

An Analysis of the Cost Effectiveness of an Adaptable Computing Cluster

Keith D. Underwood, Walter B. Ligon, III, Ron R. Sass

The corresponding author is K. D. Underwood.

K. D. Underwood is with Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM. 87185-1110,

E-mail: kdunder@sandia.gov

W. B. Ligon and R. R. Sass are with the Holcombe Department of Electrical and Computer Engineering, Clemson University,
105 Riggs Hall, Clemson, SC 29634-0915. E-mail: {walt, rsass}@clemson.edu

May 21, 2004

DRAFT

Abstract

With a focus on commodity PC systems, Beowulf clusters traditionally lack the cutting edge network architectures, memory subsystems, and processor technologies found in their more expensive supercomputer counterparts. What Beowulf clusters lack in technology, they more than make up for with their significant cost advantage over traditional supercomputers. This paper presents the cost implications of an architectural extension that adds reconfigurable computing to the network interface of Beowulf clusters. This extension is called an intelligent network interface card (INIC). A quantitative description of cost-effectiveness is formulated to compare alternatives.

Cost-effectiveness is considered in the context of three applications: the 2D Fast Fourier Transform (2D-FFT), integer sorting, and PNN image classification. It is shown that, for these three representative applications, there is a range of basic hardware costs and cluster sizes for which the INIC is more efficient than a purely serial solution or an ordinary cluster. Furthermore, the cost model has proven useful for designing the next generation INIC.

Keywords

intelligent network interface, dataflow computing, reconfigurable computing, cluster computing, Beowulf Cluster.

I. INTRODUCTION

Beowulf-class computers (cluster computers based on commodity off-the-shelf, COTS, hardware and open source system software) have emerged as a low cost supercomputing solution for a variety of problems. Unfortunately, the COTS hardware that reduces the cost of Beowulf clusters also limits their applicability to some classes of problems that depend on a high performance interconnect to achieve scalability, such as the Fast Fourier Transform. In general, Beowulf clusters lack the cutting edge network, memory, processor, and I/O subsystems found in their more expensive supercomputing counterparts. Although Beowulf clusters are low cost, they must provide scalability for applications to be cost-effective.

Reconfigurable computing (RC) is a cutting-edge processing technology currently receiving a great deal of attention. It is based on configurable hardware, predominantly Field Programmable Gate Arrays (FPGAs). By utilizing the ability of programmable hardware to instantiate custom functional units to implement dataflow computations, Reconfigurable Computing has been shown to be effective at providing speedups for an assortment of applications. Unfortunately, reconfigurable computing faces two major hurdles: cost and general applicability. The interplay of these two factors can have a devastating impact on the cost-effectiveness of RC. Specifically, hardware that is expensive and not broadly applicable is difficult to justify purchasing.

Historically, high density FPGA devices have been expensive. Reconfigurable computing cards based on these devices have been low-volume specialty parts. This further contributes to the high cost of the technology. Recently, the cost of high-density FPGA devices has been falling; however, reconfigurable computing platforms are still low-volume, specialty components. RC platforms will not reach commodity status until high-volume applications arise.

High-volume applications of FPGAs have long been hampered by FPGA's lack of general applicability. For some applications, they offer impressive speedup, while for others they offer no performance advantage. One of the earliest barriers facing RC was the insufficient gate counts to support floating-point applications. Although the underlying FPGA technology has matured to the point that it can be competitive with workstations for some floating-point applications [1], [2], it is still not ideal for a number of applications that require intensive double-precision floating-point. Hindering the technology further is its reliance on the PCI bus to transfer data to and from host memory. The low bandwidth and high latency (for a processor interconnect) of the PCI bus often decimates the speedups that RC can offer. Since a reconfigurable computing card and a network card would typically share a single PCI bus, an application that performs significant communication must pass data over the PCI bus three times — network interface to host memory to reconfigurable computing card to host memory. This overhead further complicates the applicability to commodity clusters.

A. Adaptable Computing Cluster

With the ever growing demand for higher performance, it is appropriate to investigate how revolutionary computing models, such as RC, can best be brought into the mainstream. An Intelligent Network Interface Card (INIC) based on a high performance NIC and reconfigurable computing was proposed in [3]. The resulting Adaptable Computing Cluster (ACC) was evaluated and it was shown that the network interface is an excellent place to exploit reconfigurable computing [3], [4], [5]. Given this positive evaluation of speedup over a range of applications, this paper addresses the question: “does the performance justify the additional cost?”

B. Cost-Effectiveness of an INIC

Quantitative measures, such as speedup and iso-efficiency, have been used for decades to evaluate parallel architectures [6]. Likewise, the general principle of cost-effectiveness is a key

component of the system design process. Indeed, the oft-touted advantage of a Beowulf-cluster is cost-effectiveness. However, there has been scant work in a pragmatic definition of cost-effectiveness, and the few formulations that do exist are simplistic. Some early work assumes that cost is proportional to the number of processors [6]. Others have included the number of communication links [7]. A more realistic cost equation accounts for memory and assumes that dual processor nodes cost 33% more than single processor nodes [8], [9]. However, no cost models to date address the interconnection cost, which — for Beowulf-class machines — is distinctly non-linear. While existing cost functions capture first-order effects, they are not representative of Beowulf-class machines.

The express goal of this work is to show the cost-effectiveness of adding the INIC feature to a Beowulf cluster. An important corollary to this goal is that a more complex formulation of cost-effectiveness is a valuable metric. The model is critical at two junctures: when deploying a new system to meet a particular application demand, and when designing new equipment for inclusion in such systems. While the primary purpose of a system is to accomplish the mission, an implicit goal when engineering any system is to minimize cost. When designing a new system, it is tempting to over design individual components for the sake of maximum reuse and maximum performance. In practice, many features are seldom used, and component reuse is rare. Alternatively, it is tempting to make choices that impact performance to lower cost. In reality, both cost and performance are critical to achieving cost-effectiveness, and a cost/performance analysis is needed for each design choice. Furthermore, as shown in Section V, cost-effectiveness can be used to design improved components.

C. Overview

The rest of this paper is organized as follows. Section III summarizes previous performance results to provide the performance portion of a quantitative cost/performance analysis. Using a simple linear least squares model, the performance data is fit to a linear speedup equation for three applications on three cluster platforms.

