Cost Effectiveness of an Adaptable Computing Cluster*

(Revised)

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

Parallel Architecture Research Lab
Holcombe Department of Electrical and Computer Engineering
Clemson University
105 Riggs Hall
Clemson, SC 29634-0915
{keithu, rsass, walt}@parl.clemson.edu

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. A quantitative idea of cost-effectiveness is formulated to evaluate computing technologies. Here, cost-effectiveness is considered in the context of two applications: the 2D Fast Fourier Transform (2D-FFT) and integer sorting.

1. Introduction

Beowulf-class computers (cluster computers based on COTS hardware and open 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 for a class of problems, such as the Fast Fourier Transform, that depend on a high performance interconnect to achieve scalability. 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 consisting of Field Programmable Gate Arrays (FPGAs). Reconfigurable Computing has been shown to be effective at providing speedups for an assortment of applications by relying on the ability of programmable hardware to instantiate many custom functional units to implement data flow computations. Unfortunately, the earliest incarnations of RC lacked the gate counts needed to support floating-point applications. Now, the underlying FPGA technology has matured to the point that it can be competitive with workstations for some floating-point applications [11]. The biggest challenge now facing reconfigurable computing is cost. Historically, high density FPGA devices have been expensive. Reconfigurable computing cards based on these devices have been low-volume specialty parts further contributing 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 a high-volume application arises.

It is difficult to find high-volume applications for RC because of its lack of general applicability. For some applications, it offers impressive speedup while for others it offers nothing. 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 intercon-

*This work was supported in part by the National Science Foundation under NSF Grant EIA-9985986

†Mr. Underwood is supported by a NASA GSRP Fellowship under grant number NGT5-85

SC2001 November 2001, Denver

nect) of the PCI bus often decimates the speedups that RC can offer. Further complicating the applicability to commodity clusters, a reconfigurable computing card and a network card would typically share a single PCI bus. If an application performs significant communication, data would have to make three passes over the PCI bus — network interface to host memory to reconfigurable computing card to host memory. In [17], an Intelligent Network Interface Card (INIC) based on a high performance NIC and reconfigurable computing was proposed. The network interface proved to be an excellent place to exploit reconfigurable computing.

When considering an architectural modification, it is important to evaluate both the performance and cost implications of the change. Traditional measures such as speedup and isoefficiency exist to analyze the performance of an architecture. The introduction of Beowulf clusters was prompted by a qualitative assessment of cost-effectiveness. Here, a quantitative measure of cost-effectiveness is formulated so that break-even points for a technology can be assessed. Cost-effectiveness is a function of cost and performance with a complex model of cost rather than the traditional linear cost assumption applied to clusters.

The architecture under consideration is discussed briefly in Section 2. Two applications and their performance characterizations are presented in Section 3. Section 4 compares the cost-effectiveness of a cluster with and without the enhancements to the network interface. Section 5 then presents a cost-effective INIC design. The paper closes with related works in Section 6 and conclusions in Section 7.

2. ACC System Architecture

The goal of Clemson's ACC project is to explore architectural enhancements to Beowulf clusters without reducing the cost-effectiveness of the system. Figure 1 illustrates the core of one such enhancement. Whereas a traditional NIC simply buffers data in a memory (or FIFO) between the host and network, an INIC inserts reconfigurable logic along the datapath. The reconfigurable logic could then be used in a range of modes including:

- *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. The 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 3 and such things 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 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.

Prototype System

The ACC experimental platform is an 16-node Beowulf running the Scyld Linux distribution. Each node contains a 32-bit PCI motherboard with a 1GHz Athlon and 512 MB of RAM. On the PCI system bus is a SysKonnect PCI Gigabit Ethernet NIC and a Fast Ethernet NIC. Eight systems also include an ACEII card. The ACEII has an onboard PCI bus attaching a μSPARC processor and a PCI Mezzanine (PMC) slot to the FPGAs. The PMC slot is populated with an Alta Technologies PMC Gigabit NIC based on the Packet Engines Hamachi chipset. The ACEII is a reconfigurable computing board from TSI TelSys. A simplified block diagram of the board is shown in Figure 2.

