

SCI and the Scalable Cluster Architecture Latency-hiding Environment (SCALE) Project

Alan George, William Phipps, Robert Todd, David Zirpoli, Kyla Justice,
Mark Giacoboni, and Mushtaq Sarwar

High-performance Computing and Simulation (HCS) Research Laboratory¹
Electrical Engineering Department, FAMU-FSU College of Engineering
Florida State University and Florida A&M University

Abstract

Future high-performance computing systems for both general-purpose and embedded applications must be “able”: scalable, portable, dependable, programmable, and affordable. The Scalable Cluster Architecture Latency-hiding Environment (SCALE) project underway in the HCS Research Laboratory hopes to contribute to this end by bridging the gap between the research worlds of parallel computing and high-performance interconnects by employing concurrent research methods that are both theoretical and experimental in nature. This paper provides an overview of the SCALE concepts and project, current developments and results in modeling and simulation tasks, current developments and results in experimental testbed tasks, as well as conclusions and future research plans.

1. Introduction

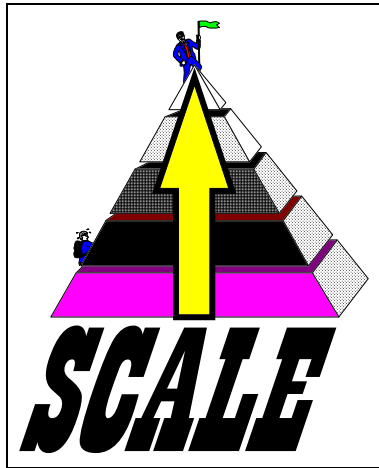
The goal of the SCALE research project at the HCS Research Lab is to investigate and develop a family of techniques by which cluster architectures can be constructed with latency-hiding, high-level parallel programming and coordination languages, and low-level lightweight communication protocols in a fashion that is open, portable, high-performance, distributed, fault-tolerant, and scalable. In particular, the goal is scalability across different interconnects supporting uniform memory access (UMA), non-uniform memory access (NUMA), or cache-coherent NUMA (CC-NUMA) shared memory as well as (and in tandem with) those with no remote memory access (NORMA) via message passing (MP). These scalable cluster architectures leverage the constantly rising levels of performance and dropping costs in commercial off-the-shelf (COTS) hardware including state-of-the-art work-

stations, high-performance interconnects, adapters, switches, single-board computers, etc.

The parallel architecture of a SCALE system is constructed by combining high-performance interconnects in a multilevel fashion. The scalable topology we propose is a cluster which starts with shared-bus, shared-memory symmetric multiprocessors (SMPs) with UMA. These SMPs are combined via the Scalable Coherent Interface (SCI) [SCI93] with NUMA and CC-NUMA across local-area distances, and then clusters of these SCI-based multiprocessors are combined via Asynchronous Transfer Mode (ATM) across metropolitan-area and wide-area distances. The software aspects of the SCALE system emphasize latency-hiding mechanisms including distrib-

uted-directory cache coherence, instruction and data prefetching, relaxed memory consistency, shared virtual memory, and multiple contexts via multithreading. High-level parallel programming and coordination language implementations must be developed to exploit latency-hiding mechanisms to achieve high-performance and scalability across multiple tiers in the SCALE system, such as multithreaded MPI and Linda. Low-level, lightweight communication protocols must be developed to exploit the low-latency and high-throughput potential of the underlying high-performance interconnects. This approach will achieve

flexibility and portability without the performance drain associated with TCP/IP, such as with the first multithreaded implementations of HCS_LIB (a lightweight parallel communications protocol developed by the HCS Lab) and Active Messages (originating from the University of California at Berkeley) for SCI and ATM/SONET (ATM over Synchronous Optical Network). While SCI and ATM are highlighted as target interconnects for this project's demonstration system, and in some sense repre-



¹ In January, Dr. George and the HCS Research Lab will join the *Department of Electrical and Computer Engineering* at the *University of Florida* located in Gainesville, Florida.

sent opposite ends of the high-performance interconnect spectrum, systems constructed with many other high-performance interconnects can also be addressed with this technology including Fibre Channel, Gigabit Ethernet, HIPPI, SuperHIPPI, Myrinet, etc.