Next, we develop a cost model to describe each of the three cluster technologies employed. There are actually two cost/performance ratios of interest. The first, denoted E_{serial} , compares each of the three parallel technologies against a single, serial computer. In contrast, $E_{cluster}$ compares two parallel cluster technologies; in our case an ordinary cluster against an INIC-enhanced

cluster. This second measure of cost-effectiveness is shown over a range of realistic network and hardware costs, identifying cost-effective break-even points (Section IV). Considering a range of realistic costs allows us to consider the cost impacts of Moore's Law on this technology.

Finally, Section V illustrates how these cost-effectiveness measures can be used to improve future generations of network interface cards. Realistic cost data from the development of the GRIP2 card [10] is used.

Two sections bookend the three core sections: the next section describes the three platform technologies used, and Section VI describes related work. Finally, Section VII summarizes the main contributions of this work.

II. ACC SYSTEM ARCHITECTURE

The goal of Clemson's Adaptable Computing Cluster (ACC) project is to explore architectural enhancements to Beowulf clusters that improve performance and scalability without reducing the cost-effectiveness of the system. Toward that end, a cluster was constructed with two network interfaces: an enhanced network interface with reconfigurable computing technology and a conventional Gigabit Ethernet NIC. Combined with data about a next generation enhanced NIC, this provides three parallel platforms that form a basis for the work presented here. The traditional technology is the standard Gigabit Ethernet card solution. The prototype technology is the initial experimental hardware, and the next generation technology is the hardware that is under development.

Figure 1 illustrates the core of the enhanced network interface. Whereas a traditional NIC simply buffers data in memory (or FIFO) between the host and the network, an INIC inserts reconfigurable logic along the datapath. The reconfigurable logic could then be used in a range of modes, including:

[Figure 1 about here.]

- *Compute Accelerator* — Defined as using the FPGAs strictly for application computing tasks, this mode significantly enhances the computing power of a node for some tasks. Research demonstrating the ability of reconfigurable computing to accelerate certain classes of applications is too extensive to document here.
- *Combined Compute/Protocol Accelerator* — Placing computing and protocol elements in the

reconfigurable logic takes advantage of the insertion of a high-performance computing core in the network datapath. Reconfigurable logic can manipulate data passing between the host and the network (at little or no cost), or can serve as a processor with a low-latency network connection. This allows the reconfigurable logic to benefit a large class of applications including those addressed in Section III, and support abstractions such as MPI derived data types.

- *Protocol Processor* — As a protocol processor, the FPGAs are used strictly for network processing. A properly designed Intelligent NIC could perform all of the protocol processing for a cluster node, offering more features (such as collective operations) and higher bandwidth than current commodity network subsystems. Unlike some solutions that have attempted to use an embedded processor on the NIC for protocol processing, the INIC approach adds additional computing capabilities to the network interface. If adequate external memory bandwidth is provided, this additional logic can provide protocol support for very high rate networks (ten gigabits per second should be achievable with current FPGAs). Protocol processors are useful for any application performing significant communication as they offload processing from the CPU and can significantly improve network performance[11].

Prototype System

The ACC experimental platform is an 16-node Beowulf running the Scyld Linux distribution. Eight of the nodes contain a 32-bit PCI motherboard with a 1GHz Athlon and 512 MB of RAM. On the PCI system bus is a SysKonnnect PCI Gigabit Ethernet NIC and a Fast Ethernet NIC. These eight systems also include an ACEII card. The ACEII card has an onboard PCI bus attaching a μ SPARC processor and a PCI Mezzanine Connector (PMC) to the FPGAs. The PMC is populated with an Alta Technologies PMC Gigabit NIC based on the Packet Engines Hamachi chipset. The ACEII card is a reconfigurable computing board from TSI TelSys. Figure 2 shows a simplified block diagram of the board. The eight remaining nodes each contain a 1.1GHz Pentium III processor with a SysKonnnect PCI Gigabit Ethernet NIC and a Fast Ethernet NIC on a 32-bit PCI bus.

The prototype used is not ideal. It has a few deficiencies that prevent it from achieving its full potential as an Intelligent NIC. These include slow PCI interfaces, a single 32-bit 33MHz bus for all data traffic to the FPGAs, an older generation of reconfigurable logic, and limited memory attached to the FPGAs. Nonetheless, it will suffice to demonstrate the concepts being

addressed, and it has the advantage of a standard PMC connector permitting the use of off-the-shelf network adapters. Newer boards with similar cost address some of these issues but lack the features necessary to support networking functions. Theoretically, an Intelligent NIC would be implemented as a single chip with external RAM — similar to modern high performance NICs. Section III predicts the achievable performance using a next generation Intelligent NIC and addresses the achievable performance with the prototype INIC.

[Figure 2 about here.]

III. APPLICATIONS AND THEIR PARALLEL SPEEDUPS

Two of the most important factors of any architecture are the performance of applications on the architecture and the cost to achieve that performance. Three applications — a 512×512 2-D Fast Fourier Transform (2D-FFT), integer sorting, and PNN image classification — were selected for this study. These applications were chosen for two reasons. Pragmatically, they can be implemented on the prototype. Secondly, they emphasize the ACC architecture's ability to significantly outperform a commodity cluster. For each of the applications, the architecture-specific implementation is a derivation of the standard parallel implementation with changes to exploit the new architectural features. This highlights the ability to use the INIC with the same programming model as existing clusters. This section briefly presents the applications, their implementations, and their performance. For further details, refer to [3], [4], [5]. In the diagrams used to explain the implementations, rounded boxes describe processes, rectangles represent function blocks, and arrows represent the flow and sequence of data. Where there is ambiguity, sequences are numbered. Each diagram shows a single network transaction. Note that many of these transactions occur concurrently.

A. 2D-FFT

[Figure 3 about here.]

The first application considered was the two dimensional Fast Fourier Transform. The baseline parallel and serial implementations use the highly optimized Fastest Fourier Transform in the West (FFTW) package[12]. On a distributed memory architecture, the matrix is distributed over the processors in a row-block distribution. The algorithm as implemented can be decomposed as:

- compute the 1D-FFT for each row
- transpose the matrix (redistribution of data)
- compute the 1D-FFT for each row
- transpose the matrix (a second redistribution of data)

The matrix transpose becomes the bottleneck in such a scheme and is a perfect target for implementation with an INIC. Like the parallel implementation, the INIC matrix transpose is composed of three operations: a local transpose step, an all-to-all communication, and a final permutation. Unlike the standard parallel implementation, an implementation using INICs pushes all of the data manipulation needed for the transpose (on both the send and receive sides) onto the INIC, as shown in Figure 3. This allows the data manipulation to be embedded in the communication at little additional cost (slightly higher latency than a network transaction without the transpose requires). Furthermore, the communication protocol used can be customized to the specific application, since each node knows exactly how much data will be exchanged with every other node.

