintain integrity across the various data sets. Such bulk transfer and batch processing of the reduction vector is more efficient than performing separate reductions for each data set individually.

1.4 Problem Statement

Many high-performance cluster 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 [24]. Similar results have been provided by an in-depth analysis of the scientific workload at Lawrence Livermore National Laboratory [12]. Reduction algorithms which minimize latency will thus substantially reduce the execution time of such programs.

The problem of developing efficient reduction algorithms has proven to be a rather rich area of research. Over the years, many researchers have committed significant time in order to derive optimal and scalable algorithms [1, 2, 3, 4, 5, 8]. These algorithms differ primarily in their assumptions about the characteristics of the underlying interconnect system.

While the collection of previous research was thorough during its time, today, the standard cluster environment is quite different. The additional speed and functionality provided by modern cluster interconnects significantly changes the underlying

network characteristics. This thesis investigates reductions from this new perspective. It takes aim at answering two questions posed from two observations of modern cluster systems.

Observation 1: Inter-node latencies within early cluster interconnects were so high that intra-node latencies, such as those incurred when transfering data between the processors and the network across the PCI-bus, were negligible. However, with today's very low latency interconnects, these same internal latencies contribute a significant cost to end-to-end message latencies.

Question 1: *Since NIC-based reductions send and receive messages directly at the link, can they improve overall cluster system performance by avoiding PCI-bus transaction latencies between the processors and the network?*

Observation 2: While cluster scalability has improved using the new interconnect technologies, researchers have discovered that process interference limits further scalability of reduction on large-scale clusters[24]. Application processes executing the reduction contend with operating system processes for control of the host processor. Because operating system processes run infrequently, such interference is unlikely to occur on a given processor at a given moment. However, when considering a large collection of processors at a time, the chances are greatly increased that at least one of them will suffer from such interference. This interference has a severe impact on reductions on large-scale systems, since the entire reduction is affected if any one of the many involved processors is affected.

Question 2: *Since NIC-based reductions run on the NIC processor, can they improve overall cluster system performance by avoiding process interference with the operating system running on the host processors.*

1.5 Solution Approach

This thesis addresses the design issues encountered and the solutions used to develop NIC-based reductions on the Quadrics network. The major issues include processing on the slower and less functional NIC processor and handling overhead costs when initiating and finalizing NIC-based reductions. Various NIC-based reduction algorithms are implemented, which operate in-part or in-full on the NIC processors. These implementations execute with little or no assistance from the host processors. A model is developed for algorithm analysis, and using the Quadrics network, experimental results are recorded to verify expectations and validate design choices.

1.6 Thesis Overview

The rest of this thesis is organized as follows. Chapter 2 outlines relevant characteristics and terminology of the Quadrics network and presents the motivation for NIC-based collectives. Chapter 3 describes design issues faced when implementing NIC-based reductions and solutions to work around them. Chapter 4 presents the model and analysis, while Chapter 5 provides experimental results. Finally, concluding remarks, related research, and ideas for future work is discussed in Chapter 6.

CHAPTER 2

BACKGROUND & MOTIVATION

This chapter presents a brief overview of the interconnect used to implement NIC-based reductions, the motivation behind their development, a background of related research, and the contributions of this work.

2.1 The Network – The Quadrics Interconnect

The NIC-based reduction algorithms were designed and implemented in the context of the Quadrics network, a modern cluster interconnect technology [23]. Quadrics is based on two building blocks: a programmable network interface card called the Elan [25, 26] and a low-latency, high-bandwidth communication switch called the Elite [27]. This section briefly describes these components.

2.1.1 The Network Interface Card – The Elan

The Elan resides on the PCI-bus and interfaces a processing node, containing one or more host processors, to the network. The Elan itself has respectable processing capabilities. It provides a user-programmable, multi-threaded, 32-bit, 100 MHz RISC-based processor; supported with a 64 MB bank of local SDRAM memory, along with an MMU and other sophisticated processing features. All of this hardware is available

to the NIC to aid the implementation of higher-level message processing protocols without requiring assistance from the host processor. In order to better support this usage model, the processor's instruction set includes specialized instructions to construct network packets, manipulate events, and schedule threads.

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 RDMA. In Quadrics, RDMA operations can access either host- or NIC-level memory.

2.1.2 The Network Switch – The Elite

The underlying network is circuit-switched and uses source-based, wormhole routing. It consists of Elite switches interconnected in a fat-tree topology [21]. Each Elite provides the following features: 8 bidirectional links each with a raw bandwidth of 400 MB/s (325 MB/s at the MPI-level), a full crossbar switch with a low 35 ns cut-through latency, and hardware support for collective communication including barriers and broadcasts.

2.2 Motivation for NIC-based Collectives

NIC-based reduction implementations avoid significant inefficiencies of current cluster systems. By eliminating inefficiencies between the processors and the network, as well as, between the processors and the operating system, NIC-based reductions can complete with reduced latency on a more consistent basis than host-based implementations. In fact, these benefits are not limited to reduction operations, and this section describes how NIC-based collectives, in general, attain such gains.

2.2.1 Elimination of PCI-bus Trasactions

NIC-based collectives can be completed significantly faster than host-based versions on fast networks by avoiding inefficiencies between the processors and the network. Modern cluster interconnects, like Quadrics, support very low message latencies; so low in fact, that PCI-bus transaction time is significant compared to the latency between nodes. By implementing collective communications in the NIC, as opposed to the host, many extraneous PCI-bus transactions can be eliminated. This can substantially reduce the total operational latency.

Collective communications, by their very nature, require a series of related messages to be exchanged between nodes involved in the operation. In host-based implementations, the host processor explicitly handles each of these messages. In order to do so, each message must be relayed back and forth between the host processor and the network via PCI-bus transactions. NIC-based implementations, on the other hand, handle messages immediately at the NIC, avoiding most of these transactions through the PCI-bus. In fact, NIC-based implementations suffer from such costs only

while initiating and terminating the operation. Collectives involving many nodes entail many messages, which implies that NIC-based collectives can scale substantially better than host-based versions as the size of the cluster increases.

Many researchers have recently utilized this advantage to implement NIC-based collectives [6, 9, 10, 11, 15, 17, 20, 31, 33] on other networks. This thesis further investigates how this advantage extends to the realm of reductions on Quadrics.

2.2.2 Avoidance of Host-level Process Interference

NIC-based collectives can be completed dramatically faster and in a more consistent fashion than host-based versions on large-scale clusters by avoiding inefficiencies between the processors and the operating system. Process interference on the host processor can be a major problem on large clusters. To demonstrate this, observe Figure 2.1. This figure shows the host-based latencies measured for a barrier and a reduction when using both one and two processes per node. As the number of nodes is increased, note how, for each collective, the latency for the two-process case deviates drastically from the one-process case.

This result is surprising since the underlying implementation efficiently reduces the two process problem to the one process problem by first performing a local shared memory step. Since shared memory operations take place quite quickly compared to typical network latencies, and because all nodes perform this brief step in parallel, the added overhead should be both small and constant with the number of nodes. Hence, knowledge of just the underlying implementation details fails to account for the observed trend; more factors must be involved.

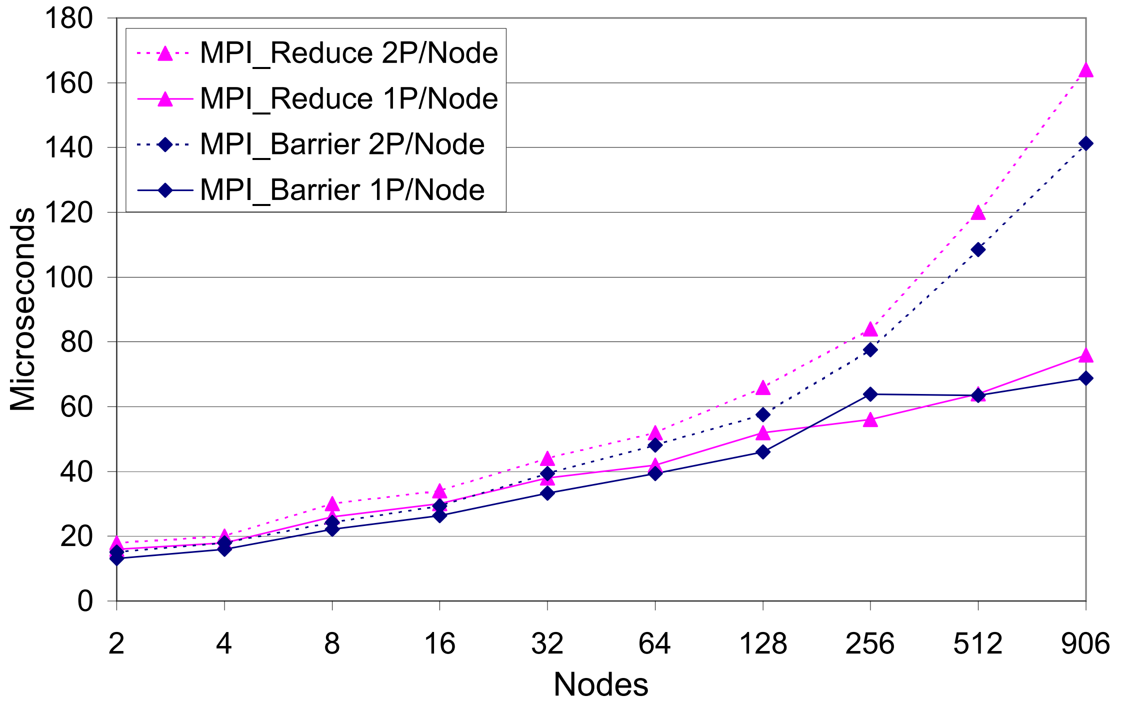


Figure 2.1: MPI Barrier and Reduce Latencies

In this cluster, there are two physical processors per node. When the application involves only one process per node, there is a spare processor on which the node may run various operating system threads. However, when both processors are used by the application, at least one of the application processes is forced to share its processor with the system threads. The ensuing process interference between the application processes and operating system turns out to be responsible for the drastic drop in performance [24].

Process interference leads to increased average latency on large-scale clusters. Many algorithms for collectives use communication structures that require messages to be passed through a chain of nodes during each invocation. Each intermediate node in the chain receives the message, processes it, and sends it on. The collective

does not complete until the message traverses the length of the chain. Due to interference with the operating system, application processes at intermediate nodes may be descheduled from the processor just before handling an incoming message. In this case, the collective will stall until the application process is rescheduled to handle the message. As the cluster size increases, existing communication chains grow longer and additional chains are introduced, so that the number of intermediate nodes increases. This improves the chances for process interference, which in turn, increases the average latency of the collective.

In addition to increased average latency, process interference is a rather non-deterministic phenomenon, which leads to a large variance in latency from one collective invocation to another. Thus, the same process interference problem simultaneously increases average latency and decreases consistency. As process interference on the host processor is inherently a host-based problem, NIC-based implementations may avoid it altogether. As a result, NIC-based collectives can complete with drastically better latency and in a more consistent fashion than host-based versions on large-scale clusters.

This represents a newly discovered advantage for NIC-based collectives. This thesis investigates its value in the context of NIC-based reductions.

2.3 Related Work

Huang and McKinley were perhaps the first to realize the potential of NIC-based collectives [17]. They examined the benefits gained by implementing broadcast and barrier operations on Asynchronous Transfer Mode (ATM) network adapters to avoid the excessive processing overhead incurred throughout the protocol stack. In order to develop implementations portable across a variety of ATM hardware, they placed rigid restrictions on the processing and memory requirements of their algorithms so that even the most limited ATM devices could adequately support them. Namely, they designed algorithms which were table-driven and performed only a small number of arithmetic and logical operations while using just a few scalar variables. In addition, they considered only static communication tree structures. Yet even with such limitations, they showed that certain NIC-based collectives scaled substantially better than host-based versions due to significantly reduced message processing (software) overhead.

While the advent of zero-copy, user-level protocols lessened the dramatic improvement shown in the above work, modern cluster interconnects, such as Quadrics and Myrinet, have reduced wire and switch latencies to the point where the cost of a PCI-bus transaction is significant. Simultaneously, the processing capability and memory available on the network interface cards have increased. Thus, the concept of NIC-based collectives remains a hot topic and researchers continue to investigate more complicated collectives and algorithms.

Many researchers have considered NIC-based multicast algorithms [6, 11, 15, 17, 20, 31, 33]. Multicast can be used as a building block to implement other collectives,

such as broadcasts or barriers, and thus an efficient multicast implementation is desirable. In addition, the problem is rather rich since the message size and destination set can be different with each invocation, and often one must design flow control and acknowledgment collection schemes for reliability. Each of the numerous publications put forth demonstrates a different approach, all of which have found success.