The prototype used is not the ideal architecture as it has a few deficiencies that prevent it from achieving its full potential as an Intelligent NIC. These include a single bus on the card for all data traffic, 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 the networking functions. Theoretically, an Intelligent NIC would be implemented as a single chip with external RAM

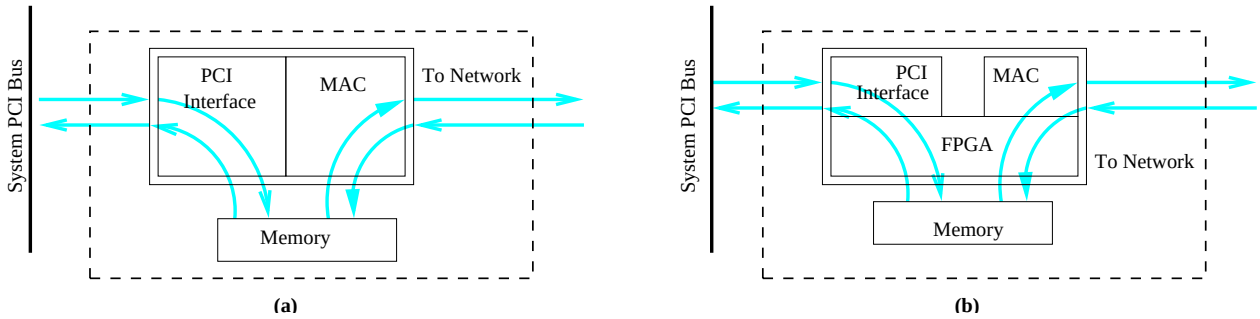


Figure 1. A comparison of (a) a traditional NIC and (b) the proposed architectural extension to form an INIC

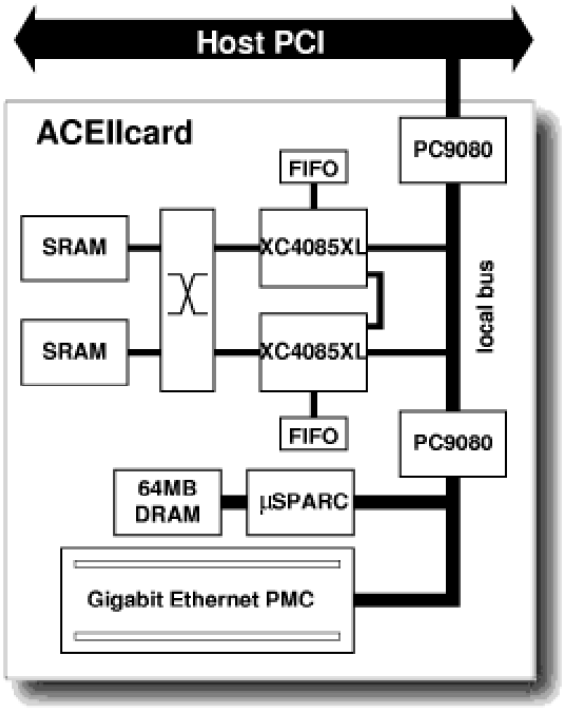


Figure 2. block diagram of an ACEII card

— similar to modern high performance NICs. Section 3 predicts the performance achievable with a next generation Intelligent NIC and addresses the performance achievable with our prototype INIC.

3. Applications

Two of the most important factors of any architecture are the performance of the users' applications on the architecture and the cost to achieve that performance. Two applications, a 512×512 2-D Fast Fourier Transform (2D-FFT) and integer sorting, were chosen for our preliminary evaluations of the architecture. These two applications were chosen because they can be implemented on the prototype and they emphasize the capabilities of the proposed architecture to integrate computational tasks with communication operations. For both applications, our architecture-specific implementation is a derivation of the standard parallel implementation taking advantage of 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, the implementations of the applications, and the performance of the applications. For further details, refer to [17]. 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. When there is ambiguity, sequences are numbered. Also note that each diagram shows a single network transaction. This is one of many concurrent transactions in the cluster.