Multiple levels of interconnects offer a number of distinct advantages for scalable, parallel machines. Like their fat-tree network brethren, the topology of SCALE systems affords different levels of throughput, latency, functionality, reliability, and versatility for varying performance and dependability demands. For example, two nodes in the same SCI cluster of a SCALE machine might typically communicate via shared-memory access for small exchanges or DMA transfers for large ones. However, if the source node finds its SCI link congested to a certain degree it may be more advantageous to communicate with its destination via its ATM link despite the inherent inferiority in peak throughput and minimal latency. With respect to dependability, by connected nodes to two or more of the interconnects in the multilevel interconnect fabric (e.g. SCI for local cluster communication and ATM for wide-area network communication), SCALE systems provide a degree of hardware redundancy such that node, link, port, and switch faults can be tolerated or masked and the system continues to function with a graceful degradation in performance. In fact, by considering link, port, and switch congestion as a form of fault, SCALE systems can be studied and developed to provide dynamic performance and dependability adjustments. Models, simulations, and analyses are needed to better understand how best to maximize these benefits in terms of cost.

Another key area of research related to the SCALE project is heterogeneous scheduling and computing. By using different types of parallel processors, processing elements, and interconnection paradigms, heterogeneous computing has the potential to maximize performance and cost-effectiveness for a wide variety of challenging scientific computing problems. Such applications typically have computation and communication requirements which not only vary between applications but dynamically within applications. The specialist parallelism approach of heterogeneous computing is often particularly attractive, where the processors and computers are scheduled for subtasks of the application based on their fit to the requirements of the subtask instead of merely using load balancing to spread out the problem evenly or proportionately. For example, those aspects of a program which are

inherently vector processing oriented would be mapped to a vector node within the virtual heterogeneous machine, those that are data parallel in nature might be mapped to a SIMD/SPMD node, etc. The suitability of the tasks must first be evaluated, and only then is load balancing used among selected machines for the final assignment based on these suitabilities. Research topics in heterogeneous computing are diverse and involve issues related to connectivity, bandwidth, granularity, high-level orchestration tools, and programming paradigms. Heterogeneous computing methods and tools like SmartNet [FREU93] can help to harness the dynamic performance requirements associated with sequential and parallel job sets.

The research, development and demonstration of the SCALE system will take place in terms of both modeling and simulation tasks and experimental testbed tasks. Researchers in the HCS Lab have constructed and verified some of the first fine-grain functional and statistical models of base SCI and are developing switch models and several proposed real-time SCI protocol extensions. These models will be leveraged and extended in support of the SCALE project. For example, the degree to which CC-NUMA can be effectively and efficiently supported with the distributed-directory, cache-coherence elements of the SCI protocol is not clear. Certainly the CC-NUMA afforded by SCI is likely to be capable of efficiently supporting several processors and even dozens, but the overhead associated is likely to prove a bottleneck amidst thousands of processors. Modeling and simulation of ATM/SCI/SMP architectures for SCALE will help determine the limits of effectiveness afforded by UMA, NUMA, CC-NUMA, shared virtual memory (SVM), and message passing (MP) within a global system address space perspective (see Figure 1). Similarly, network performance and dependability modeling with ATM/SCI/SMP holds the potential to characterize the optimum design methods with respect to topologies, bridge architectures, lightweight protocol mechanisms, etc. involved in a wide variety of SCALE systems. Model parameters can be injected not only from analytical representations but also from concurrent efforts in the measurements with the SCALE testbed. For example, SCALE models can be constructed with 1-Gbps SCI and 155-Mbps ATM in close cooperation with the testbed, and then extended to support 8-Gbps SCI and 2.4-Gbps ATM for studying the behavior of future SCALE components and systems.