The work most closely aligned with this thesis is that by Buntinas and Panda [10]. They investigated the potential of NIC-based reduction on clusters interconnected with Myrinet. In particular, they modified the network drivers to implement binary AND and OR operations, as well as, integer and floating-point addition on 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 shows performance gains in clusters as small as 8 nodes. Although they only used binomial trees, with each reduction invocation, the host processor passes an operation descriptor, including the list of communication partners, to the NIC so their implementation could be easily generalized to use other tree structures. However, they leave the investigation of other communication trees, as well as, larger reduction vectors for future work.

2.4 Thesis Contributions

This work serves two purposes. In part, this thesis picks up where the previous work left off. The effect of using different tree structures and various vector sizes for an expanded set of reduction operations is investigated, as well as, an optimization for multi-element vectors. This work also proposes a parameterized model which can be accurately used to select the best available tree for a given instance of a reduction. In remainder, this work is the first to show the dramatically reduced latency and

increased consistency which NIC-based reductions may achieve through avoidance of host-level process interference on large-scale clusters.

2.5 Summary

First, this chapter introduced the components of the Quadrics network, a modern cluster interconnect which provides very low RDMA latencies, a programmable processor on the Elan NIC, and hardware support for collective communications through the Elite switch.

Then, motivating factors for implementing NIC-based collectives on networks like Quadrics were explained. Namely, NIC-based reductions improve large-scale cluster system performance by eliminating PCI-bus transactions between the processors and the network and avoiding interference with operating system processes on the host processor.

Related research about NIC-based collectives was discussed, and the contributions of the current work were listed.

CHAPTER 3

DESIGN ISSUES & SOLUTIONS

Previous research with NIC-based barriers, broadcasts, and multicasts has delivered good results [6, 9, 11, 15, 17, 20, 31, 33]. The success obtained by this previous work with simpler NIC-based collectives provides inspiration for investigating more complicated cases like reductions. The design goals of this work are 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. The design and implemention of NIC-based reduction is non-trivial, and this chapter presents the design issues faced and the solutions available to overcome them.

3.1 NIC Processor Capability

In meeting the above design goals, the first issue encountered is the lack of hardware support for floating-point operations on the Quadrics Elan processor. The NIC processor offers only integer instructions, so floating-point operations must be emulated in software.

To efficiently emulate floating-point operations that meet all of the various representations, rounding methods, and exceptions standardized in the IEEE 754 floating-point specification is a daunting task in its own right. Fortunately, some have already undertaken this task and packaged their progress into sophisticated software libraries as a service for others. In particular, this problem was tackled using SoftFloat [16], a freely available, IEEE 754 compliant floating-point package written by John R. Hauser, after Juan Fernandez ported it to run on the Quadrics Elan processor.

While the SoftFloat library suffices, it may provide more functionality than actually required. For instance, since the design goals do not include floating-point multiplication, it is unnecessary to support this operation in the library. Similarly, the library supports 80- and 128-bit floating-point operations, which are not used. Additionally, given a particular host architecture, it is likely that some rounding methods can be discarded. Supporting such unnecessary functionality may add overhead to the library, in which case, improvements could be made by using more specialized alternatives. Having noted this, improving the floating-point emulation software is out of the scope of this thesis and is left for future research.

3.2 NIC Processor Speed

The biggest obstacle faced in designing NIC-based reductions is the speed of the NIC processor. Simple comparison of the clock speed of the Elan processor, at one-hundred megahertz, to that of typical host processors, somewhere in the gigahertz range, immediately declares an order of magnitude difference in processor speed. The gap grows much wider when dealing with floating-point operations which must be

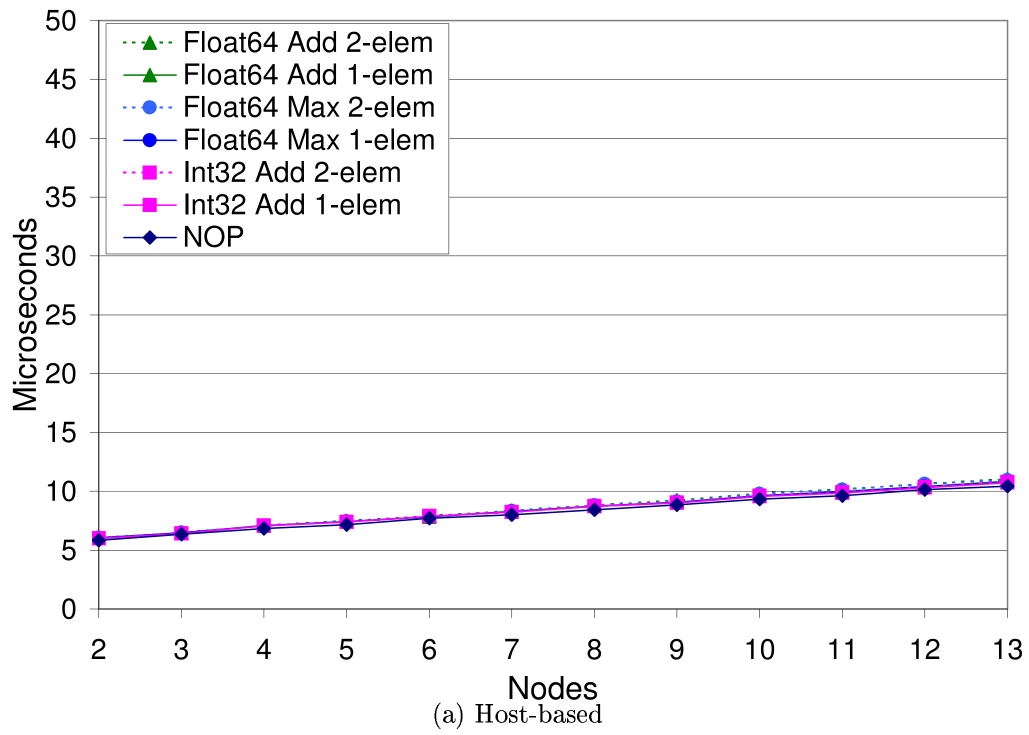
emulated on the Elan. This section provides some quantitative idea of the difference in processor speeds along with constraints and design choices to deal with it.

3.2.1 Serial Reduction

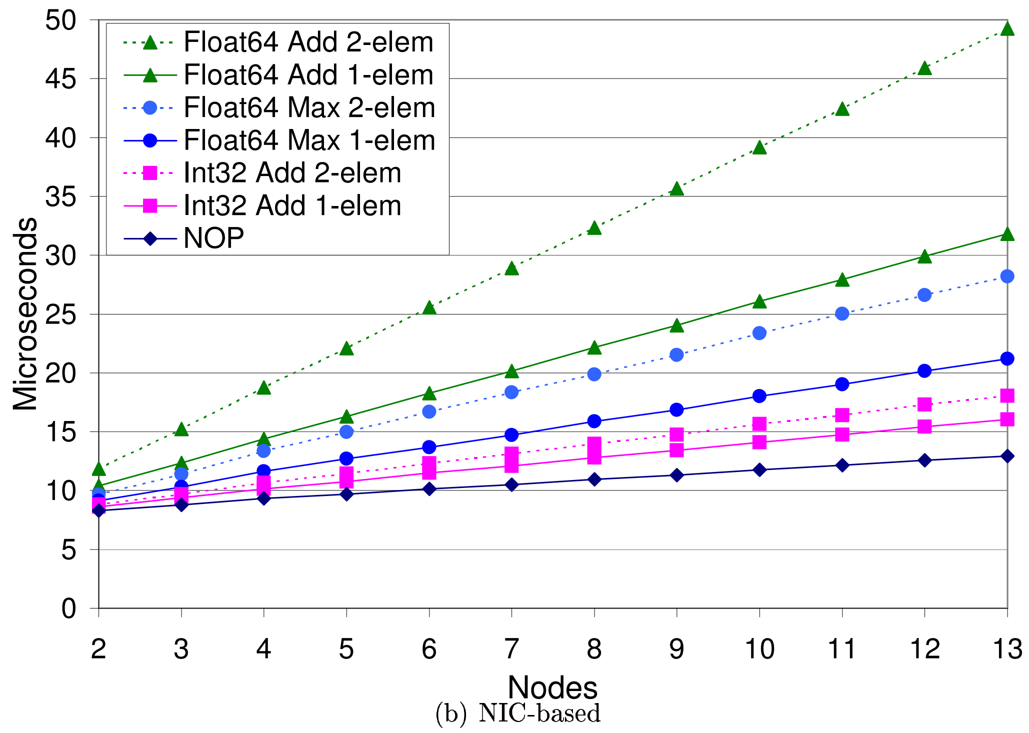
A simplistic reduce algorithm, refered to as *serial reduction*, was implemented for investigation of the communication and computation characteristics of the Elan and the host processors. In this algorithm, once the root of the reduce is designated, the non-root nodes simultaneously send their data 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. Finally, the root uses the hardware-based broadcast mechanism to synchronize the group and signal the completion of the operation.

Serial reduction tests involving 2-13 nodes for various reduction operations and data sizes produced Figure 3.1. Figure 3.1(a) shows the host-based serial reduction latencies, while Figure 3.1(b) shows the NIC-based times. Each figure provides the latencies of 32-bit integer addition, 64-bit floating-point maximum, and 64-bit floating-point addition operations for both one- and two-element vectors. Also shown is a NOP operation which consumes no computation time. Since the latency for a NOP serial reduction consists purely of communication costs, it represents the lower bound for the algorithm with respect to computation.

In comparing these two figures, it is immediately obvious that NIC-based reductions depend significantly on both the reduction operation and the reduction vector size, while host-based reductions are largely independent of both. Each curve in the host-based graph lies on or just above the NOP curve, implying that computation costs are insignificant when compared to the communicaiton costs. The NIC-based



(a) Host-based



(b) NIC-based

Figure 3.1: Serial Reduction Latencies

latencies, on the other hand, tell a different story. Complicated operations scale considerably worse than simpler ones: compare floating-point addition to integer addition. Also, even fast operations are rather sensitive to small changes in data size: observe integer addition for one- and two-element vectors.

These observations put the difference in processor speeds in clear perspective. That the NIC processor is one or two orders of magnitude slower than the host processor was a known fact, but now it is also clear that this difference in computation costs is substantial compared to communication costs. Thus, while efficient host-based reductions may be designed considering only communication costs, desiging NIC-based reductions is more complicated since computation costs must also be considered. In fact, since computation costs are so expensive for NIC-based reductions, they will execute with reasonably low latency only for simple operations and small data sizes. For these reasons, the slow NIC processor is the toughest issue to be dealt with when designing NIC-based reductions.

3.2.2 Simple Operations and Small Data Sizes

From the previous section, it is apparent that NIC-based reductions will perform well only for simple operations and small data sizes. The slow NIC processor becomes overwhelmed if given anything more complicated. This is a stringent restriction on the class of problems where NIC-based implementations may be valuable, however, a large majority of the problems posed by practical programs falls within this class.