B. Integer Sorting

[Figure 4 about here.]

The second application considered, integer sorting, is a common benchmark. Although some benchmarks use non-uniformly distributed keys (often Gaussian [13]), the results presented here are based on synthetically generated and uniformly distributed keys. This is a well-established precedent and allows a focus on the evaluation of the basic I/O and computational performance of the architecture. As others have recognized, sampling in a pre-sort phase helps address the shortcomings of this assumption by leading to a more balanced workload.

Each of the implementations first sorts the data into buckets that fit well in the processor cache. These buckets are then sorted with Count Sort as in [14]. For the parallel implementation, a distributed memory to distributed memory sort over a power-of-two ($2^k, k \in \mathbf{Z}$) number of processors is evaluated. Each processor begins by bucket sorting its data into P buckets. Bucket i from each processor is then sent to processor i . As a processor receives the data, it bucket sorts the data into buckets designed to fit in processor cache. Once all of the data has been collected, each bucket is sorted with Count Sort. The Count Sort is the final sorting phase. With 32 bit

integers and more than 128 buckets there is no need for the final bubble sort described in [14]. On a problem size of 2^{21} keys or more, a minimum of 128 buckets are needed for the problem to map into cache.

Figure 4 illustrates the differences in data flow for a traditional parallel implementation and two INIC implementations. Since bucket sorting is particularly amenable to implementation on the INIC, both bucket sorts and the communication operations should be implemented in hardware. Unfortunately, the limited resources of the Xilinx 4085XLA devices on the prototype prohibit performing the full receive side bucket sort in hardware; however, the received data can be pre-sorted into 16 buckets, each of which can be bucket sorted by the host. As shown in Section III-D, this approach can still provide a performance benefit.

Figures 4(b) and (c) show a block diagram of operations in the INIC reconfigurable logic. On the sending side, data is transferred directly from host memory to INIC memory. Along the path, the data is manipulated to perform the bucket sort operation. Like the parallel implementation, the INIC implementation can overlap communication with computation. In fact, the INIC can start transmitting data at lower bucket thresholds (one packet) since there is no computational overhead¹ for starting a send. Hence, on the sending side, the INIC handles all of the computation and protocol processing, leaving the processor free for other tasks such as disk I/O. On the receiving side, the bucket sort can be done as data is received². As minimum thresholds are reached, data is transferred to the host. After all data is received, each bucket is sorted with Count Sort.

C. PNN Image Classification

[Figure 5 about here.]

The third application considered is PNN image classification. The Probabilistic Neural Network (PNN) was designed by Specht [15] using a kernel function by Parzen [16]. It was later applied to the domain of satellite image classification by Chettri [17]. The algorithm is based on the equation:

$$f(\mathbf{X}|S_k) = \frac{K}{P_k} \sum_{i=1}^{P_k} \exp \left[-\frac{(\mathbf{X} - \mathbf{W}_{ki})^T (\mathbf{X} - \mathbf{W}_{ki})}{2\sigma^2} \right] \quad (1)$$

¹This is not to say that there is no computation involved in starting a send, only that starting a send is handled by hardware that sits idle if no send is in progress.

²Again, the prototype splits the receive operation between the card and the host.

where $K = (2\pi)^{-d/2}\sigma^{-d}$, \mathbf{W}_{ki} is the i^{th} training vector from the k^{th} class, P_k is the total number of training vectors in the k^{th} class, d is the dimension of each vector and σ is a smoothing parameter. The pixels, \mathbf{X} , and weights, \mathbf{W}_{ki} are vectors of length d where each element of the vector corresponds to a satellite sensor measurement. The algorithm computes a value for each pixel for each of N classes and assigns a class to the pixel based on the highest score.

The PNN algorithm is a data-parallel application that operates on a single pixel at a time. Classification of a single pixel depends only on the value of the pixel and not on the value of any adjacent pixels. More importantly, the computation required for each pixel is significant. Thus, PNN is easily parallelized on a standard Beowulf Cluster with a relatively slow network interface.

The parallel implementation used here assumes that the image to be classified resides on the “master” node and that the classified image will be returned to the master node. Each “slave” node maintains a local copy of the training vectors. Total time is measured as the time to distribute pixel data to the slaves, classify the pixels on the nodes, and return the classified pixels to the master.

For the PNN algorithm, the INIC is capable of implementing all of the computation. Therefore, it is configured as an auxiliary processor with no communication capabilities, as shown in Figure 5. Ostensibly, the pixels to classify and the classified results could be communicated directly between the master and the INIC in each node, but this is unnecessary. The performance impact from using a separate network interface is negligible.

D. Performance Analysis

[Figure 6 about here.]

Figure 6 compares potential next generation INIC speedup with the speedup of a prototype INIC implementation and the measured performance of a baseline (Gigabit Ethernet) cluster. Results are measured for the prototype implementation of the 2D-FFT and PNN³ and estimated (based on measured performance for the 2D-FFT) for the integer sort implementation. All speedups for the next generation INIC are estimated. Speedups are relative to a single node without a reconfigurable computing card. This is a reasonable comparison for FFT because in

³Measurements were taken on 2, 4, and 8 nodes and estimated for 16 nodes

the serial implementation, the transpose is unneeded. For integer sorting, the potential increase in performance given by a reconfigurable computing board in a single node would be eliminated by the PCI interface. For all three applications, the cost is compared to that of an unenhanced serial node; thus, speedup should be relative to the same technology. Speedups are plotted as individual points on the graph. In addition, the least squares method was used to generate a linear model of speedup as a convenience for later comparison. This line is also plotted.

Figure 6 shows that the INIC has great potential to accelerate the three applications under consideration. Figure 6(a) indicates that the prototype INIC will offer better performance than the baseline cluster for the 2D-FFT. This is achieved by performing the transpose data manipulation along the network data path (with minimal additional cost) and by implementing a custom protocol that is efficient for small data transfers. Figure 6(a) also indicates that a next generation INIC has the potential for linear speedup up to 16 processors, but the prototype cannot achieve this. In this instance, the prototype performance is limited by the single 32-bit 33 MHz bus connecting the FPGAs to the host and the network.