3.1. 2D-FFT

One application under consideration is the two dimensional Fast Fourier Transform. The baseline parallel and serial implementations use the highly optimized Fastest Fourier Transform in the West (FFTW) package[9]. 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:

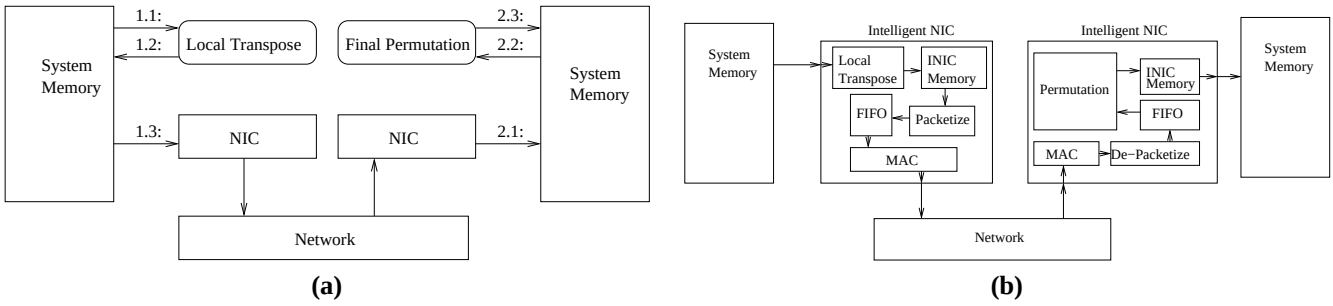


Figure 3. (a) block diagram of the transpose algorithm used in the FFTW package; (b) data manipulation on the network data stream as it passes through the INIC

❶ compute the 1D-FFT for each row

❷ transpose the matrix (redistribution of data)

❸ compute the 1D-FFT for each row

❹ 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 would require). Furthermore, the communication protocol used can be customized to the specific application since each node knows exactly how much data will be sent to and received from every other node.

3.2. Integer Sorting

The second application considered, integer sorting, is a common benchmark application. Although some benchmarks involve non-uniformly (often Gaussian [2]) distributed keys, synthetically generated and uniformly distributed keys are used for this discussion. This is a well-established precedent allowing 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 of integer sort first bucket sorts the data into buckets that fit well in the processor

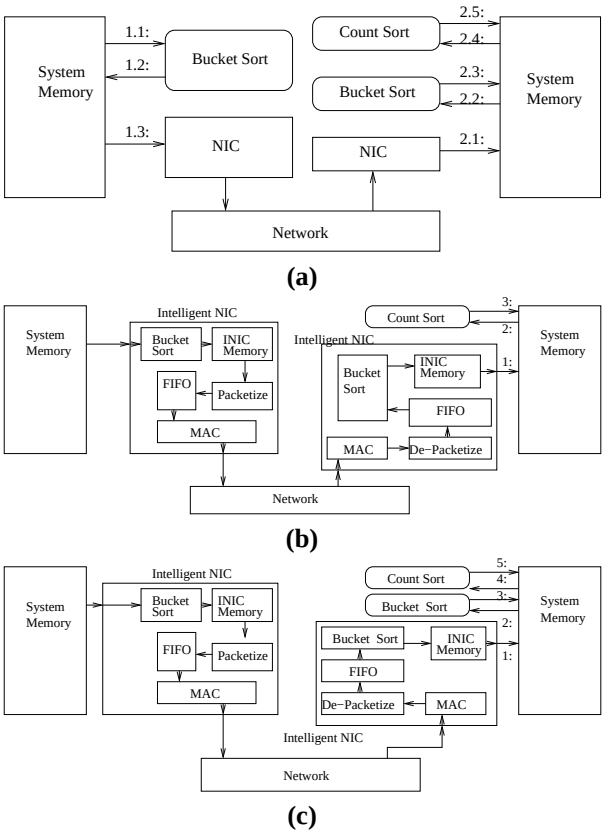


Figure 4. block diagrams of parallel integer sort implementations on: (a) a standard cluster with Gigabit Ethernet; (b) an INIC; (c) the prototype INIC with limited resources