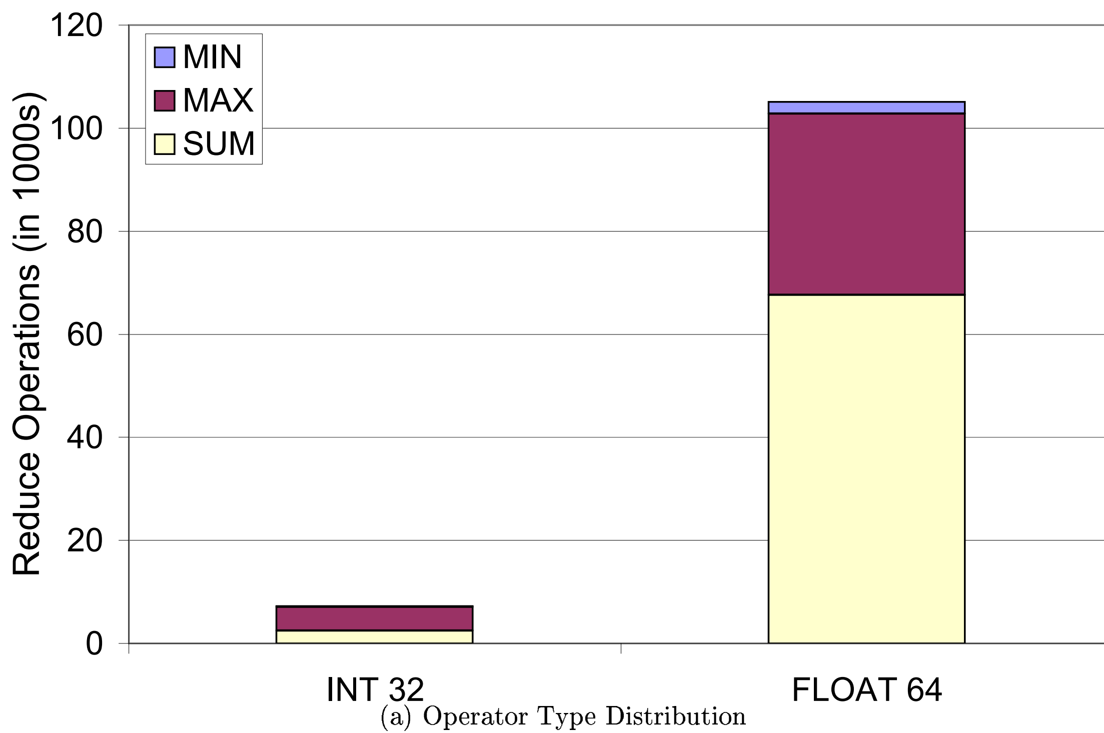
Reductions involving simple operations on small data sizes are the common case in many scientific applications. Researchers have verified this claim across a collection of large-scale scientific programs that represent a range of application domains and

programming languages [32]. That collection includes linear systems solvers and simulators for gas dynamics, particle and photon transport, and shockwave analysis. The applications are written in a variety of languages including Fortran, C, and C++. In further support of this, the MPI allreduce operations performed during the execution of SAGE [19] are profiled in Figure 3.2. SAGE is a sophisticated hydrodynamics simulation program used within Los Alamos National Laboratory to model, among other things, energy coupling and shockwaves travelling through ground, water, and air. It is representative of typical scientific applications running on large-scale parallel machines.

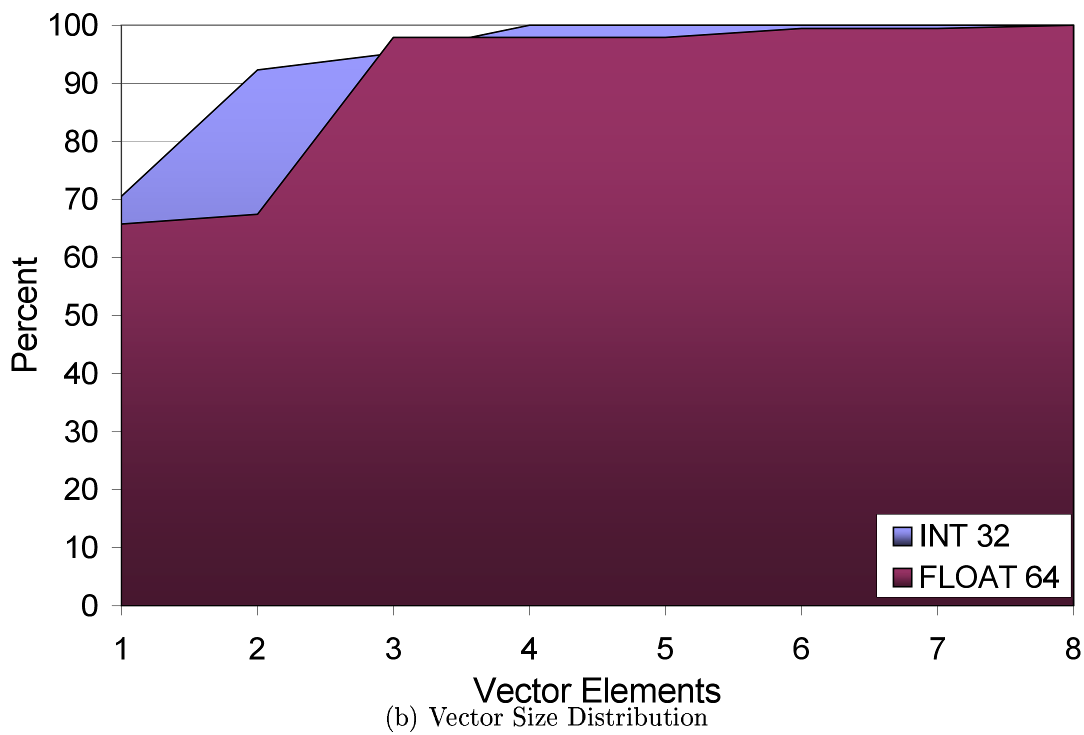
Figure 3.2(a) shows the distribution of reduction operator types. Note that only two simple data types are used by SAGE: 32-bit integers and 64-bit floating-point numbers. Additionally, only a few simple types of operations are used: minimum, maximum, and summation. Typical reduction operations thus require limited processing.

Equally important is Figure 3.2(b), which shows the cumulative distribution of the data sizes for both integer and floating-point data types. Direct observation makes a striking point: 95% of all reductions invoked by SAGE use 3 or fewer elements and 100% use 8 or fewer.

Observations from these two figures are key. Together, they imply that typical reductions involve simple operations on small vectors, which so happens to be the same class of reductions for which one may benefit from NIC-based implementations. In other words, NIC-based reduction implementations benefit the common case the most.



(a) Operator Type Distribution



(b) Vector Size Distribution

Figure 3.2: Profile of MPI_Allreduce Operations in SAGE

3.2.3 f-nomial Trees – Generalized Binomial Trees

Communication structures in efficient reduction algorithms tend to balance message processing time with message latency. The results from the serial reduction tests demonstrate that message processing time in host-based reductions is completely determined by the communication characteristics, while it is dependent on the communication characteristics, the reduction operation, and the vector size in NIC-based reductions. Thus, while a single communication structure will suffice for efficient host-based reductions, the slow NIC processor demands a range of communication structures to support efficient NIC-based reductions.

For this work, f-nomial trees (a.k.a. k-nomial trees) were chosen since they provide such a range of structures by generalizing a well-known reduction algorithm, binomial trees. Binomial trees are commonly used in reduction algorithms because they offer 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 communication 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. This property is especially useful for NIC-based reductions, where the computation costs incurred by the slow processor may change message processing time substantially depending on the operation being performed and the amount of data being processed.

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 f-nomial trees are described starting from a quick review of the operation of binomial trees, from which the generalization is trivial. Also, although reduction trees collapse to the root node, it is easier to describe the structure of a tree as it expands. For convenience then, say 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 that holds a copy of the broadcast message at the start of the phase sends to another node which doesn't. In this way, the number of nodes that hold a copy of the message doubles at the end of each phase. 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 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. For instance, during the first phase, the root sends to $(f-1)$ children, so that by the end of the first phase the message has spread from the root, 1 node, to the root and its $(f-1)$ children, a total of $1+(f-1)=f$ 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 $f+f(f-1)=f(1+(f-1))=f^2$ nodes. Similarly, in the third phase, each of these f^2 nodes become a parent and each sends to $(f-1)$ children who have yet to receive the message, so that by the end of the

third phase, the message spreads to a total of f^3 nodes, and so on. Thus now, 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 used; only remember, the tree collapses rather than expands since a reduction is to be implemented rather than a broadcast. For reference, Appendix A provides psuedo/C code that drives a reduction algorithm using f-nomial tree communication structures.

As a concrete description of an f-nomial reduction, consider Figure 3.3, which shows a graph representing a 4-nomial tree overlaid atop 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, 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. To be precise, the number of nodes containing data relevant to the reduction has been cut by a factor of four, from 16 to 4. The algorithm completes after the second phase again cuts the number of nodes by a factor of four, from 4 to 1, when node 0 receives and reduces the partial results from the three, now child, nodes 4, 8, and 12. Thus, in two communication rounds, 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 3.4 shows f-nomial trees

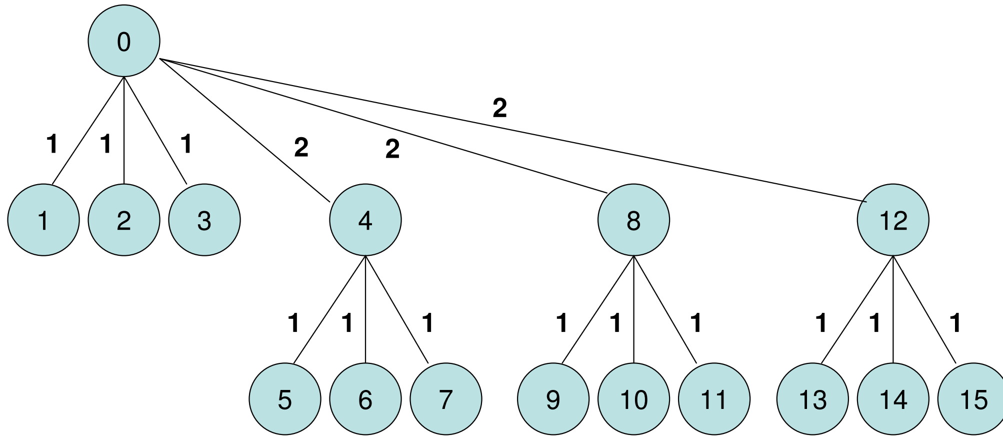


Figure 3.3: 4-nomial Reduction over 16 Nodes

of various degrees, all which cover 16 nodes. This flexibility allows one to trade off between communication and computation costs, choosing an appropriate mix for a given problem. 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. Efficient algorithms will tend to balance the costs of communication and computation. Communication bound reductions will favor wide trees to minimize the number of tree levels, and hence, the number of communication phases. Computation 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. Chapter 4 illustrates how to choose the optimal degree analytically.

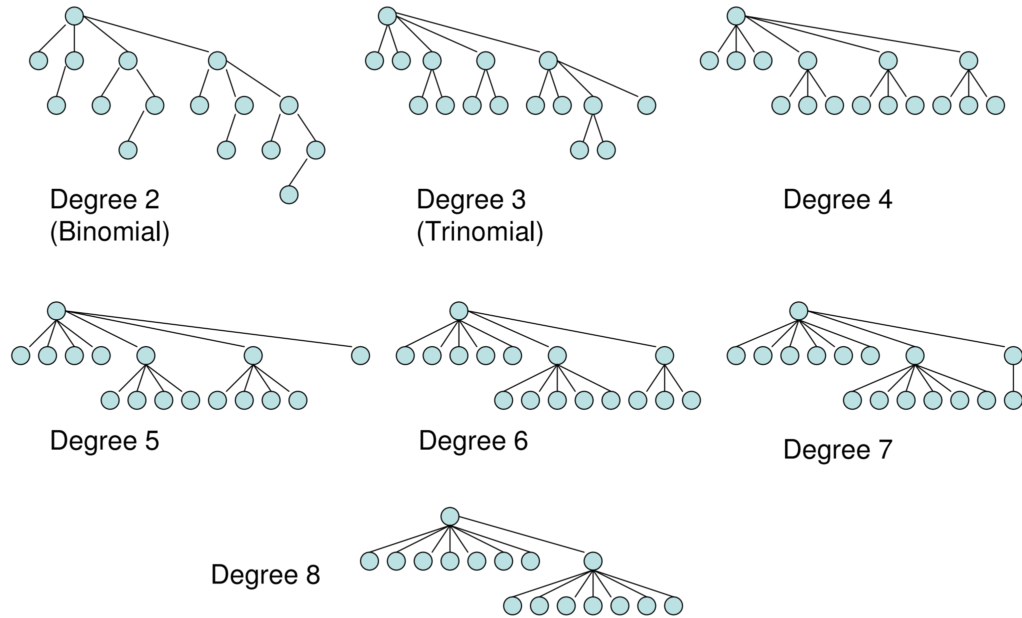


Figure 3.4: f-nomial Trees of Varying Degrees covering 16 Nodes

3.2.4 Vector Split Optimization

Since NIC processors are very slow, it helps to keep as many of them working as possible to maximize utilization of their limited processing power. Often times, it is worth suffering a little extra communication cost in return for a substantial reduction in computation cost. In other words, computationally intensive NIC-based reductions should be highly parallelized.

For multi-element vectors, parallelism can be increased through an optimization proposed by Van de Geijn [30]. Basically, the idea is to split the vector and assign the different pieces to distinct groups of nodes. The groups then reduce the vector pieces in parallel and recombine the reduced pieces in the last step to form the reduced vector. In other words, presented with this optimization, there are two options available to reduce multi-element vectors: 1) reduce one large vector serially through a single

tall tree, or 2) distribute and reduce smaller pieces of the vector in parallel through shorter trees, incurring the added overhead of splitting and recombining. The second approach suffers from extra communication while distributing and recombining the vector pieces, however, if computation is expensive, significantly savings are gained by processing smaller amounts of data during each phase of the tree. For tall trees, which require many phases, this savings can amount to a lot.