Figure 6(b) shows that the potential prototype INIC performance is only moderately better than the performance of the baseline cluster for integer sorting. For this application, the prototype INIC is constrained by both the limited bus bandwidth and the density of the FPGAs. The potential INIC speedup is much higher, even superlinear. Superlinear speedup is achieved by performing both the send side and the receive side bucket sorts in the INIC. This hides a significant part of the computation in the communication operation.

Finally, Figure 6(c) indicates that even the prototype can achieve drastically better performance than a baseline cluster for PNN image classification. This is simply an illustration of the capabilities of FPGAs as computing elements. In the case of an Adaptable Computing Cluster, this capability can scale very well for some applications.

IV. COST ANALYSIS

Using commodity PCs in a cluster environment requires the addition of high-performance networking equipment to each node. In addition, a network backplane (often a switch) must be added externally, further increasing the cost of a cluster. Furthermore, achieving the Adaptable Computing Cluster (ACC) results presented in Section III requires the addition of reconfigurable hardware to each network interface. Each piece of extra hardware added to the cluster

increases the cost of achieving the promised performance. Although clusters are typically built from a collection of serial nodes, the extra hardware needed to build the cluster changes the cost relationship between clusters and single node serial implementations. This is particularly significant when adding an expensive component like an INIC. The question that arises is one of cost-effectiveness: how can the maximum performance per unit cost be achieved?

Cost-effectiveness is an extension of the traditional measure of speedup. Though measurements such as speedup and iso-efficiency are valuable tools, they do not account for the cost of achieving performance targets. At first glance, it would seem that cost-effectiveness adds little information; however, the cost model for a cluster of N nodes is more complex than a simple linear relationship ($N \times$ the cost of a serial machine). Cost-effectiveness, E , is defined as a ratio of the ratios of price to performance for two technologies, or:

$$E = \frac{\frac{C_1}{Speedup_1}}{\frac{C_2}{Speedup_2}}. \quad (2)$$

Values of E greater than one indicate that technology 2 is more cost-effective. Likewise, a value less than one means that technology 1 is more cost-effective.

To calculate E , the cost of the two technologies being compared must be determined. The ACC will be evaluated based on its cost-effectiveness relative to a single serial node and a standard Beowulf Cluster with Gigabit Ethernet. The cost of a single node is simple to determine, but costs for clusters are more complex than those typically modeled. The first step to modeling these costs is to define the cost of a cluster as the sum of the cost of the nodes and the cost of the network:

$$C_{Cluster}(N) = N \times C_{Node} + C_{Net}(N) \quad (3)$$

where C_{Node} is the cost of a node, and $C_{Net}(N)$ is the cost of a network for N nodes. Here, it is assumed that the cost of a node is constant across all nodes in a given cluster. Modern high-performance networks are often purchased as an expandable chassis with some number of installed modules. Typically, even those networks that come as a single unit can be modeled as a base cost plus some incremental cost of expansion over some set of sizes. It is seldom possible to buy high-performance networks of arbitrary sizes. The model accounts for this by defining a

“switch increment” which is the minimum difference in the number of ports between two switch configurations. Hence, $C_{Net}(N)$ can be modeled as:

$$C_{Net}(N) = B(N) \times C_{SwitchIncrement} + C_{SwitchBase} \quad (4)$$

where $B(N)$ is the number of switch increments needed, $C_{SwitchIncrement}$ is the cost of each increment, and $C_{SwitchBase}$ is the baseline cost of the switch. $B(N)$ can in turn be defined as:

$$B(N) = \left\lceil \frac{N}{SizeofSwitchIncrement} \right\rceil \quad (5)$$

Equation 5 contributes a step function characteristic to the overall cost of a cluster.

Turning to the node cost, there are three configurations to consider. The first is the cost of a serial node with no networking, or $C_{SerNode}$. The second adds the cost of a network adapter to form the node cost:

$$C_{Node} = C_{SerNode} + C_{NetworkAdapter} \quad (6)$$

The third adds the cost of reconfigurable computing technology for an INIC enhanced node:

$$C_{INICNode} = C_{SerNode} + C_{NetworkAdapter} + C_{RC} \quad (7)$$

or,

$$C_{INICNode} = C_{Node} + C_{RC} \quad (8)$$

[Figure 7 about here.]

In turn, this can be applied to Equation 3 to get the cost model of an ACC.

$$C_{ACC}(N) = N \times (C_{Node} + C_{RC}) + C_{Net}(N) \quad (9)$$

[Table 1 about here.]

Table I shows the costs for a prototype INIC cluster constructed in December, 2000. Defining C_{RC} to be the difference in component costs between a traditional NIC and an INIC constructed from current generation FPGAs implies that C_{RC} could range as low as \$250. As transistor sizes

shrink in accordance with Moore’s Law, the cost of the FPGA device sufficient to perform the operations considered here will continue to drop.

Now that a cost function has been developed, cost and performance can be combined to consider cost-effectiveness. Cost-effectiveness can be used to compare two technologies based on their *price / performance* ratios. Here, an Adaptable Computing Cluster is compared to a baseline fixed-performance, single-node, serial solution and a Gigabit Ethernet based cluster solution. E_{Serial} will be used to refer to all comparisons (ACC and Gigabit Ethernet) to a serial implementation. In turn, $E_{Cluster}$ will be used to refer to comparisons between the clusters. Since speedup depends on the application and parallel technology, E is defined on a per-application, per-platform basis. E_{Serial} is a special case in which the speedup for the serial node is one; hence, it can be defined as:

$$E_{Serial} = \frac{C_{SerNode}}{\frac{C_{technology}(N)}{Speedup(N)}} \quad (10)$$

where $C_{technology(N)}$ refers to the cost of the technology considered.

Figure 7 compares E_{Serial} for three technologies: a baseline cluster, a prototype ACC (based on the prototype INIC), and a next generation ACC (based on a next generation INIC). For C_{RC} , \$7500 was used for the prototype INIC and \$750 (based on the cost determined in Section V) was used for the next generation INIC. For each graph, two lines are included for each application. One is based on the actual (or estimated) performance of the various platforms. The other is based on linear model of speedup achieved with the least squares estimate. This allows intermediate points to be plotted to illustrate the distinct step-like nature of the cost-effectiveness curve.