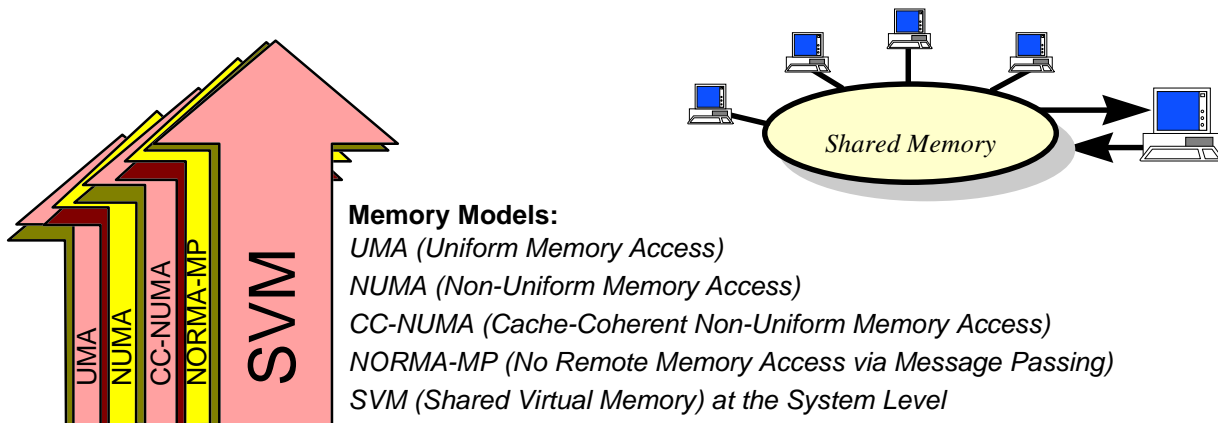


Figure 1. Shared Memory and SCALE

Another major thrust to the SCALE project is centered on experimental testbed research with existing and future facilities. For over a year the HCS Lab has had installed and operational one of the first and largest SCI cluster testbeds in the world. A number of upgrades are underway that bring the testbed up to more than twenty SPARC and UltraSPARC workstations (including eight dual-CPU SMP workstations), both 1.6- and 1.0-Gbps SCI adapters with a 4-port SCI switch, and 155-Mbps ATM adapters with several ATM switches connecting workstations spanning hundreds of miles (i.e. between the primary HCS Lab facility at the University of Florida in Gainesville and the satellite facility at the FAMU-FSU College of Engineering in Tallahassee). This ATM/SCI/SMP testbed is believed to be the first of its kind in the United States. Future plans include incremental steps toward the eventual realization of 8-Gbps SCI and 2.4-Gbps ATM in the SCALE system in a manner which supports LAN, MAN, and WAN connectivity and scalability.

Since the inception of the testbed facility in 1995, a number of software mechanisms and tools have been created by researchers in the HCS lab for the purpose of achieving parallel communication and coordination that can be exploited for SCALE. These tools include the first reliable, lightweight, multithreaded communications protocol and API for SCI/Sbus (HCS_LIB) as well as the first multithreaded SCI designs and implementations of Active Messages (HCS_AM), the Shared Virtual Memory coordination language Linda (HCS_LINDA), and the Message Passing Interface (HCS_MPI). While all of these parallel computing mechanisms, both for high-level parallel pro-

gramming and low-level, lightweight, high-speed communications, are still quite new and continue to be perfected, they have already been successful in achieving a reasonably high degree of parallel system efficiency [GEOR95, GEOR96]. These tools will be extended, leveraged, and complimented with mechanisms to design, develop, and demonstrate the SCALE system. A number of scalable, multithreaded, distributed/parallel programs with embedded granularity knobs are being developed for these demonstrations that are representative of the operations required in a wide variety of grand-challenge applications. Some of these include matrix algebra operations, fast Fourier transforms, beamforming, sorting, neural network training and processing, etc. In addition to the introduction of new software tools and applications for ATM/SCI/SMP parallel and distributed computing, the experimental testbed research results will, as mentioned previously, also be used to inject real-world data into the SCALE models so that performance and dependability studies can take place for next-generation SCALE systems concurrently with experimental testbed studies of state-of-the-art SCALE.