As an example, which is diagrammed in Figure 3.5, this optimization is used to reduce a two-element vector over 8 nodes. Here, the vector elements are shown as small rectangles located adjacent to circles representing the nodes on which they reside. As shown in the left section of the figure, the group of 8 nodes is first split into two groups of 4, as represented with the dotted line bisecting the circles. The top element of the vector is then assigned to the top group of 4 nodes and the bottom element to the bottom group. This is accomplished once nodes in the two groups send the appropriate element to a partner in the opposite group, as represented by the arrows, and reduce the received data with their copy of the corresponding element. At this point, the two-element vector has been split among the two groups. 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 group-wise reductions on the element assigned to them, represented with the dotted boxes 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 completes the reduction using the vector split optimization.

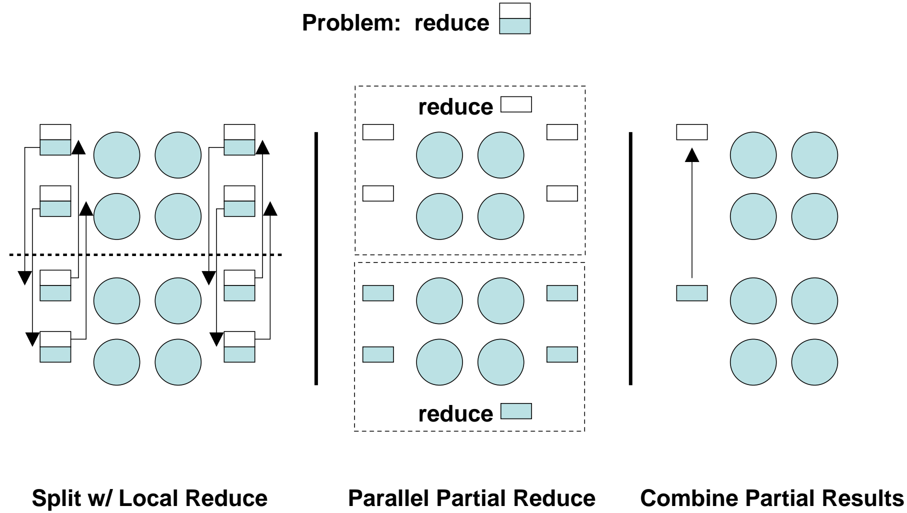


Figure 3.5: Vector Split Optimization

This optimization was added to the basic f-nomial algorithm to create a new algorithm named *f-nomial split*. During 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. As discussed, these partial reductions occur in parallel across the multiple trees. 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 5.2.1. Basically, it allows NIC-based reductions to scale substantially better than they otherwise would have for larger vector sizes.

3.3 Host-NIC Synchronization Overhead

In order for the host processor to delegate a collective operation, such as a reduction, to the NIC, several administrative tasks must be performed. These tasks include writing the application data to and reading the final result from NIC memory; informing the NIC processor of what operation to do, what data type to use, and the number of vector elements to process; and providing the NIC with a list of communication partners and internal communication and processing buffers to use. Additionally, the host must instruct the NIC when to start the operation, and the NIC must notify the host of the operation's completion. Since these tasks act to synchronize the host and NIC processors, they are refered to collectively as *host-NIC synchronization*.

Host-NIC synchronization introduces a certain amount of overhead. In some cases the overhead may be tolerable, however more often, one would like to avoid it. To do this, there are two options available: minimize the overhead directly and/or hide it behind other latencies.

3.3.1 Minimizing the Overhead

To minimize host-NIC synchronization overhead, one can improve the manner in which information is transferred between the host and NIC. For instance, the data structures which describe the operation to the NIC should be well-designed. In practice, only a small amount of information needs to be transferred between the host and the NIC, however if not done carefully, this transfer can be costly. There are several distinct data items which must be transferred to the NIC, some of which include lists of items themselves. If each of these items is written to the NIC

individually, the expensive PCI-bus transaction start-up costs will mount up. Thus, one should design the data structures in a way so that they can be copied to the NIC in block-copy fashion or through DMA transfers.

It is also possible to reduce the amount of data that must be written to the NIC. In some cases, it may be possible to refer to a collection of data items through a single index parameter. For example, the internal communication and processing buffers can be grouped into "channels". This enables the host processor to refer to an entire set of buffers through a single channel number. One could likely do something similar with the communication data structures. Many applications will likely use a limited set of distinct communication structures, and since the NIC provides a large amount of memory, it seems as though a caching system may be useful. This would allow the host processor to recycle data structures previously copied to the NIC by referring to them with a simple cache-line number.

In other cases, different data items can be nicely consolidated. Many of the data items may have small bounds on their value. For example, 8 or 16 different channels and 16 or 32 cache slots may be more than enough. In this case, one could pack data items such as the channel number and cache-line number with other fields like those listing the collective, e.g. reduce or allreduce, the reduction operation, e.g. 32-bit integer addition, 64-bit floating-point maximum, etc., and the data size into a single 32-bit value using bit masks.

This work used a single channel and assumed cached communication structures. The message size was transfered as a separate item, however, the collective and reduction operation fields were consolidated into a single value. The time required to pack

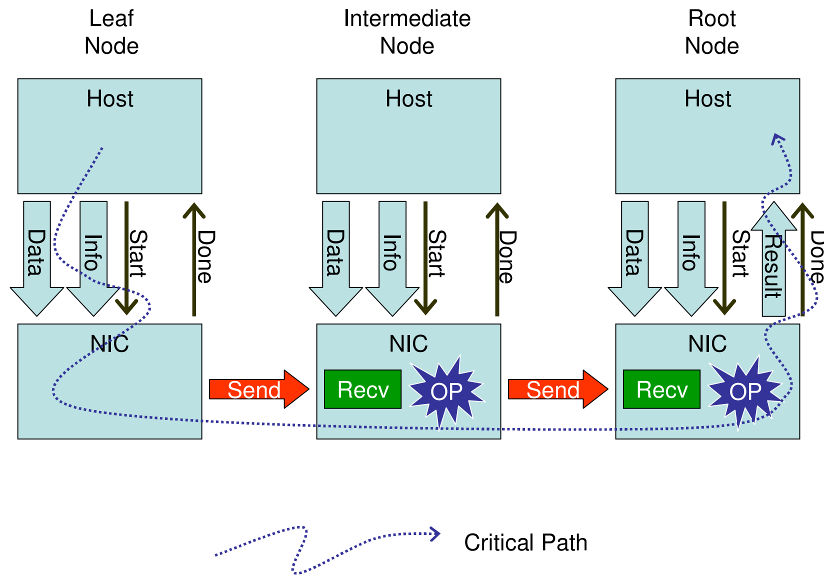
and unpack the data is less than the time required to transfer each item individually across the PCI-bus.

3.3.2 Hiding the Overhead

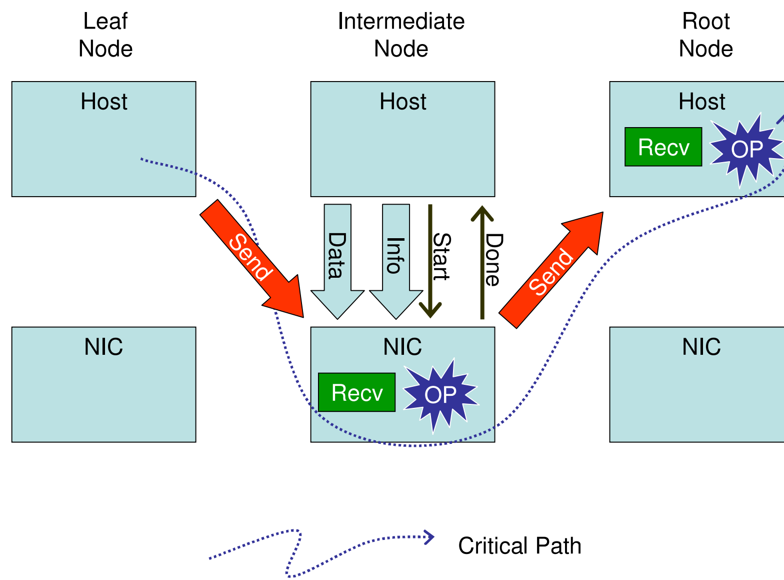
Additionally, the host-NIC synchronization overhead can be hidden behind other latencies. This can be accomplished by using the NIC-based implementation only at the intermediate nodes, where the host-NIC synchronization can be largely hidden behind latencies in the initial and final phases of the algorithm. This approach is illustrated in Figure 3.6.

Figure 3.6 is composed of two subfigures. Each subfigure shows the critical path of some reduction tree as a dashed arrow winding its way through three nodes, each of which is depicted with separate host and NIC levels. A leaf node is shown on the left; its parent, an intermediate node is in the middle; and the reduction root is located on the right. The two subfigures show how the critical path changes under two implementations.

Figure 3.6(a) shows the straight-forward NIC-based implementation in which all nodes use the NIC. To begin, all nodes simultaneously initiate the NIC. In doing so, as represented by the downward pointing arrows, the application data and reduction information are first transfered to NIC memory, and then, the NIC is signaled to start the reduction operation. Following this initial part of the host-NIC synchronization, the NIC on the leaf node immediately relays its data to the intermediate node for reduction processing. The NIC at the intermediate node receives the leaf data, performs the reduction operation, and sends the result to the root node. The NIC at the root similarly receives and reduces the data from the intermediate node to obtain



(a) Incurring Overhead in the Critical Path



(b) Avoiding Overhead in the Critical Path

Figure 3.6: Host-NIC Synchronization Overhead

the final result. The latter part of the host-NIC synchronization is then performed as the result is copied up to its destination buffer and the NIC notifies the host processor that the result is ready. Note that in this straight-forward implementation, the host-NIC synchronization overhead is included in the critical path of the reduction latency.

Figure 3.6(b) shows a modified implementation in which just the intermediate node uses the NIC. In this case, the leaf node sends its data to the intermediate node directly from host memory, bypassing the NIC. As the message traverses through the network, the intermediate node uses this time to initiate the NIC. As before, the NIC at the intermediate node then receives and reduces the leaf data. However this time, when the result is sent to the root, it is delivered to the host rather than the NIC. The host at the root then receives and reduces the data, placing the final result in the destination buffer. Note that this modified implementation overlaps the host-NIC synchronization with the message latencies incurred in the initial and final communication phases, thus hiding it from the critical path.

This work implements both methods, and the two are compared in the experimental section.

3.4 Summary

This chapter discussed the design issues and available solutions to be considered when implementing NIC-based reductions. The major issues include performing reduction processing on the less functional, slower NIC processor and dealing with the overhead associated with initializing and finalizing a NIC-based reduction.

NIC processors typically provide limited functionality in hardware, however software may be used as a substitute. For instance, the Quadrics Elan processor does not provide hardware support for floating-point operations, however, software libraries which emulate floating-point operations using integer instructions are available.

NIC processors run much slower than those at the host, often by an order of magnitude or more. Fortunately, reductions used in typical programs often require minimal processing, so even while the NIC processor is very slow, it is powerful enough for the common case. Given a slow NIC processor connected to a very fast network, computation costs are typically comparable to communication costs. NIC-based reductions thus require a range of communication structures which may be used to balance message processing time against message latency depending on the data size and operation to be performed. Also, mechanisms, like the vector split optimization, which increase the degree of parallel-processing and better utilize the limited power of NIC processors are valuable.

Host-NIC synchronization costs may be expensive, but it can also be minimized and/or hidden. Techniques that minimize the amount of reduction information that the host must write to the NIC, like using channels of communication buffers, caching communication structures, and consolidating data items into packed values, act to reduce host-NIC synchronization overhead. Additionally, employing NIC-based reduction at only the intermediate nodes of a communication structure allows one to hide host-NIC snchronization cost behind other latencies.

The value of many of these design choices will be validated in the following chapters.

CHAPTER 4

ANALYTICAL MODELS

In this chapter, details in the design of efficient NIC-based reductions are addressed using analytical models. Simple model parameters are first introduced and are then used to provide 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.

4.1 The Model Parameters

Observations of the serial reduction data, as shown previously in Figure 3.1, suggest a simple parametric model. Namely, it is difficult to overlook the sharp linear trend that relates the reduction latency to the number of nodes involved. Using just the slope and intercept, such a tight trend provides a very simple but accurate analytical model to estimate the serial reduction latency. Furthermore, the serial reduction algorithm will form the basic building block of more sophisticated tree-based algorithms. Given an accurate model for the building blocks, one can piece together a model for more sophisticated algorithms. In other words, the slope and intercept of the serial reduction latency curves are sufficient to quite accurately predict the performance of any other proposed algorithm.