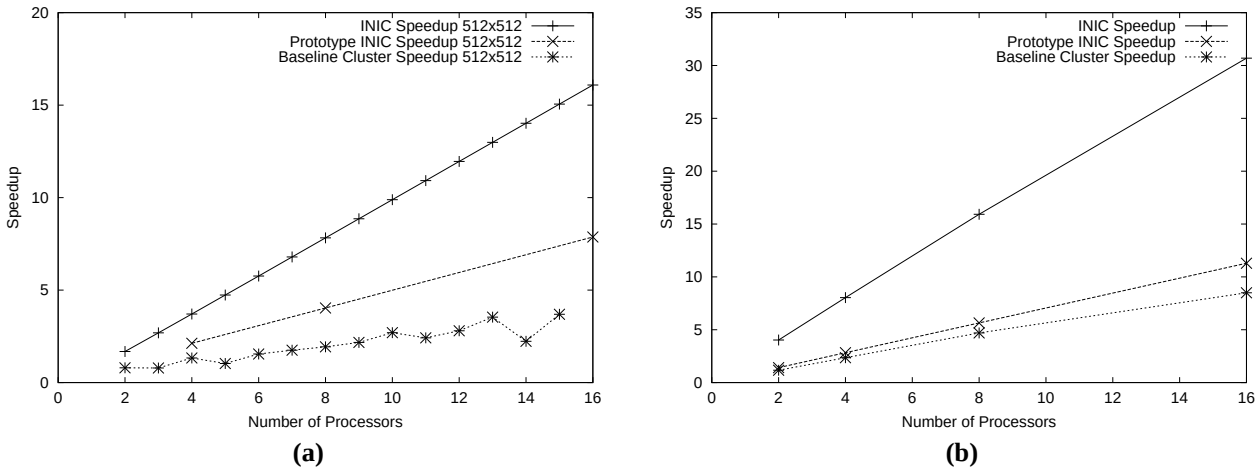


Figure 5. Potential Intelligent NIC speedups are shown with potential speedups for the prototype INIC and a baseline cluster (based on Gigabit Ethernet). (a) shows speedups for a 512×512 FFT (b) shows speedups for a 32M key integer sort

cache. These buckets are then sorted with Count Sort as in [1]. For the parallel implementation, a distributed memory to distributed memory sort over a power of 2 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 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 [1]. 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 Subsection 3.3, this can still provides a performance benefit.

Figure 4(b) and (c) shows 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 overhead[1] for starting a send. Hence, on the sending side, the INIC handles all of the computation and protocol processing leaving the processor free to other tasks such as disk I/O. On the receiving side, the bucket sort can be done as data is received[2]. As minimum thresholds are reached, data is transferred to the host. After all data is received, each bucket is sorted with Count Sort.

3.3. Performance Analysis

Figure 5 compares potential INIC speedups with speedups of a prototype INIC implementation and the measured performance of a baseline (Gigabit Ethernet) cluster. Results are measured for the prototype implementation of the FFT an estimated (based on measured performance for the FFT) for the integer sort implementation. Speedups are relative to a single node without a reconfigurable computing card. This is a reasonable comparison for FFT because in the serial implementation, the transpose is not performed. 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.

Figure 5 shows that the INIC has great potential to accelerate the two applications under consideration. Figure 5(a) indicates that the prototype INIC will offer better perfor-

[1] This 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.

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

mance 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 5(a) also indicates that an INIC has the potential for linear speedup up to 16 processors but that 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 5(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.

4. 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. Achieving the results presented in Section 3 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 using 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 measures 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-effectivenes, E, is defined as a ratio of the ratios of price to performance for two technologies, or:

$$E = \frac{\frac{C_1}{Speedup_1}}{\frac{C_2}{Speedup_2}}. \quad (1)$$

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 known. The INIC based cluster 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:

$$C_{Cluster}(N) = N \times C_{Node} + C_{Net}(N) \quad (2)$$

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:

$$C_{Net}(N) = B(N) \times C_{SwitchIncrement} + C_{SwitchBase} \quad (3)$$

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:

$$B(N) = \left\lceil \frac{N}{SizeofSwitchIncrement} \right\rceil \quad (4)$$

Equation 4 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:

$$C_{Node} = C_{SerNode} + C_{NetworkAdapter} \quad (5)$$

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

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

or,

Item	Cost
C_{RC}	\$7500
C_{Node}	\$2000
$C_{SerNode}$	\$1500
$C_{SwitchIncrement}$	\$10000
$C_{SwitchBase}$	\$8000

Table 1. December 2000 costs for a prototype INIC cluster

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

In turn, this can be applied to Equation 2 to get the cost model of an INIC enhanced cluster.

$$C_{INICCluster}(N) = N \times (C_{Node} + C_{RC}) + C_{Net}(N) \quad (8)$$

Table 1 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.

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 INIC enhanced 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 (INIC and Gigabit Ethernet) to a serial implementaion. In turn, $E_{Cluster}$ will be used to refer to comparisons between the clusters. Generally speaking, E will be defined on a per-application basis. E_{Serial} is a special case in which the speedup is one; hence, it can be defined as:

$$E_{Serial} = \frac{C_{SerNode}}{\frac{C_{technology}(N)}{Speedup(N)}} \quad (9)$$

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

Figure 6 compares E_{Serial} for three technologies: a baseline cluster, a cluster enhanced with the prototype INIC, and a theoretical INIC. For C_{RC}, \$7500 was used for the prototype INIC and \$1000 (based on the cost determined in Section 5) was used for the next generation INIC. The prototype INIC is less cost-effective due both to its high cost and relatively low performance increase; however, the theoretical INIC can achieve a cost-effectiveness near one for the integer sorting application.