In each case, the prototype ACC is distinctly less cost-effective than the other technologies due both to its high cost and relatively low performance increase; however, the next generation ACC can achieve a cost-effectiveness near one for the integer sorting application. In fact, the next generation ACC is expected to achieve a cost-effectiveness drastically higher than one for the PNN application. This is a significant accomplishment since an ACC is drastically more expensive than a serial node.

While E_{Serial} defines the relative cost per unit speedup, Figure 7 is somewhat non-intuitive in that the term $C_{Net}(N)$ causes clusters which do not achieve superlinear speedup to have a cost-

effectiveness less than one. Further, since the purpose is to assess an enhancement to a cluster architecture, it is better to evaluate the cost-effectiveness relative to the baseline architecture — a standard Beowulf Cluster with Gigabit Ethernet. For this, $E_{Cluster}$ is used.

$$E_{Cluster} = \frac{\frac{C_{Cluster}(N)}{Speedup_{Cluster}(N)}}{\frac{C_{ACC}(N)}{Speedup_{ACC}(N)}}. \quad (11)$$

Using Equation 11 as a metric assumes that adequate funds are available to build a cluster of size N , other constraints (space, heat, data sets to be processed) limit the practical cluster size to N , and it is desirable to determine if the performance gains from adding an INIC are justifiable. Where $E_{Cluster}$ is greater than one, an Adaptable Computing Cluster is more cost-effective than a standard Beowulf Cluster (showing an improvement in the *price / performance* ratio). Likewise, a value less than one means that the standard Beowulf Cluster is more cost-effective. Figure 8 compares the relative cost-effectiveness of three implementations of three applications. The baseline cluster always has a cost-effectiveness of one relative to itself. While the prototype ACC is less cost-effective than the baseline cluster, the ideal ACC achieves significantly better cost-effectiveness in many cases. As a note, the anomalies near the low end the SSE graphs in Figure 8 illustrate the dangers of assuming a linear speedup model. The least squares estimate misses some of the actual speedups dramatically and yields inaccurate results for those points in Figure 8.

A significant factor in the E_{Serial} for the baseline cluster is the relatively high cost of the gigabit switch used. This cost also has the potential to skew the relative cost-effectiveness of the two clusters. Similarly, the high cost associated with the low volume of the prototype INIC significantly reduces the cost effectiveness of the prototype ACC. Thus, to illustrate the potential impact of these issues and Moore's Law, Figures 9, 10, and 11 illustrate the impacts of switch cost and INIC cost on the relative cost-effectiveness of a fixed size (sixteen nodes) ACC. Figures 9, 10, and 11 also provide an example of how E can be used to assess a new architectural feature. $C_{Net}(N)$ was chosen to range from the current cost of a Fast Ethernet switch, \$1000⁴, to the cost of a Gigabit Ethernet switch for the prototype, \$28000. Similarly, C_{RC} was chosen

⁴This cost is appropriate for high end Fast Ethernet switches that offer the performance desired in a cluster as opposed to lower end switches targeted at small office environments that do not require simultaneous full-duplex bandwidth on all links.

to range from a minimal cost of \$100 to the prototype INIC cost, \$7500.

For the purposes of Figures 9, 10, and 11, the cost of nodes was maintained at a constant \$2000. While the cost of equivalent nodes will drop over time or, correspondingly, the speed of a fixed price node will go up over time, the applications considered are already limited by network and memory subsystem performance. Network performance is being held constant and memory performance is growing significantly slower than processor performance; hence, it is reasonable to consider the graphs without the need to vary node cost or performance.

The breakpoint between “cost-effective” and “not cost-effective” illustrated in Figures 9, 10, and 11 occurs when $E_{Cluster} = 1$. From the figure, it is clear that an ACC can be a cost-effective enhancement for the applications under consideration at the right cost. The additional performance achievable with a next generation ACC make it a clear choice for these applications in all scenarios except those where the switch cost drops near its minimum while the INIC cost is still near its maximum. The prototype INIC, however, is clearly not a cost-effective enhancement at its current cost. Indeed, reaching cost-effectiveness for both applications would require that C_{RC} drop to \$1000. This is not surprising as the performance enhancement provided by the prototype INIC is significantly constrained by the single-bus architecture employed. Section V will discuss a design capable of achieving the next generation INIC performance enhancement while constraining C_{RC} to \$750.

[Figure 8 about here.]

[Figure 9 about here.]

[Figure 10 about here.]

[Figure 11 about here.]

V. APPLYING COST-EFFECTIVENESS TO BUILD A BETTER INIC

Cost-effectiveness is an important metric for the design of new architectural features. The proper design of an INIC provides an excellent example of this principle at work. When building an INIC, it would be easy to take an off-the-shelf reconfigurable computing board and an off-the-shelf Gigabit Ethernet card and plug the two together. Indeed, this was the approach taken to build the prototype for this work. This same approach could be applied using newer technology cards; however, doing so would yield an \$18,000 INIC. It can be inferred from the preceding

cost analysis that this is not a good idea. If the goal is to be significantly more cost-effective than a commodity cluster, Figures 9, 10, and 11 clearly indicate that a target INIC cost of \$2000 is more appropriate. If performance is ignored, building a NIC with an FPGA for \$2000 is relatively easy. Unfortunately, performance implications cannot be ignored.

In Section III, Figure 6 references the potential speedup of a next generation Intelligent NIC. This potential, based on the use of Gigabit Ethernet for the network fabric, must be achieved to have a significant cost-effectiveness advantage at a \$2000 price point. Thus, designing an INIC requires careful attention to both performance and cost. This section presents a design to achieve the performance of the next generation INIC built for a Gigabit Ethernet backplane presented in Section III while minimizing development and production costs. The proposed design is illustrated in Figure 12.

The first objective is achieving the full potential performance of the INIC architecture. This requires that at least two gigabits per second of bandwidth be available between the FPGA device(s) and the MAC. Simultaneously, there must be two gigabits per second of bandwidth available between the FPGA device(s) and host memory. In addition, there needs to be sufficient FPGA logic resources to perform the tasks required. Finally, there must be enough memory and memory bandwidth to support the buffering and processing of data.