In summary, the SCALE project is designed to bridge the gap between the research worlds of parallel computing and high-performance interconnects by employing concurrent research methods that are both theoretical and experimental in nature. Future high-performance computing systems, for both general-purpose and embedded applications, must be "able": scalable, portable, dependable, programmable, and affordable. The SCALE project hopes to contribute to this end.

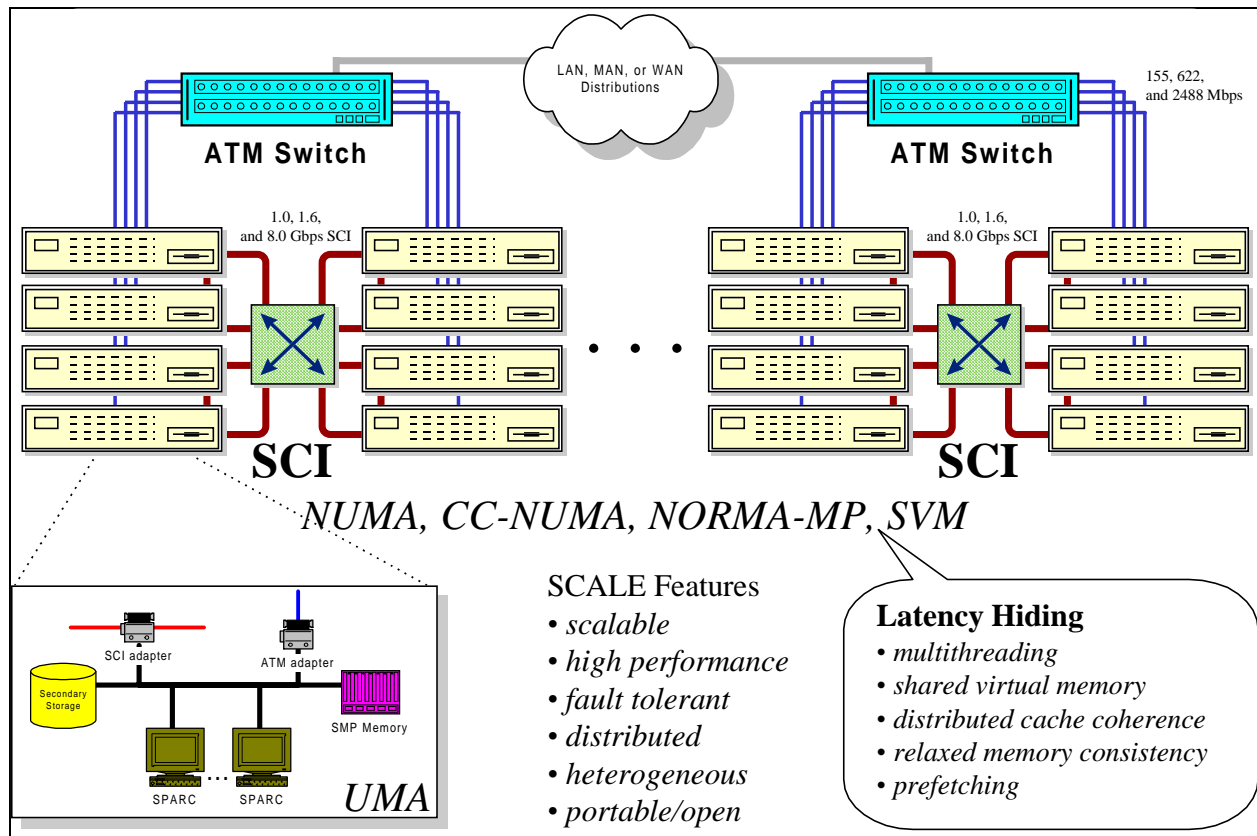


Figure 2. Conceptual Diagram of the ATM/SCI/SMP Architecture for SCALE

In the next section, an overview of the research tasks currently underway involving fine-grain network modeling and simulation related to the SCALE project is presented. In the third section an overview of the tasks completed and currently underway involving experimental testbed research is described, including lightweight, low-level communications protocols and tools, and high-level parallel coordination and programming language tools. The fourth section provides network performance measurements while the fifth section presents parallel performance measurements on a first-generation SCALE system. Finally, a set of conclusions is presented and future research plans are enumerated.

2. Modeling and Simulation

To complement the experimental and analytical elements of performance analysis in progress at the HCS Research Lab, modeling and simulation of both current and future testbed configurations is routinely performed. This work allows the gauging of expected system behavior while also showing the performance that might be achieved in the future. Several models based on the SCI

standard and related work have been built using the BONEs network simulator.

BONEs