While E_{Serial} defines the relative cost per unit speedup, Figure 6 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.

$$E_{Cluster} = \frac{\frac{C_{Cluster}(N)}{Speedup_{Cluster}(N)}}{\frac{C_{INICCluster}(N)}{Speedup_{INICCluster}(N)}}. \quad (10)$$

Using Equation 10 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 7 compares the relative cost-effectiveness of three implementations of two applications. The baseline cluster always has a cost-effectiveness of one relative to itself. While the prototype INIC is less cost-effective than the baseline cluster, the ideal INIC achieves significantly better cost-effectiveness in many cases.

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 its cost effectiveness. Figure 8 addresses this issue by illustrating the impacts of switch cost and INIC cost on the relative cost-effectiveness of a fixed size (sixteen node) INIC enhanced cluster. Figure 8 also provides 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, \$1000, to the cost of a Gigabit Ethernet switch for the prototype, \$28000. Similarly, C_{RC} was chosen to range from a minimal cost of \$100 to the prototype INIC cost, \$7500.

For the purposes of Figure 8, 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, both 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

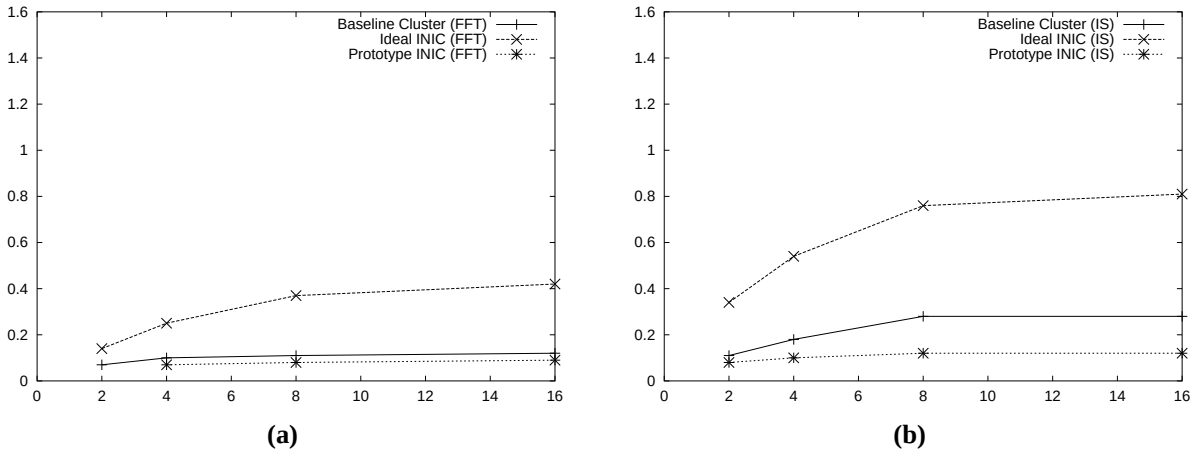


Figure 6. E_{Serial} of three possible implementations for **(a)** 2D-FFT and **(b)** Integer Sort

to consider the graphs without the need to vary node cost or performance.

The breakpoint between "cost-effective" and "not cost-effective" illustrated in Figure 8 occurs when $E_{Cluster} = 1$. From the figure, it is clear that an INIC can be cost-effective for the applications under consideration at the right cost. The additional performance achievable with a theoretical INIC 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 cost-effective 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 of the prototype INIC is significantly constrained by the single-bus architecture employed. Section 5 will discuss a design capable of achieving the theoretical INIC performance while constraining C_{RC} to $\$1000$.