To achieve two gigabits per second of bandwidth to the host memory, 64-bit 66-MHz PCI is required. It is possible to achieve two gigabits of bandwidth between the FPGA(s) and the network using 64-bit 66-MHz PCI, but it is undesirable. Commodity chipsets are available (such as the Vitesse XMAC-II VSC8840) that provide a simple asynchronous FIFO interface with a full gigabit of bandwidth in each direction. Sufficient FPGA logic will be application dependent. For the 2D-FFT, the two Xilinx 4085XLA devices available on the prototype have abundant resources; however, the integer sort application needs additional RAM resources to maintain counts for enough buckets to perform the full bucket sort on the receiving side. A single Xilinx Virtex-II 1000 provides as much logic as the two Xilinx 4085XLA devices and provides the additional RAM needed for the integer sort application.

Memory is used for two purposes on an INIC: buffering packets for communication and buffering data for computation. In the prototype, the SRAM is used as a computation buffer and the external FIFOs are used as a communication buffer. This is not an ideal scenario since

packet data must be kept in the SRAM until an acknowledgment is received. It would be better to have a separate communications buffer to hold data that has been transmitted while waiting for acknowledgment. The buffer should be large enough to keep a sufficient number of outstanding packets to allow full rate communications. In addition, there should be enough memory to statically allocate buffer space for each connection (typically one connection would be used per node in the cluster). Static allocation is not strictly necessary and removing this constraint reduces the size of the communication buffer needed. Fortunately, the memory needed for this type of access is inexpensive, and the hardware design can be simplified by allowing static allocation. Accesses to this buffer will be bursty and sequential and will require 4 gigabits per second of sustainable bandwidth. Since memory needs for the computation buffer are dependent on the application, as much storage and bandwidth as possible should be provided. The computation buffer should be based on Zero-Bus-Turnaround (ZBT) SRAM to allow random accesses without penalty.

[Figure 12 about here.]

The second objective is minimizing cost. The Xilinx Virtex-II 1000 can provide both the PCI interface and FPGA resources needed for the INIC. This saves the cost of a separate component for the PCI interface and saves the design cost of building an integrated device. The BG560 package provides an adequate number of user I/O pins. The Gigabit Ethernet MAC is most easily provided by an off-the-shelf chip such as the Vitesse XMAC-II VSC8840 used by SysKonnnect in some versions of their Gigabit Ethernet cards. It provides a full-featured Ethernet interface for a relatively low cost. Alternatively, the MAC could be implemented in the FPGA, but providing the same level of features found in commercial interfaces would consume a significant amount of relatively costly FPGA fabric.

For the communications buffer, 133-MHz SDRAM with a 64-bit datapath is adequate to meet the performance targets. This memory is the same commodity RAM used in modern desktop machines, so 256 MB can be provided at little additional cost. Accesses to the communication buffer are long, sequential bursts, so SDRAM introduces little penalty. For the computation buffer, ZBT-SRAM should be used to provide penalty free random accesses with reads and writes back to back. ZBT-SRAM is much more expensive, so only a limited amount can be provided while maintaining cost-effectiveness. The proposed design, shown in Figure 12, includes two banks of 2 MB each.

Table II provides a list of components and costs for the design proposed in Figure 12. Support parts include such things as transceivers and clock generators. Volume production begins to play a factor when considering the board fabrication costs. The quoted cost per board was based on production of 1000 boards per month. Dropping to 100 boards per month has only an incremental impact on cost (\$54, or six percent). Dropping to 10 boards per month, which is more characteristic of current reconfigurable platforms, increases fabrication cost to \$400 per board, an increase of thirty percent in overall board cost⁵. More importantly, the non-recoverable engineering (NRE) costs associated with each new board design must be amortized across all of the cards produced. Unlike reconfigurable computing cards that will have total sales of only a few hundred boards at most, a intelligent network adapter could be sold in higher volumes, since it could be marketed for cluster applications, traditional reconfigurable computing applications, and network encryption applications such as IPSec[18], [10].

[Table 2 about here.]

The data in Table II can now be used to calculate a C_{RC} for this design. Of the items listed, only the Virtex-II and the SRAM are completely unique to the INIC. The size of the SDRAM is larger than that necessary for a standard NIC and so it contributes additional cost. Also, the board fabrication costs would be higher for an INIC than a standard NIC because the INIC is a more complicated board and a lower volume board than that used in a commodity NIC. Allowing ninety percent of the SDRAM and board fabrication costs, C_{RC} is calculated to be \$750. This value can be expected to drop over time as FPGA technology matures. Current (2002) market prices are a good indication of this: a Xilinx Virtex 1000 ranges between \$1400 and \$2700 (depending on package and speed grade) while a Xilinx Virtex-II 1000 (a part from a newer family with comparable density) is only \$323.

VI. RELATED WORK

The Adaptable Computing Cluster project seeks to accelerate high-performance parallel computing with RC technology. Similarly, SRC Computers, Inc.[19] has employed reconfigurable computing in a high-performance parallel computing environment. In the SRC-6, MAP processors incorporate FPGAs and give them full access to the shared memory subsystem. While

⁵All cost information was provided courtesy of USC/Information Sciences Institute East.

this addresses many of the interfacing issues facing reconfigurable computing technology, it is implemented in search of peak performance rather than cost-effectiveness.

Technologies such as Myricom's Myrinet [20], used in the Berkeley NOW [21], and Compaq's Servernet-II [22] are demonstrating that cluster users are willing to pay for higher performance networks in commodity systems. It is reasonable to expect that a volume-produced INIC would be of similar cost to Myrinet or Servernet-II interfaces.

A number of efforts have researched using dedicated computational resources for network processing. Research at the University of Wisconsin [23] suggests that fixing one processor of an SMP for communication processing benefits light-weight protocols and improves performance when communication is a bottleneck. Indeed, many gigabit networks now include embedded processors on the NIC for various network processing tasks. Research efforts such as Typhoon [24], Georgia Tech's VCM [25], RWCP's GigaE PM project [26], and the University of British Columbia's GMS-NP project [27] all use such a processor to accelerate distributed computing. Similarly, research at CMU explored hardware to augment an ATM card to boost distributed programming speeds with a Hardware Assisted Remote Put (HARP)[28]. An INIC can subsume the functionality proposed by these efforts; however, it has the potential to be more cost-effective by providing flexible computational capabilities as well.