The Block-Oriented Network Simulator (BONEs) Designer, produced by The Alta Group, is an integrated and highly-polished UNIX-based CAD tool for simulating the processes of an event-driven data transfer system [ALTA94]. It allows for a hierarchical, building-block perception of network model construction along with finite-state machine modeling and the importation of handwritten C/C++ code. BONEs was developed to model and simulate the flow of information represented by bits, packets, messages, or any combination.

BONEs is comprised of several parts all layered to create a structured environment for the development and simulation of network models. The seven major parts of BONEs, in chronological order of use in model development, are the Data Structure Editor (DSE), the Block Diagram Editor (BDE), the Finite State Machine Editor (FSM), the Primitive Editor (PE), the Symbol Editor (SE), the Simulation Manager (SM), and the Post Processor (PP). Figure 3 shows the parts of BONEs and their interaction with the library to produce the cohesive system.

These modules can all be used to construct a model of the desired system which is then evaluated using Monte Carlo, event-driven analysis techniques measuring performance of the system and displaying those metrics graphically.

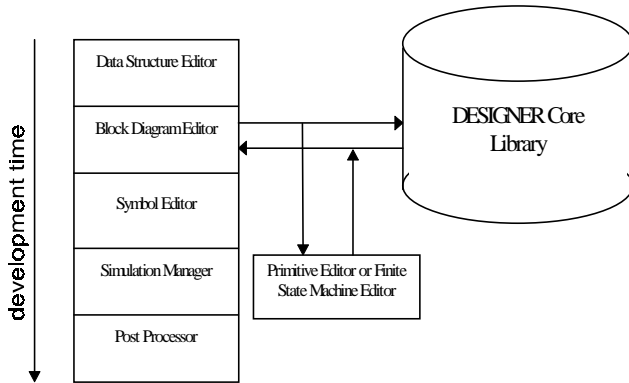


Figure 3: BONEs designer layout

SCI Models

Researchers at the HCS Lab have developed a fine-grain emulation model of SCI using BONEs. In its third revision, this model offers a graphical user interface during simulation to observe how data flows through the SCI network. Simulation is done at the packet level (as opposed to the symbol level) and assumes error-free communication and control. In order to quantify the packet handling limits of SCI, large systems can be created using basic node blocks and traffic sources, equipped with numerous parameters to vary individual performance. Table 1 shows the available traffic interarrival rate choices and Table 2 lists the destination choices. Figure 4 shows the top-level block diagram of the SCI emulation node model.