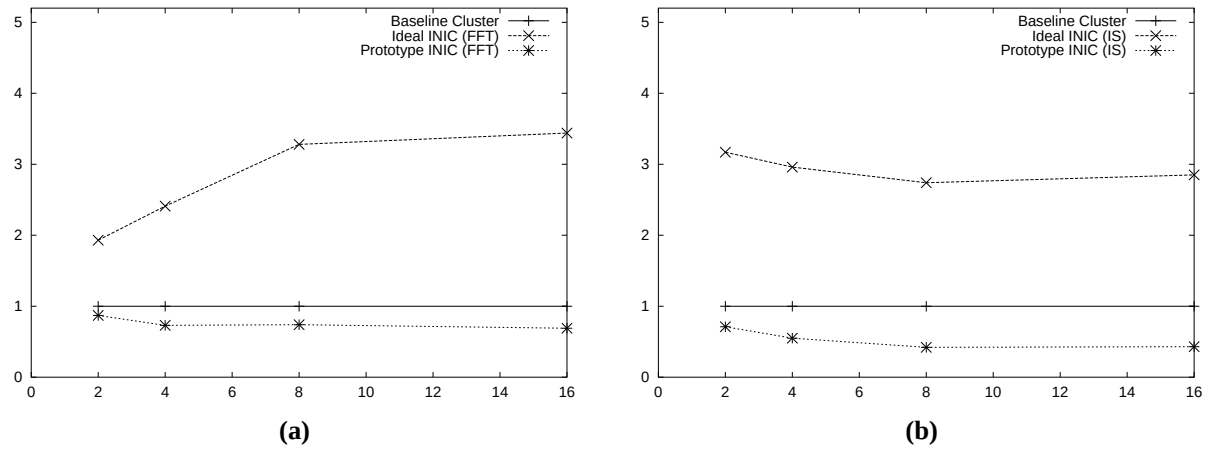
5. Cost-Effective INIC Design

In Section 3, Figure 5 references the potential speedup of an Intelligent NIC. This potential is based on the use of Gigabit Ethernet for the network fabric. Based on the preceding cost analysis, designing a cost-effective INIC requires careful attention to performance and cost. This section presents a design to achieve the performance of the ideal INIC built for a Gigabit Ethernet backplane presented in Section 3 while minimizing development and production costs. The proposed design is illustrated in Figure 9.

The first objective is achieving the full potential performance of the INIC architecture. This requires that at least two gigabits of bandwidth be available between the FPGA device(s) and the MAC. Simultaneously, there must be two gigabits 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.

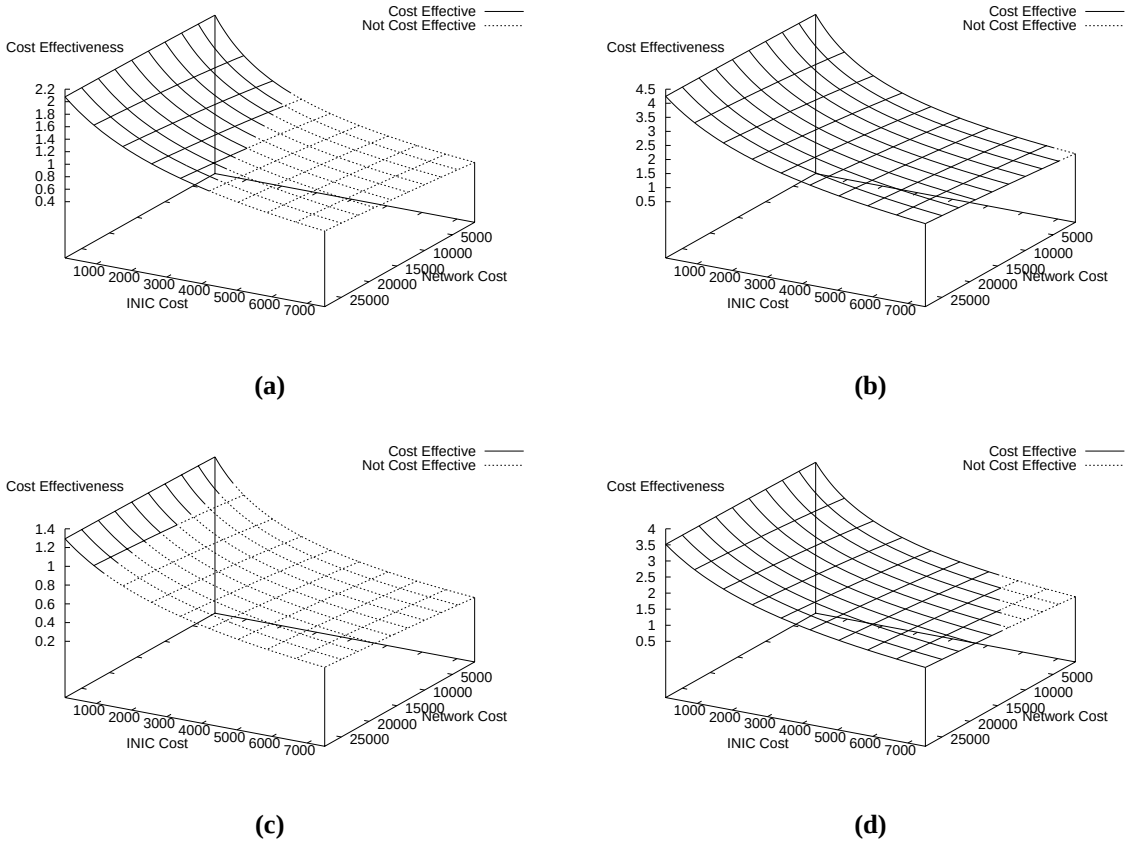
To achieve two gigabits of bandwidth to the host memory, 64-bit 66-MHz PCI will be required. It is possible to achieve two gigabits of bandwidth between the FPGA(s) and the network using 64-bit 66-MHz PCI, but it is undesirable. Commodity chipsets are available (such as the Vitesse XMAC-II VSC8840) which 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 1000 provides more logic than 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 data for a packet must be kept in the SRAM until an acknowledgment for that packet is received. It would be better to have a separate communications buffer that could hold data that had been transmitted while waiting on 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