Parameter	**Meaning**
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
C(OP)	constant due to initial overhead, in general dependent on the operation OP

Table 4.1: Model Parameters

Moving along this direction, it is instructive to define the slope and intercept in terms of more meaningful parameters. To account for the linear trend, 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 order. Since the nodes send to the root simultaneously, all messages worm their way to the root in parallel. Hence, regardless of the number of nodes, 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, the model parameters are defined in Table 4.1. Throughout the rest of this thesis, the functional parameters M (message size) and OP (reduce operation) will typicailly be suppressed from the various terms.

Essentially, this model modifies LogP [14] to better serve 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 account for receive overhead in this work. 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, $c(M, OP)$. These modifications were made since conceivably, the message latency, the reception costs, and the reduction costs may all differ between host-based and NIC-based implementations, and 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.

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

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

This expression is shown pictorially in Figure 4.1, which presents a timeline depicting the time 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 cases this work is interested in. This is because the focus is on reductions involving vector sizes of only a few elements, say up to eight, which typically fit into a single 64-byte fixed-length packet on the Quadrics network. Thus, whether the problem involves one-element vectors or eight-element vectors, the receive time is the same, i.e. the cost to receive one 64-byte packet.

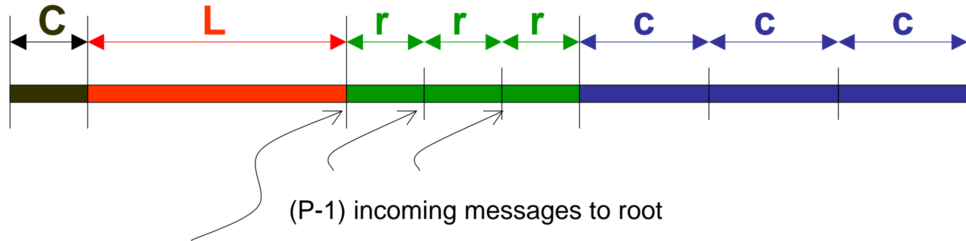


Figure 4.1: Model of Serial Reduction Latency

To provide some context of typical model parameter values, communication and initialization values are given in Table 4.2 and computation values are listed in Table 4.3.

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 constant C is much more for NIC-based reductions. This difference represents the host-NIC synchronization costs. Finally, the computation costs are much, much higher for the NIC-based reductions. The NIC processor is substantially slower than the host. In fact, in some cases the NIC processor falls behind by nearly two orders of magnitude.

The parameter values also suggest the general form of efficient algorithms. One may note that for small messages, the latency, L, is often significantly more than the receive time, r. This is relevant considering the circuit-switched nature of the

Parameter	Value
L	2.90
r	0.42
C	2.70

(a) Host-based

Parameter	Value
L	2.10
r	0.42
C	9.20

(b) NIC-based

Table 4.2: Communication and Initialization Parameter Values (μs)

Operation	1-elem	2-elem	4-elem	8-elem
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	1-elem	2-elem	4-elem	8-elem
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 4.3: Computation Parameter Values (μs)

Quadrics network as the sender may only send a message every L units of time, while the receiver can receive one in every $r \ll L$ units. As a result of this asymmetry, nodes in efficient algorithms will tend to receive more often than they send, which leads to tree-shaped communication structures. Foresight of this fact explains why a tree-based algorithm, like f-nomial trees, was chosen instead of a more symmetrical type of algorithm, like pair-wise exchange.

4.2 Modeling f-nomial Trees

One would like to be able to choose the best f-nomial tree for a given problem through analytical methods, so in this section, the model is applied to the algorithm.

The model is first applied to full f-nomial trees, in which each parent has a complete set of children during each phase. This simplifies the process and provides a clear, concise expression. This expression is then manipulated analytically to arrive at the condition which gives the best degree. Finally, the model is applied to the more general case of an f-nomial trees consisting of an arbitrary number of nodes.

4.2.1 Full f-nomial Trees

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 $T_{serial}(f)$ derived from the previous section and adjusting for initial overhead, one arrives at the following expression as a quick analysis of the f-nomial reduction latency:

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

An example application of the model to intermediate phases is shown pictorially in Figure 4.2. In this figure, the two horizontal timelines 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 a length of time L, the two parent nodes each receive and reduce the data from their $(f-1)$ children of the first phase. Starting

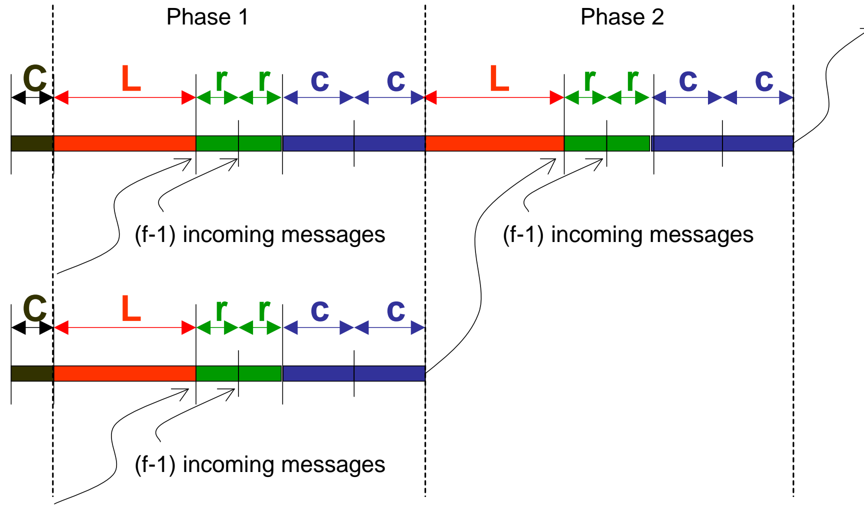


Figure 4.2: Model of f-nomial Reduction Latency

the second phase, the bottom node, now a child to the top node, immediately sends its partial result to its parent. Again, after a length of time L, the top node receives and reduces the data from its $(f-1)$ children of the second phase. The reduction continues off the diagram 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.

4.2.2 Finding the Optimal Degree

Given the model for $T^{full}_{fnomial}(P, f)$, it is straight-forward to compute the optimal degree f to use for a particular problem. One seeks to find that value of f which minimizes the reduction latency. To do so, one simply takes the derivative of $T^{full}_{fnomial}(P, f)$ with respect to f and finds the condition which sets the result equal to zero. This section details this process.

Basically, the goal is to find that value of f which minimizes the following expression:

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

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

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

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

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

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

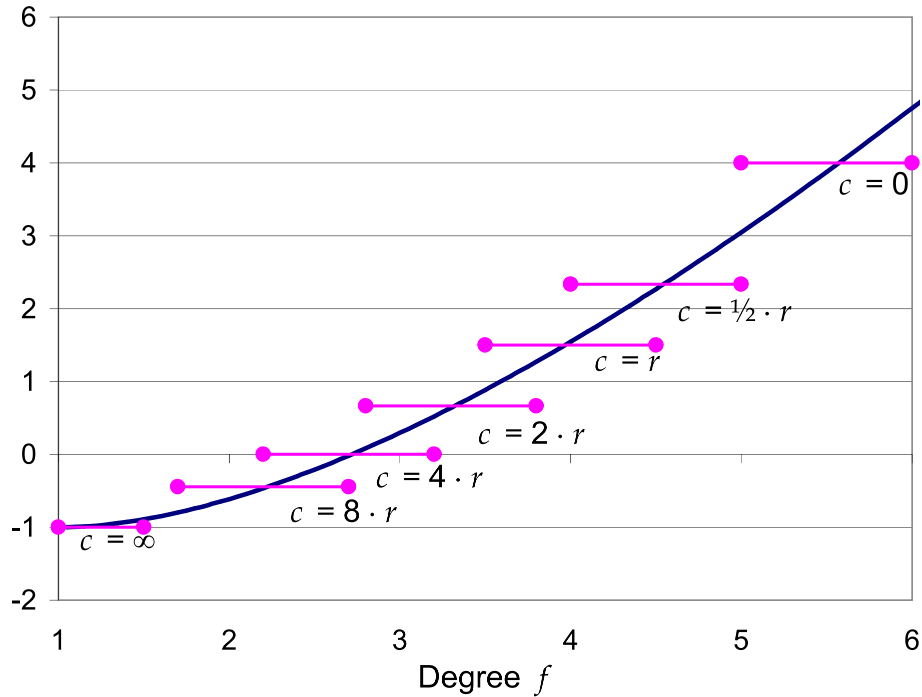


Figure 4.3: Plot of $f \cdot (\ln f - 1)$ and $L/(r + c) - 1$ for $L = 2.10$ μs, $r = 0.42$ μs, and various c

The above expression gives the best value of f to use with L, r, and c. It is a transcendental equation, so one must solve for f numerically by finding the intersection of $f \cdot (\ln f - 1)$ with the function $L/(r + c) - 1$, which is constant once c is decided by the operation and data size of a particular problem. The intersection of these two functions is plotted for various values of c in Figure 4.3, after setting $L = 2.10$ μs and $r = 0.42$ μs, the values corresponding to NIC-based reduction in Section 4.1.

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 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. 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 some arbitrary number in this range, say $f = 1.5$, is selected and pluged back into $T^{full}_{fnomial}(P, f)$ one gets:

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

When compared to binomial trees, these values of f tend to construct trees which have more communication phases, since $\log_{1.5} P > \log_2 P$. They do so in return for a reduced amount of reception and computation costs, $0.5 \cdot (r + c)$ instead of $(r + c)$. Thus, trees in this range wish to suffer extra communication in order to save on computation, so this is the range in which optimizations like the vector split are valuable.

Even though the above analysis applies only to full f-nomial trees, it helps to build intuition and establishes reasonable expectations by using actual parameter values.

4.2.3 Refining the Model

Unfortunately, the simplistic expression for $T^{full}_{fnomial}(P, f)$ in the previous sections does not accurately account for trees with an arbitrary number of nodes. The previous expression was derived assuming a full tree, i.e. assuming $\log_f P$ is an integer. When the number of nodes is not an integer power of the degree 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,

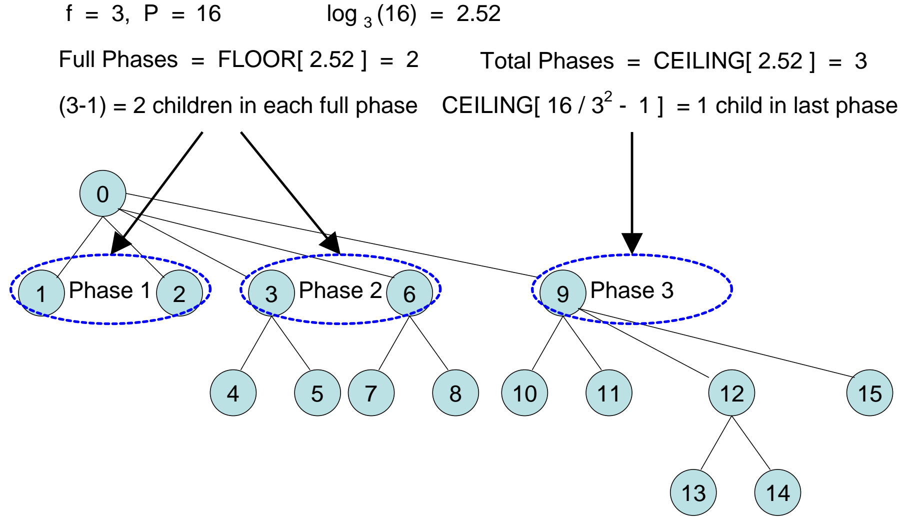


Figure 4.4: Application of Reduction Latency Model to 16-Node 3-nomial Tree

there will be fewer than the full set of $(f-1)$ messages to receive 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 involving a full set of $(f-1)$ children for the root. The L term 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 due to 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 4.4 demonstrates how the various terms refer to a 16-node, 3-nomial tree.

When using this expression for $T_{fnomial}(P, f)$, it is non-trivial to express the best degree f in terms of the other model parameters, as done before. However, in practice the best degree tends to be a small value, so one can simply cycle through a limited set of values and evaluate the expression numerically to find the best one. This approach is illustrated graphically in Section 5.2.3 when the model is validated.