While others have created clusters with reconfigurable cards in each node [29], we believe that integrating the configurable fabric with the NIC is an important innovation. Specifically, it is difficult to achieve cost-effectiveness across a wide range of applications if the reconfigurable computing units are unable to process the network data stream directly.

Research and development efforts using FPGAs for networking are too numerous to cover thoroughly here. Applications at the network interface include such things as data encryption [10], network intrusion detection systems [30], and prototyping [31]. These applications leverage the configurability of FPGAs to compensate for continuously changing environments. Network switches and routers have long used FPGAs for a variety of purposes [32], [33], [34], [35], [36], [37], but recent work [38], [39] has extended that functionality to include LAN and WAN applications. For cluster computing, moving the FPGA from the switch into the network interface has two distinct advantages. First, for applications that use the FPGA as a computer accelerator, the I/O bus is a much better interconnect than the network link. Second, it brings

the FPGA into the protection domain of the node. Thus, issues such as access controls can be addressed by the node operating system. In addition, this work differs by focusing on the implications of FPGAs in the network for a parallel computing system rather than on WAN or LAN style applications.

VII. CONCLUSIONS

Cost-effectiveness is of particular concern in the Beowulf community because Beowulf Clusters were introduced as a low-cost alternative to traditional supercomputers. Although previous work established the performance advantage of an INIC feature, potential users of the resulting adaptable computing cluster are very sensitive to cost. Any extension to this architecture must provide a performance improvement without imposing a significant cost penalty if it is to be accepted. Consequently, a complete analysis of the INIC must include some characterization of its cost-effectiveness. In this paper, a complex quantitative measure of cost-effectiveness for Beowulf-class machines was introduced, specifically incorporating the nonlinear aspects of a cluster's cost.

Two important results have been established. First, cost-effectiveness was considered for three representative applications over a range of basic hardware costs and cluster sizes. The prototype for the INIC architecture would not be considered cost-effective. It offered significant performance improvements in many cases; however, these were far outweighed by the additional cost for all applications except PNN image classification. The next generation architecture does have the potential to be cost-effective. To achieve this potential, an INIC must be closer to the theoretical performance capabilities while minimizing cost. Section V presents a design for an INIC that would offer significantly higher performance than the prototype and would be significantly lower cost. Second, the added complexity of the cost model is not just academic. Cluster costs are distinctly non-linear and this fact must be accounted for when comparing technologies. In this effort, the complex cost model was used to indicate when to switch from an ordinary cluster to an adaptable computing cluster.

ACKNOWLEDGMENTS

K. Underwood was supported by a NASA GSRP Fellowship under grant number NGT5-85. Additional support for this work was received from the National Science Foundation under NSF

Grant EIA-9985986. Opinions expressed are those of the authors and not necessarily those of the Foundation.

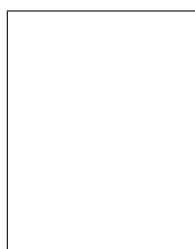
We also gratefully acknowledge the reviewers for their input to improve this paper and our colleagues who were kind enough to offer their suggestions for improvement of the text.

REFERENCES

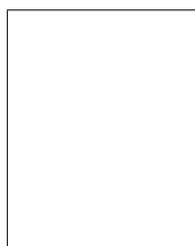
- [1] Walter B. Ligon, Scott P. McMillan, Greg Monn, Fred Stivers, Kevin Schoonover, and Keith D. Underwood, "A re-evaluation of the practicality of floating-point on FPGAs," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 1998.
- [2] Gerhard Lienhart, Andreas Kugel, and Reinhard Manner, "Using floating-point arithmetic on FPGAs to accelerate scientific N-Body simulations," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.
- [3] Keith D. Underwood, Walter B. Ligon, and Ron R. Sass, "Analysis of a prototype intelligent network interface," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 7-8, pp. 751–777, 2003.
- [4] Keith D. Underwood, *An Evaluation of the Integration of Reconfigurable Hardware with the Network Interface in Cluster Computer Systems*, Ph.D. thesis, Clemson University, Aug. 2002.
- [5] Keith D. Underwood, Walter B. Ligon, and Ron R. Sass, "Extension of the beowulf cluster system architecture with an intelligent network interface," Submitted, 2003.
- [6] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994.
- [7] Dilip Sarkar, "Cost and time-cost effectiveness of multiprocessing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 6, pp. 704–712, June 1993.
- [8] Babak Falsafi and David A. Wood, "When does dedicated protocol processing make sense?," Tech. Rep. CS-TR-1996-1302, Computer Sciences Department, University of Wisconsin–Madison, 1996.
- [9] David A. Wood and Mark D. Hill, "Cost-effective parallel computing," *IEEE Computer*, vol. 28, no. 2, pp. 69–72, 1995.
- [10] Peter Bellows, Venkatesh Bhaskaran, Jaroslav Flidr, Tom Lehman, Brian Schott, and Keith D. Underwood, "GRIP: A reconfigurable architecture for host-based gigabit-rate packet processing," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.
- [11] Ranjesh G. Jaganathan, Keith D. Underwood, and Ron Sass, "A configurable network protocol for cluster based communications using modular hardware primitives on an intelligent NIC," in *Proceedings of the 2003 Conference on Supercomputing*, Nov. 2003.
- [12] Matteo Frigo and Steven G. Johnson, "FFTW: An adaptive software architecture for the FFT," in *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Seattle, WA, May 1998, vol. 3, pp. 1381–1384.
- [13] David Bailey, Tim Harris, William Saphir, Rob van der Wijngaart, Alex Woo, and Maurice Yarrow, "The NAS parallel benchmarks 2.0," Tech. Rep. NAS-95-020, NASA, Dec. 1995.
- [14] Ramesh C. Argarwal, "A super scalar sort algorithm for RISC processors," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, June 1996, pp. 240–246.
- [15] D. R. Spect, "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109–118, 1990.
- [16] E. Parzen, "On the estimation of a probability density function and mode," *IEEE Transactions Electronic Computers*, vol. EC-16, pp. 309–319, 1967.