(a) **(b)**

Figure 7. $E_{Cluster}$ of three possible implementations for **(a)** 2D-FFT and **(b)** Integer Sort



(a) **(b)**

(c) **(d)**

Figure 8. Impact of cost trends on $E_{Cluster}$ for **(a)** 512×512 FFT on the prototype INIC; **(b)** 512×512 FFT on a theoretical INIC; **(c)** 32M key integer sort on the prototype INIC; **(d)** 32M key integer sort on a theoretical INIC

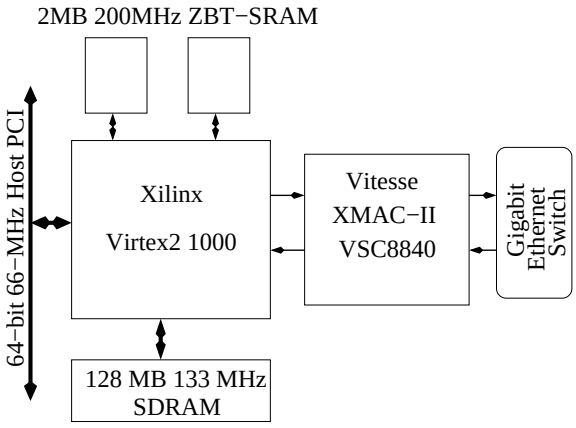


Figure 9. a cost-effective INIC

would be used per node in the cluster). Static allocation is not strictly necessary and removing this constraint would reduce the amount of communication buffer needed. Fortunately, the memory needed for this type of access is inexpensive and so the hardware design can be simplified by allowing static allocation. Accesses to this buffer will be bursty and sequential and will require 4 gigabits of sustainable bandwidth. Memory needs for the computation buffer will be dependent on the application, so 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.

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 will provide 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 SysKonnect in 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 is the same commodity RAM used in modern desktop machines so 128 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 9, includes two banks of 2 MB each.

Part	Qty	Unit Cost	Total
Virtex-II 1000	1	$323.00	$323.00
Gigabit Ethernet MAC	1	$62.50	$62.50
2MB 200MHz SRAM	2	$140.00	$280.00
128MB 133MHz SDRAM	1	$26.00	$26.00
Support Parts	1	$52.20	$52.20
Board Fabrication	1	$135.00	$135.00
Total			$878.70

Table 2. parts and cost for an INIC

Table 2 provides a table of components and costs for the design proposed in Figure 9. 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 the 10 boards per month more characteristic of current reconfigurable platforms increases fabrication cost to $400 per board, an increase of thirty percent in overall board cost[3]. More importantly, the non-recoverable engineering (NRE) costs associated with each new board design must be amortized across all of the cards that will be 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 as it could be marketed for cluster applications, traditional reconfigurable computing applications, and network encryption applications such as IPSec[7].

The data in Table 2 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 (2001) 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 the same density) is only

[3]All cost information was provided courtesy of USC/Information Sciences Institute East.

$323.