4.3 Summary

This chapter presented a simple but accurate model which may be used to predict reduction latencies. Typical parameter values were provided for host-based and NIC-based reductions, which were used to compare and contrast the two. The model is then applied to f-nomial reductions and is used to find the best degree f for a given problem analytically. The model expressions involve transcendental equations and must be solved numerically, but the possible solution space is very small so that one may be found trivially through brute force.

CHAPTER 5

EXPERIMENTS

Various versions of the f-nomial algorithm were implemented for experimental purposes. Results from these tests are presented in this chapter to validate design choices and to illustrate the benefits of NIC-based reduction. The algorithms were developed and initial performance evalutaions 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 network. Scalability analysis was performed on the ASCI Linux Cluster (ALC) [28] located at Lawrence Livermore National Laboratory. The ALC uses 960 dual-processor nodes with 2.4 GHz Xeons and the Quadrics network.

5.1 Procedural Details

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

First, regardless of the number of host processes per node, each node implements the NIC-based reduction using a single thread running on a single NIC. Whenever

there are multiple host processes involved on a node, the host processor first reduces the local data vectors through shared memory before it initiates 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 to and from the host. However, if a collection of data (e.g. vectors for multiple local processes) is already located at the host, 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, just the locally reduced result rather than each of the local vectors.

Second, while the design goals of this work are to investigate both reduce and allreduce operations, only the results for reduce are presented. The approach taken was to focus on reduce, which is simpler to implement, model, and analyze. Admittedly, since allreduce is often used more frequently than reduce in parallel programs, its inspection is more relevant. However, the hardware-based broadcast provided by Quadrics simplifies things. A hardware-based broadcast, which scales very well (small and almost constant) as the number of nodes is increased, can be tacked on to the end of an efficient reduce operation to implement an efficient allreduce operation. Thus, since the observed reduce latencies can be extrapolated to estimate allreduce latencies with the addition of a small constant, the measurements for reduce are representative of what could be expected for allreduce.

Third, for testing purposes, a barrier was inserted between each of the reductions in order to serialize consecutive invocations. As Quadrics also provides a hardware-based barrier mechanism, such barriers keep the distributed nodes very tightly synchronized in real time. Although this synchronization is not required for reductions

and thus adds unncessary overhead to the latency, the measurement procedure is simplified since there is no need to worry about pipelining effects associated with nodes which escape ahead to start the next operation before the previous one has completed.

Fourth, unfortunately, f-nomial host-based reductions were not available at the time when an opportunity to run on the ALC presented itself. Host-based results for clusters up to 32 nodes used f-nomial reductions, however anything larger used the reduce collective in the provided production-level MPI library. The MPI implementation internally delegates the work to a reduction function, called elan_reduce(), supported in the lower-level Quadrics Elan library [25]. The Elan algorithm, in turn, performs a reduction via a 4-ary tree communication structure 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 really implements an allreduce operation, rather than the simpler reduce operation used in the NIC-based reduction. Even so, the tests remain fair since 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, there was a large variance in the measured reduction latency from one invocation to another, especially for host-based reductions. Unless otherwise stated, this variance was compenstated by reporting the average reduction latency: computed as the total time required to complete 100,000 iterations, divived by 100,000.

5.2 Validation

The experiments presented in this section were designed to illustrate the impact of design choices and to check the accuracy of the analytical model.

5.2.1 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 some extra 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 5.1, 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 the vector is broken into two pieces, two splits imply four pieces, three imply eight, 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 value of the vector split optimization for multi-element vectors is clear. After 3 recursive splits, the 8-element latency is improved by nearly a factor of three, while for 4 recursive splits, the 16-element case is over three 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-based implementation can provide. And interestingly, one may carefully note that the latency for a 2-element vector actually increases slightly

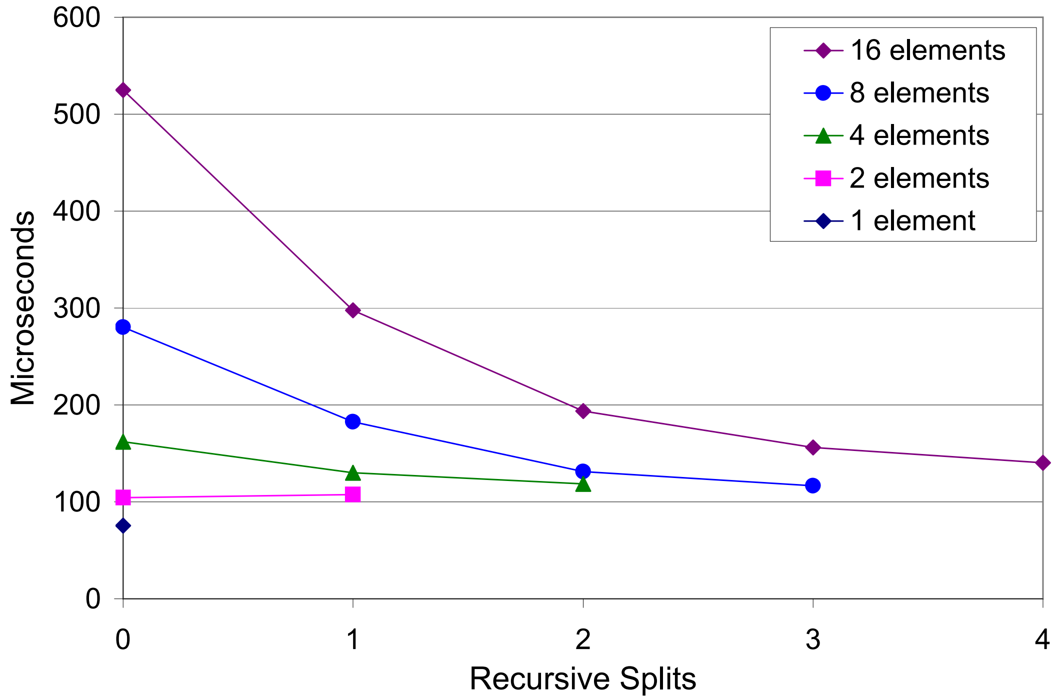


Figure 5.1: f-nomial Split on Various Vector Sizes for 64-bit Floating-Point Addition over 512 Nodes

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 cross-over point can be computed so as to always pick the better of the two options. Van de Geijn discusses the details in [30].

5.2.2 Validating the Host-NIC Synchronization Overhead

This section shows the impact of host-NIC synchronization overhead and the improvement available through implementation of the optimizations discussed in Section 3.3.

Figure 5.2 shows latencies for three different implementations of single-element 32-bit integer addition over binomial trees of various sizes. A host-based implementation is shown, along with a NIC-based implementation, and an optimized NIC-based

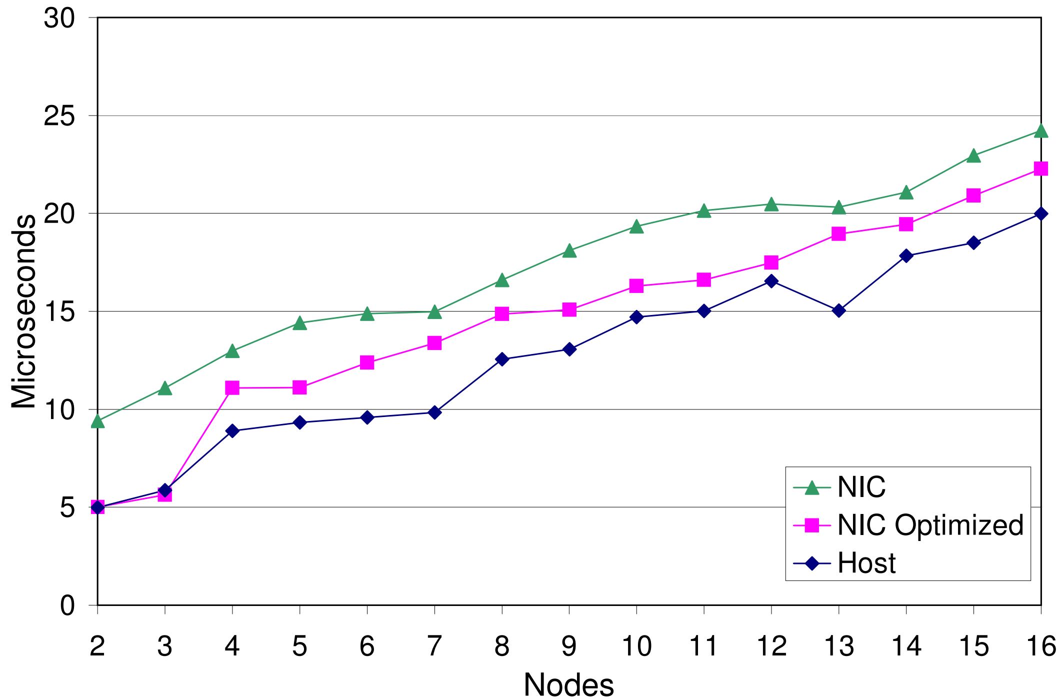


Figure 5.2: NIC-based and Host-based Latencies for Single-Element 32-bit Integer Addtion using Binomial Trees

implementation, which only employs the NIC for reduction processing at intermediate nodes.

The standard NIC-based implementation gains on the host-based version as the number of nodes is increased, however, there is a significant initial penalty to overcome before the NIC will actually outperform the host. In particular, when only two nodes are involved in the reduction, the host is about twice as fast at the NIC, which lags behind by about 5 μs. This extra cost comes from the host-NIC synchronization overhead.

By using the NICs at just the intermediate nodes, the optimized NIC-based reduction is able to reduce a significant portion of the overhead. For two and threes nodes, the optimized NIC-based reduction is actually equivalent to the host-based version

and incurs no penalty from host-NIC synchronization, since there are no intermediate nodes in binomial trees of these sizes. For larger trees, which involve some intermediate nodes, the optimized NIC-based reduction still manages to cut the host-NIC synchronization overhead roughly in half.

5.2.3 Validating the Model

Since opportunities to run tests on large-scale systems are difficult to come by, it is important to have an accurate model to provide extrapolations of algorithm scalability. This allows one to consider tradeoffs between different design choices analytically, and thus avoid the need to run extensive tests on rare machines that are in high demand. This section illustrates the accuracy of the proposed model.

Figure 5.3, shows the predicted and measured NIC-based f-nomial reduction latencies as a function of the degree f for 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 4.2.3 uses the NIC-based parameter values given in Section 4.1, which were derived from serial reduction tests on crescendo.

Direct observation of the figure suggests that the model predicts NIC-based reduction latencies with high accuracy. For instance, when chosing among NIC-based f-nomial trees, the model correctly claims 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 on this particular 31-node system. Although not shown here, the model also accurately predicts host-based reduction performance. Thus, as hoped, the model enables one to choose the best algorithm for a given reduction problem analytically.

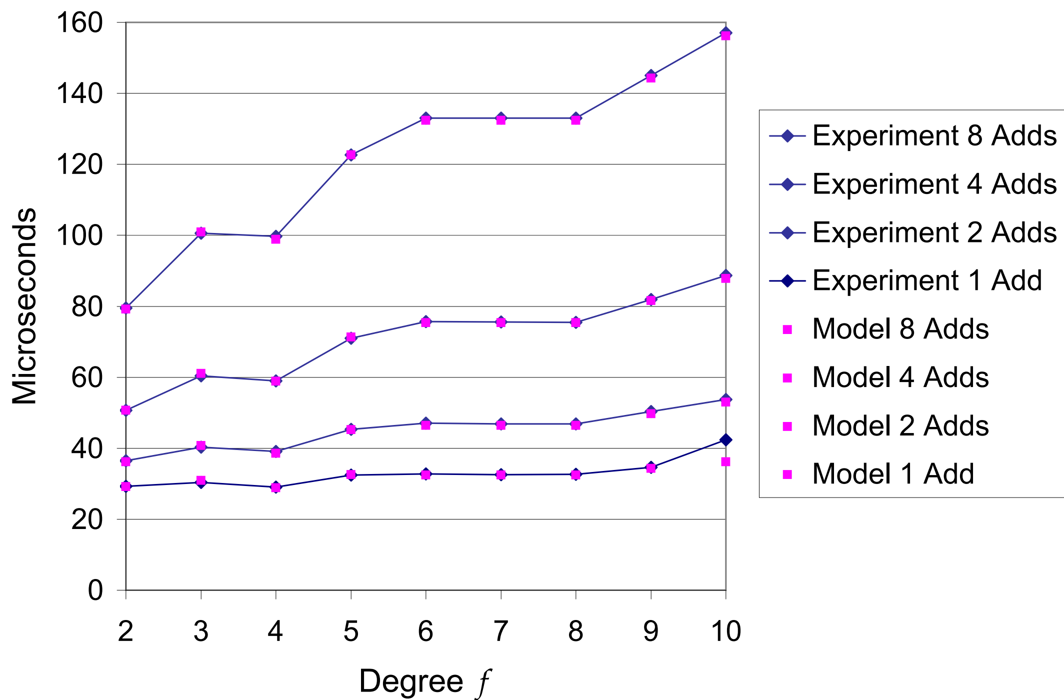


Figure 5.3: Predicted and Measured NIC-based Latencies for 64-bit Floating-Point Addition over a 31-Node f-nomial Tree

5.3 Elimination of PCI-bus Transactions

The experiments presented in this section were designed to illustrate the claim that NIC-based reductions are able to improve the cluster system performance by avoiding inefficiencies between the processors and the network.