- [17] S. R. Chettri and R. F. Crompt, "Probabilistic neural network architecture for high speed classification of remotely sensed imagery," *Telematics and Informatics*, vol. 10, no. 3, pp. 187–198, 1993.
- [18] Andreas Dandalis, Viktor K. Prasanna, and Jose D. P. Rolim, "An adaptive cryptographic engine for IPSec architectures," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 2000, pp. 132–141.
- [19] SRC Computers, Inc., "MAP processor," Aug. 2002, From webpage at <http://www.srccomp.com/>.
- [20] Nanette J. Boden, Danny Cohen, Robert E. Felderman Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su, "Myrinet: A gigabit-per-second local area network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, Feb. 1995.
- [21] David E. Culler, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Brent Chun, Steven Lumetta, Alan Mainwaring, Richard Martin, Chad Yoshikawa, and Frederick Wong, "Parallel computing on the Berkeley NOW," in *9th Joint Symposium on Parallel Processing*, Kobe, Japan, 1997.
- [22] Compaq, "Compaq Servernet II SAN interconnect for scalable computing clusters," June 2000, From Whitepaper found at <http://www.compaq.com/support/techpubs/whitepapers/tc000602wp.html>.
- [23] Babak Falsafi and David A. Wood, "Scheduling communication on an SMP node parallel machine," in *Proceedings of Third International Symposium on High Performance Computer Architecture*, San Antonio, Texas, USA, Feb. 1997.
- [24] S. K. Reinhardt, J. R. Larus, and D. A. Wood, "Tempest and Typhoon: User-level shared memory," in *International Conference on Computer Architecture*, Chicago, Illinois, USA, Apr. 1994, pp. 260–267.
- [25] Marcel-Cătălin Roșu, Karsten Schwan, and Richard Fujimoto, "Supporting parallel applications on clusters of workstations: The intelligent network interface approach," in *Proceeding of the 6th International Symposium on High Performance Distributed Computing (HPDC 97)*, 1997.
- [26] Shinji Sumimoto, Hiroshi Tezuka, Atsushi Hori, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa, "The design and evaluation of high performance communication using a Gigabit Ethernet," in *International Conference on Supercomputing*, Rhodes, Greece, June 1999, pp. 260–267.
- [27] Yvonne Coady, Joon Suan Ong, and Michael J. Feeley, "Using embedded network processors to implement global memory management in a workstation cluster," in *Proceedings of The Eighth IEEE International Symposium on High Performance Distributed Computing*, Redondo Beach, California, USA, Aug. 1999.
- [28] Todd Mummert, Corey Kosak, Peter Steenkiste, and Allan Fisher, "Fine grain parallel communication on general purpose LANs," in *Proceedings of 1996 International Conference on Supercomputing (ICS96)*, Philadelphia, PA, USA, May 1996, pp. 341–349.
- [29] Mark Jones, Luke Scharf, Jonathan Scott, Chris Twaddle, Matthew Yaconis, Kuan Yao, and Peter Athanas, "Implementing an API for distributed adaptive computing systems," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 1999, pp. 222–230.
- [30] R. Franklin, D. Carver, and B. L. Hutchings, "Assisting network intrusion detection with reconfigurable hardware," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.
- [31] David C. Hoffmeister and Patrick W. Dowd, "An fpga-based network interface for wdm gigabit networks," in *Proceedings of the 1998 Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, September 1998.
- [32] Apostolos Dollas, Dionisios Pnevmatikatos, Nikolaos Aslanides, Stamatios Kavvadias, Euripedes Sotiriades, Sotirios Zogopoulos, Kyprianos Papademetriou, Nikolaos Chrysos, and Konstantinos Harteros, "Architecture and applications of PLATO, a reconfigurable active network platform," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2001.
- [33] Jason R. Hess, David C. Lee, Scott J. Harper, Mark T. Jones, and Peter M. Athanas, "Implementation and evaluation of

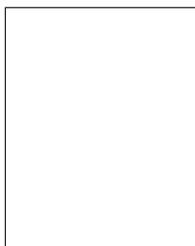
- a prototype reconfigurable router,” in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 1999, pp. 44–50.
- [34] John W. Lockwood, Jon S. Turner, and David E. Taylor, “Field programmable port extender (FPX) for distributed routing and queueing,” in *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, Napa Valley, CA, April 2000, pp. 137–144.
- [35] John W. Lockwood, Naji Naufel, Jon S. Turner, and David E. Taylor, “Reprogrammable network packet processing on the field programmable port extender (fpx),” in *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, Napa Valley, CA, April 2001, pp. 87–93.
- [36] John T. McHenry, Patrick W. Dowd, Frank A. Pellegrino, Todd M. Carrozzi, and William B. Cocks, “An FPGA-based coprocessor for ATM firewalls,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, April 1997, pp. 30–39.
- [37] Chun-Chao Yeh, Chun-Hsing Wu, and Jie-Yong Juang, “Design and implementation of a multicomputer interconnection network using FPGAs,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, April 1995, pp. 30–39.
- [38] Todd S. Sproull, John W. Lockwood, and David E. Taylor, “Control and configuration software for a reconfigurable networking platform,” in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.
- [39] David E. Taylor, Jonathan S. Turner, John W. Lockwood, Todd S. Sproull, and David B. Parlour, “Scalable IP lookup for internet routers,” *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 21, no. 4, pp. 522–534, May 2003.



Keith D. Underwood received the BS and PhD degrees in computer engineering from Clemson University in 1995 and 2002, respectively. He is currently a senior member of the technical staff at Sandia National Laboratories. His research interests include cluster computing, programmable network interfaces, and the role of reconfigurable computing in high performance computing systems. He is a member of the IEEE, IEEE Computer Society, and the ACM.



Walter B. Ligon, III received his Ph.D. in Computer Science from the Georgia Institute of Technology in 1992 and is currently an Associate Professor of Computer Engineering at Clemson University. At Georgia Tech he developed the Reconfigurable Architecture Workbench and also was part of a team that developed system software and programming tools for the SPOCK parallel processor. Currently he is the director of the Parallel Architecture Research Laboratory at Clemson. Current projects include the Parallel Virtual File System (PVFS), Coven parallel programming framework, Intelligent NIC, and Beowulf System Software project. His primary research interests are in computer architecture, parallel processing, operating systems, and compiler design.



Ron R. Sass received a B.S. in Computer Science and Engineering from the University of Toledo and an M.S. and Ph.D. in the same from Michigan State University. His dissertation investigated matrix-based restructuring compiler techniques for clusters of workstations. He is an assistant professor in the Holcombe Department of Electrical and Computer Engineering at Clemson University where his principle research focus is in reconfigurable computing and networking. He is a member of the IEEE, IEEE Computer Society, IEEE Communications Society, and the ACM.