6. Related Work

Much like this effort, SRC Computers, Inc.[15] seeks to employ 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 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 [3], used in the Berkeley NOW [6], and Compaq's Servernet-II [5] are demonstrating that cluster users are willing to pay for higher performance networks in their 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 on network interfaces. Research at the University of Wisconsin [8] 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 [13], Georgia Tech's VCM [14], RWCP's GigaE PM project [16], and the University of British Columbia's GMS-NP project [4] all use such a processor to accelerate distributed computing. Similarly, research at CMU explored hardware to augment ATM card to boost distributed programming speeds with a Hardware Assisted Remote Put (HARP)[12]. An INIC offers a potentially more cost-effective solution than these efforts because it provides the flexible computational abilities needed by these efforts and provides adequate resources to place application specific computation on the INIC.

Others have created clusters with reconfigurable cards in each node [10], but 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.

7. Conclusions

Cost-effectiveness, which adds a cost dimension to the traditional measure of speedup, was introduced as a method for evaluating architectures. It was then used to assess an extension to the traditional Beowulf Cluster architecture. Cost-effectiveness is of particular concern in the Beowulf community because Beowulf Clusters were introduced as a low-cost alternative to traditional supercomputers. Any extension to this architecture must provide a performance improvement without imposing a significant cost penalty if it is to be accepted.

It was discovered that the prototype for the INIC architecture would not be considered cost-effective. Although it offered significant performance improvements in many cases, these were far outweighed by the additional cost. The theoretical architecture, however, does have the potential to be cost-effective. To achieve this potential, an INIC must be closer to the theoretical performance capabilities. It is also important that it be relatively low cost. Section 5 presents a design for an INIC that would offer significantly higher performance than the prototype and would be significantly lower cost.

References

[1] R. C. Argarwal. A super scalar sort algorithm for RISC processors. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 240–246, June 1996.

[2] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Technical Report NAS-95-020, NASA, Dec. 1995.

[3] N. J. Boden, D. Cohen, R. E. F. 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), Feb. 1995.

[4] Y. Coady, J. S. Ong, and M. 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.

[5] 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.

[6] D. E. Culler, A. Arpaci-Dusseau, R. Arpaci-Dusseau, B. Chun, S. Lumetta, A. Mainwaring, R. Martin, C. Yoshikawa, and F. Wong. Parallel computing on the Berkeley NOW. In *9th Joint Symposium on Parallel Processing*, Kobe, Japan, 1997.

[7] A. Dandalis, V. K. Prasanna, and J. D. P. Rolim. An adaptive cryptographic engine for IPSec architectures. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 132–141, Napa Valley, CA, April 2000.

[8] B. Falsafi and D. 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.

[9] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, volume 3, pages 1381–1384, Seattle, WA, May 1998.

[10] M. Jones, L. Scharf, J. Scott, C. Twaddle, M. Yaconis, K. Yao, and P. Athanas. Implementing an API for distributed adaptive computing systems. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 222–230, Napa Valley, CA, April 1999.

[11] W. B. Ligon, S. P. McMillan, G. Monn, F. Stivers, K. Schoonover, and K. D. Underwood. A re-evaluation of the praticality of floating-point on FPGAs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 1998.

[12] T. Mummert, C. Kosak, P. Steenkiste, and A. Fisher. Fine grain parallel communication on general purpose LANs. In *In Proceedings of 1996 International Conference on Supercomputing (ICS96)*, pages 341–349, Philadelphia, PA, USA, May 1996.

[13] S. K. Reinhardt, J. R. Larus, and D. A. Wood. Tempest and typhoon: User-level shared memory. In *International Conference on Computer Architecture*, pages 260–267, Chicago, Illinois, USA, Apr. 1994.

[14] M.-C. Roşu, K. Schwan, and R. 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.

[15] I. SRC Computers. Map processor, May 2001. From webpage at http://www.srccomp.com/.

[16] S. Sumimoto, H. Tezuka, A. Hori, H. Harada, T. Takahashi, and Y. Ishikawa. The design and evaluation of high performance communication using a Gigabit Ethernet. In *International Conference on Supercomputing*, pages 260–267, Rhodes, Greece, June 1999.

[17] K. D. Underwood, R. R. Sass, and W. B. Ligon. A reconfigurable extension to the network interface of beowulf clusters. In to appear in *Proceedings of IEEE Cluster Computing*, Newport Beach, CA, October 2001.