5.3.1 Reduced Latency

Since the PCI-bus transaction cost is significant compared to the network latencies between nodes in Quadrics, NIC-based reductions can complete with reduced latency when compared to equivalent host-based implemenations. To show this, NIC-based and host-based binomial tree reduction latencies for several operations are given in Figure 5.4.

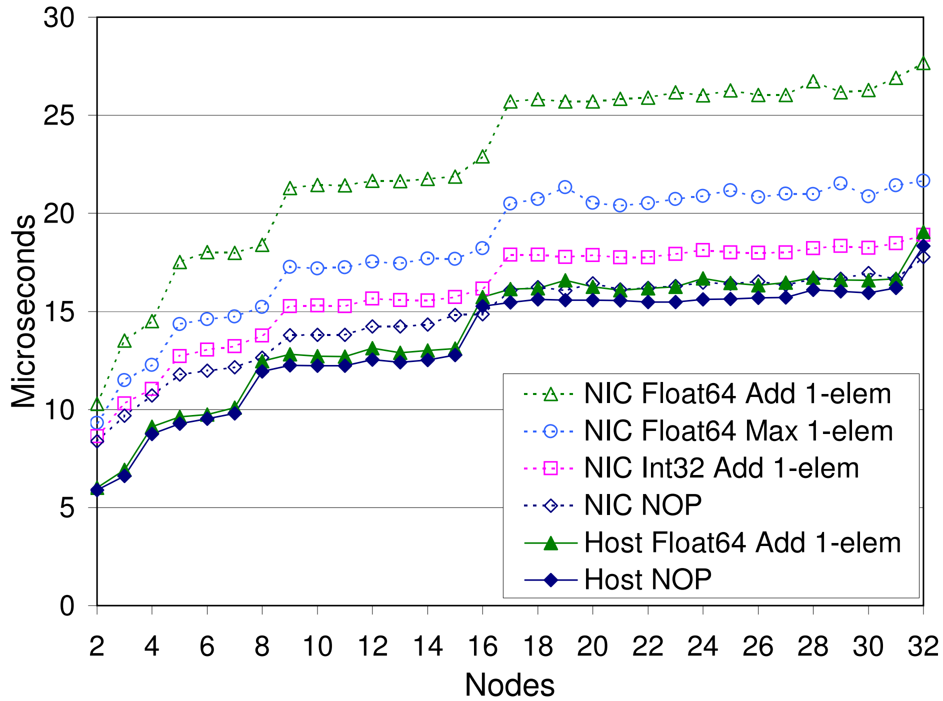


Figure 5.4: NIC-based and Host-based Latencies for Various Operations using Binomial Trees

NIC-based reductions will scale better than host-based equivalents when the savings in PCI-bus transactions outweigh the expense of additional computation cost. From Section 4.1, the savings in PCI-bus transactions is known to be 0.80 μs. Thus, NIC-based reductions involving computation less than 0.80 μs will scale better than host-based reductions. While the computational cost for single-element 64-bit floating-point addition is too steep at 1.50 μs and the cost for single-element 64-bit floating-point maximum is close to the limit at 0.67 μs, the requirements for simple operations like NOP and single-element 32-bit integer addition fall well within the limit at 0.00 μs and 0.25 μs, respectively. As a result, NIC-based implementations for simple operations scale better than the host-based versions. This is what is observed in the curves plotted in Figure 5.4.

It is this advantage that other researchers have used to develop efficient NIC-based implementations of simpler collectives. Figure 5.4 illustrated how this advantage can be extended to reduction operations, albeit for only limited cases.

5.4 Avoidance of Host-level Process Interference

The experiments presented in this section were designed to illustrate the claim that NIC-based reductions are able to improve the cluster system performance by avoiding inefficiencies between the processors and the operating system.

5.4.1 Increased Consistency

NIC-based reductions avoid much of the process interference which host-based implementations are subject to. As a result, NIC-based reductions complete with more consistent latencies than host-based implementations. In fact, 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 steady.

To clarify this point further, Figure 5.5 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 these data points, 100,000 reduction invocations were timed individually, and the resulting set of 100,000 times were grouped into bins of a histogram to produce a distribution.

The NIC-based latencies are largely contained within a sharp spike, while the host-based latencies are spread smoothly across a wide range of values. To be precise, 97% of the NIC-based reductions fall with a spread of only 4 μs, while for host-based

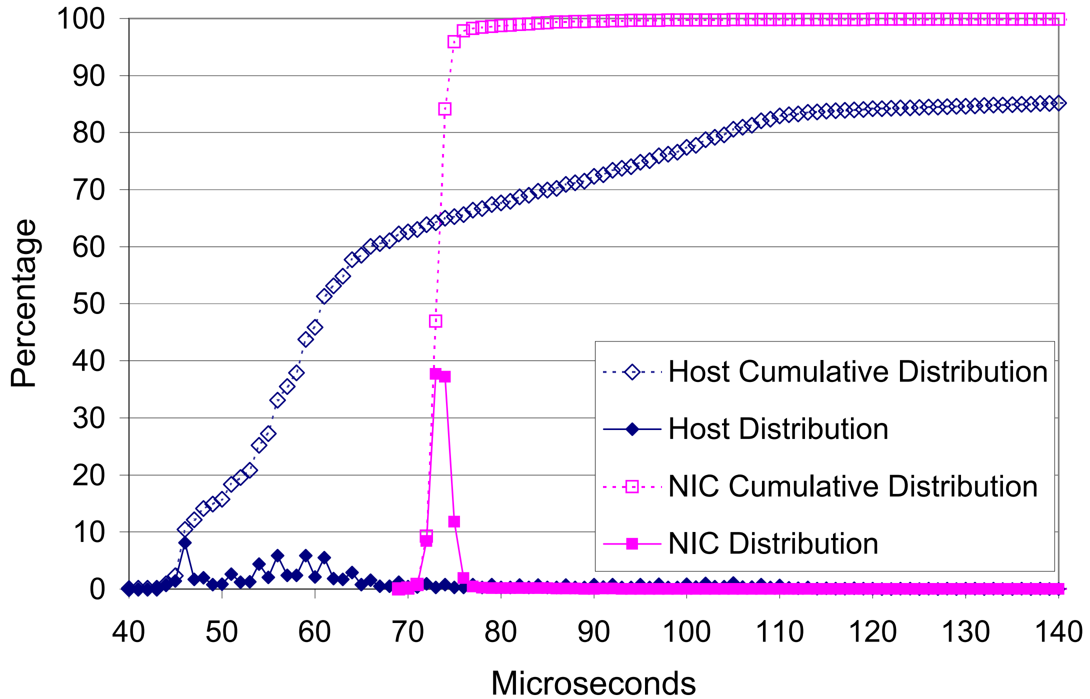


Figure 5.5: Reduction Latency Distributions for Single-Element 64-bit Floating-Point Addition over 900 Nodes

Reduction	**Average (μs)**	**Std. Deviation (μs)**
host-based	89.30	65.26
NIC-based	73.67	0.29

Table 5.1: Reduction Latency Statistics for Single-Element 64-bit Floating-Point Addition over 900 Nodes

reductions, only 57% fall within a spread of 20 μs. After pitching out the highest 1% of the samples, the statistics in Table 5.1 were calculated.

Note the drastic, two order-of-magnitude difference in the standard deviations. This large contrast in consistency is indicative of the non-deterministic effect that process interference imposes on host-based reduction implementations. By avoiding

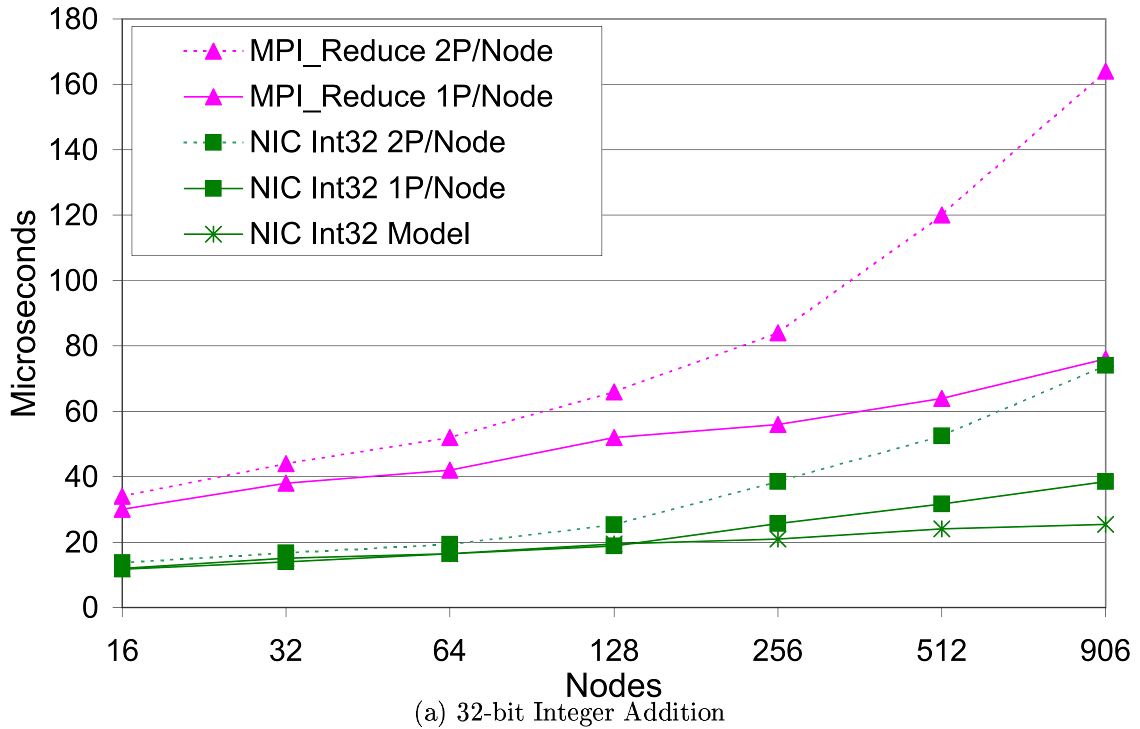
this process interference, NIC-based reductions are more consistent than host-based versions on large-scale systems.

5.4.2 Reduced Latency

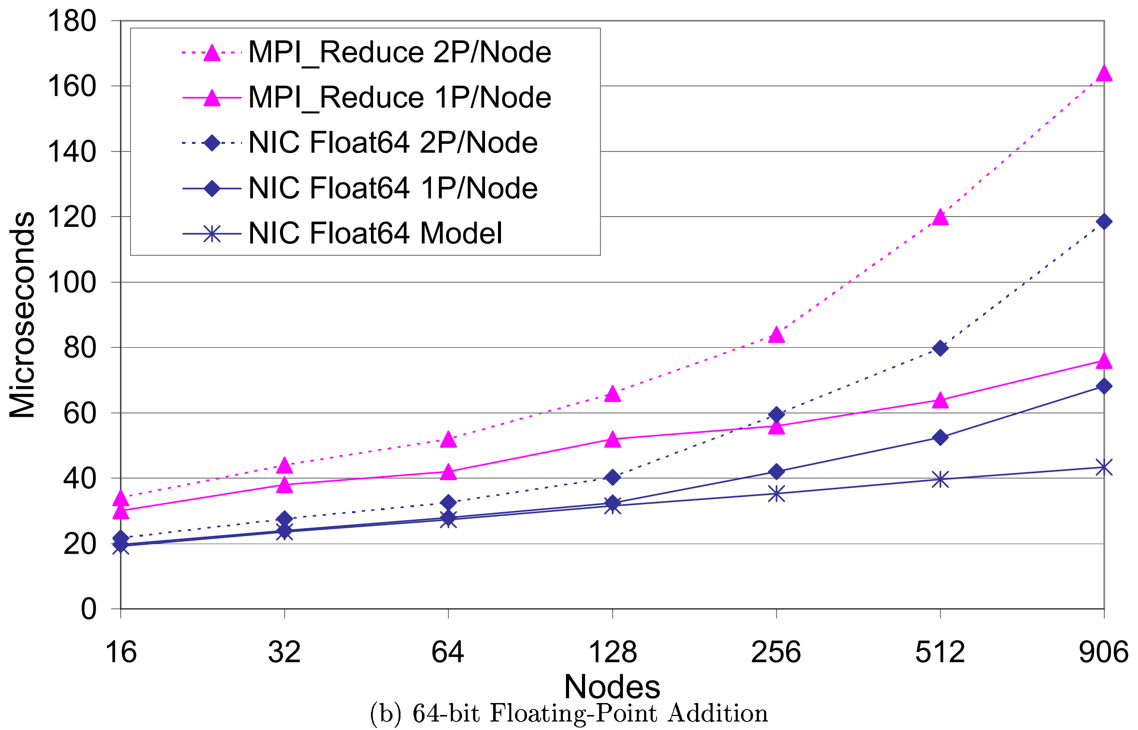
Because NIC-based reductions avoid much of the host-level process interference, they complete with reduced latency on average. Although a first glance, Figure 5.5 may suggest that NIC-based reduction takes more time than host-based reduction, a substantial number of host-based latencies extend far past the right-hand limit of the distribution graph. Actually, as listed in Table 5.1, the average host-based latency is 89 μs, while the NIC-based latency is 74 μs.

To further illustrate this point, the latencies for host-based and NIC-based reduction for a variety of operations and data sizes were recorded, using both one and two processes per node. In all measurements the NIC-based reductions use 4-nomial trees, which provide good performance for the configurations used in the experiments. Figure 5.6 presents single-element vector results obtained for host-based and NIC-based reductions. Figure 5.6(a) presents the results for 32-bit integer addition while Figure 5.6(b) displays the results for 64-bit floating-point addition.

The NIC-based curves scale better than the host-based results. Indeed, the plot for 32-bit integer addition shows that the 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 slower processor, the NIC-based implementation was able to improve upon the host-based reduction, In the largest configuration tested —1812 processors— the



(a) 32-bit Integer Addition



(b) 64-bit Floating-Point Addition

Figure 5.6: Host-based and NIC-based Reduction Latencies for Single-Element Vectors

NIC-based implementation summed 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 host-based versions.

Note that even in the NIC-based implementations, the latencies recorded for two processes per node deviate from the results for one process per node. To a lesser degree, the NIC-based curves follow the same general trend as the host-based latencies. 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 is able to avoid process interference throughout the execution of the reduction. As a result, when compared to the host-based results, the NIC-based reduction implementation is only marginally affected by the process interference.

5.5 Summary

This chapted presented experimental results to validate design choices and show the benefits of NIC-based reductions. First, several details related to the measurement procedure were explained.

Then, experiments were designed to validate the vector split optimization, host-NIC synchronization overhead, and the accuracy of the proposed model. It was shown that the vector split optimization can be quite valuable for larger vectors, decreasing latency by more than a factor of three in one case. The host-NIC synchronization overhead was shown to be substantial, nearly doubling the latency for reductions

involving only a small number of nodes. Optimizations to minimize and hide the host-NIC synchronization were shown to valuable, cutting the host-NIC synchronization overhead in half. Also, the model was shown to be very accurate.

Next, experiments showed that NIC-based reductions do indeed improve large-scale cluster system performance. For simple operations, NIC-based reductions were shown to scale better than host-based versions by eliminating many unnecessary PCI-bus transactions. Also, NIC-based reductions were shown to be more consistent and finish with lower latency than host-based versions on large-scale systems by avoiding interference with the operating system on the host processors.

CHAPTER 6

CONCLUSIONS & FUTURE WORK

This section provides a wrap-up of this thesis. It provides concluding remarks about this work and suggests some ideas for future work in NIC-based reductions on modern large-scale clusters.

6.1 Conclusions

This thesis shows that NIC-based reductions are able to improve cluster system performance by avoiding inefficiencies between the processors and the network, as well as, between the processors and the operating system, which are inherent to host-based implementations. Elminating PCI-bus transactions and avoiding process interfernce at the host-level enables NIC-based reductions to complete with reduced latency on a more consistent basis than host-based versions.

NIC-based implementations are thus potentially valuable, but they do not come for free. One must perform processing on a slower and less functional processor and deal with host-NIC synchronization overhead. However, design techniques are available to work around these issues. NIC processors that only support integer instructions may emulate floating-point operations using available software libraries. Even with their slow processing speed, NIC processors are fast enough to handle the limited processing

requirements of typical reductions in practical programs. Additionally, algorithms, such as f-nomial trees, can be used to balance communication with computation by providing a range of communication structures. Further enhancements like the vector split optimization can be used to increase parallelism to better utilize the collective power of many NIC processors. Finally, optimizations can be used to minimize and/or hide host-NIC synchronization costs.

A simple model derived from LogP was developed to analyze algorithm design choices and to extrapolate behavioral trends to predict reduction latencies on large-scale systems. This model was applied to f-nomial trees, and the analysis for finding the optimal degree for a given problem was presented.

Finally, experimental results are provided to validate various design choices and demonstrate the benefits of NIC-based reduction. NIC-based reductions show lower latency and better scalability than host-based implementations on large-scale systems. In the largest configuration tested —1812 processors— the NIC-based algorithm summed 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 host-based versions.

6.2 Future Work

The concept of NIC-based reductions is quite fresh and many interesting ideas remain to be explored. Currently, there seems to be three especially promising areas: hybrid host-based/NIC-based reductions, asynchronous collectives, and improved NIC processor architectures.

NIC-based reductions save during communication but lose during computation because NIC processors provide limited processing capability and are very slow. Host-based reductions perform fast computation but suffer from PCI-bus transactions and process interference with the operating system. A hybrid approach could be developed to take advantage of the benefits of both while minimizing the penalties. The most natural application would be to use NIC-based gathers in between reduction phases to collect data to be reduced using the host processor. In this way, one may employ the benefits of NIC-based reduction without incurring the penalty of expensive computation costs.

Once initiated, NIC-based implementations execute collectives without requiring assistance from the host processors. By offloading the collective to the NIC processors, the host processors become free to do other work. On large-scale clusters, where collectives take considerable time to complete, the host processor may be allocated substantial time in which to do extra processing. This suggests that asynchronous collectives may be valuable. Such collective implemenations would allow the host to initiate a collective and then perform significant amounts of computation while the NIC is busy carrying it out. By overlapping computation with communication, it may be possible to significantly reduce application execution time.

Even with the current limitations of NIC processors, NIC-based reductions are able to outperform host-based implementations. However, NIC processors lag far behind host processors in capability and speed, and the potential exists to attain much greater benefits by improving NIC processor architecture. For instance in SAGE, the large majority of reductions use floating-point operations. Currently, these operations must be emulated using integer instructions and are very slow. By addiing a floating-point unit (FPU) to the NIC processor, one could expect to see performance much closer to that observed for integer results, which scale considerably better. Even further improvement could be achieved by increasing NIC processor speeds, which are currently one to two orders of magnitude slower than host processors. In addition to decreasing NIC-based reduction latencies, such architectural enhancements would simplify their implementation by reducing the processor's sensitivity to different reduction operations and data sizes.

Appendix A

This appendix provides a psuedo/C code listing of how to initialize and use an f-nomial tree communication structure for reduction operations.

It is assumed that the processes in the collective group are numbered with IDs starting from 0. Without loss of generality, it is also assumed that process ID 0 is the root of the reduction. (The algorithm can be used to describe the communication structure for a general root if one performs a logical remapping by shifting all process IDs cyclically down by an amount equal to the ID of the root.)

Each process must first initialize the f-nomial tree communication data structure by calling init_fnomial_tree() with the degree f, the number of processes involved, and the local process ID, as well as, a pointer to an empty communication data structure to be filled-in by the function. Having done this, a process may use the communication structure to carry out an f-nomial reduction operation as shown in reduce_fnomial_tree(). One must pass in a pointer to the local data, a pointer to a memory segment to be used as receive buffers, the data size, and the filled-in communication structure.

Reducing by an f-nomial Tree:

```
procedure reduce_fnomial_tree

  // Inputs:
  //   data_local:   pointer to local data
  //   data_remote:  pointer to memory segment for receive buffers
  //   data_size:    size of reduction vector
  //   fnomial_tree: filled-in communication data structure

  // Temporaries:
  uint data_temp = 0;
    // index into receive buffers
  uint num_recvs;
    // convience variable of the number of receives in a phase

  // Process any children
  for(phase = 0; fnomial_tree->array_receives[phase] != 0; phase++)
  {
```

```
    // number of messages we'll receive from children of this phase
    num_recvs = fnomial_tree->array_receives[phase];

    // wait for messages from children
    wait for messages to fill buffers at
    [data_temp, data_temp + num_recvs - 1] * data_size + data_remote;

    // reduce the data from children with local data
    reduce the data in buffers at
    [data_temp, data_temp + num_recvs - 1] * data_size + data_remote
    with             data_local
    store result in  data_local;

    // point to head of receive buffers for next phase
    data_temp += num_recvs;
  }

  // Send reduction data to parent
  send       data_local
  to         fnomial_tree->id_parent
  at buffer  fnomial_tree->child_num * data_size + data_remote;

end procedure
```

Initializing an f-nomial Tree:

```
procedure init_fnomial_tree
  // Maps process ID == 0 as root.

  // Inputs:
  //   degree:   degree, f, of the f-nomial tree
  //   id_count: number of processes involved
  //   id_local: local process ID, counts from 0

  // Outputs:
  //   fnomial_tree: communication data structure
  // i.e.,
  //   fnomial_tree->array_receives[]:
  //      number of children during each phase
  //   fnomial_tree->id_parent:
  //      parent process ID (equals id_local for root)
  //   fnomial_tree->child_num:
  //      which number child we are to our parent

  // Temporaries:
  uint phase  = 0;
    // which communication phase we are computing
  uint stride = 1;
    // how many process IDs are skipped between
    // adjacent children within a phase

  // Initialization
  // assume we are the root
  fnomial_tree->id_parent = id_local;

  // While we haven't covered the entire tree...
  while(stride < id_count)
  {
    // assume we have no children in this phase
    fnomial_tree->array_receives[phase] = 0;

    // If we are a parent in this phase...
    if(FLOOR(id_local / stride) MOD degree == 0)
    {
      // For each of our (possible degree-1) children...
```

```
      for(uint index = 0; index < (degree-1); index++)
      {
        // If the possible child really exists,
        // increase number of receives for this phase
        if(id_local + (index+1) * stride < id_count)
          fnomial_tree->array_receives[phase]++;
      }

    // Else, we must be a child in this phase...
    } else {
      // Note our parent's id
      fnomial_tree->id_parent =
        FLOOR(id_local / (stride * degree)) * (stride * degree);

      // and which number child we are to our parent
      fnomial_tree->num_child =
        phase * (degree-1) + (FLOOR(id_local / stride) MOD degree) - 1;

      // After determining our parent, our part is done
      break;  // out of the while
    }  // end if

    // Move on to the next phase of the tree
    stride *= degree;
    phase++;
  }  // end while

end procedure
```

BIBLIOGRAPHY

[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] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. K. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, Febuary 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] Compaq and Intel and Microsoft. *VI Architecture Specification*, 1.0 edition, December 1997.

[14] 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.

[15] 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.

[16] J.R. Hauser. SoftFloat.

[17] 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.

[18] InfiniBand Trade Association. *InfiniBand Architecture Specification*, 1.0 edition, October 2000.

[19] 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.

[20] 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.

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

[22] S. Pakin, V. Karamcheti, and A. A. Chien. Fast Messages: Efficient, portable communication for workstation clusters and MPPs. *IEEE Concurrency*, 5(2):60–73, April /June 1997.

[23] 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.

[24] F. Petrini, D. Kerbyson, and S. Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *Proceedings of the Supercomputing Conference*, Phoenix, Arizona, November 2003.

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

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

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

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

[29] P. Shivam, P. Wyckoff, and D. K. Panda. Emp : Zero-copy os-bypass nic-driven gigabit ethernet message passing. In *Proceedings of the Supercomputing Conference*, Denver, Colorado, November 2001.

[30] R.A. van de Geijn. On Global Combine Operations. April 1991.

[31] 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.

[32] 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.

[33] 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*, page to be presented, Kahosiung, Taiwan, October 2003.