Table 1. Source Timing Parameters

Source Interarrival Rate	Description
Uniform	Request packets are generated at a uniform rate
Poisson	Request packets are generated with exponentially varying random delays
Available	Request packets are generated to fill empty output queue slots

The emulation model was designed to eventually implement all of the SCI standard using the standard's C-code for verification of critical operations. Currently, the emulation model implements all of the non-cache-coherent transactions except broadcasts. The packet format is identical to the standard and all packet types such

as *init*, *reset*, *idle*, and *sync* are included. For simulation purposes, the network is assumed to be stable and preinitialized. Node transmission follows the "pass transmission" protocol using only the low-go bits with initial scrubber go-bit injection. The nodes are designed with parameterized queue sizes and out-of-order packet transmission and retries.

Table 2. Request Destination Choices

Request Destinations	Description
Fixed	All packets are sent to a single responder to test congestion
Random	Each packet has an equally-likely chance of being sent to any node
Self	All requests are sent via SCI to the sender to test worst-case utilization
Downstream	Each packet is sent to its nearest neighbor on the ring for best case
Upstream	Each packet is sent to its farthest neighbor on the ring for worst case

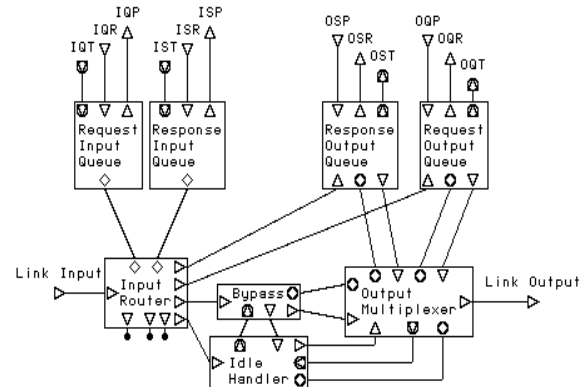


Figure 4. SCI Node Model

The host interface to the node was created to facilitate both synchronous and asynchronous sources and sinks as well as allowing two back-to-back nodes to act as a dual-ported SCI switch with dynamic routing tables. To estimate the performance of these nodes, certain estimated delays were used to simulate the hardware delays and transmission delays. Table 3 lists all modifiable delays.

Table 3. Delay Parameters

Delay Parameters	Description
Stripping Time	Time taken to determine target id and packet type for internal routing and queuing on the input queue
Routing Table Lookup	Time taken to search the generic routing table for the target id, i.e. interval routing or table routing
Host Interface	Delay taken to pass the address and data across the host interface, i.e. shared memory bus
Responder Latency	Delay for the responder to generate the correct response packet, i.e. memory access time
Bypass Delay	Delay to enter the bypass FIFO
Transmission	Time taken for the packet to propagate across a physical medium, i.e. fiber or parallel electric

SCI Switch Models

The emulation model has three switch models available with other designs pending. The simplest switch is a dual-ported bridge node in which one port of the bridge (input and output) is connected to one ringlet while the other port is connected to another ringlet. With the appropriate routing tables for either side, this node acts as an SCI agent and supports all transactions across the bridge. Figure 5 shows the top-level block diagram of the dual-ported bridge node in BONEs.

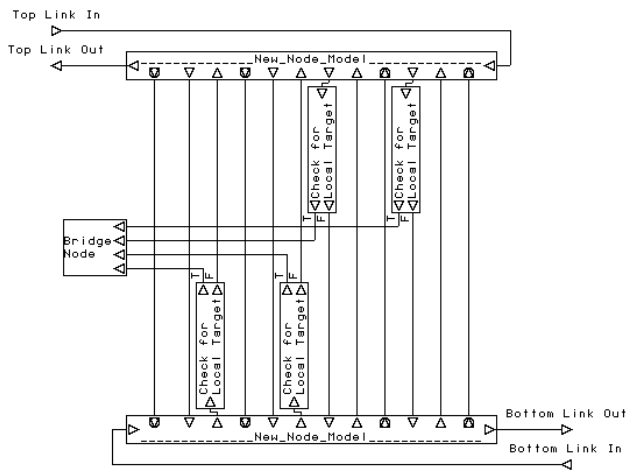


Figure 5. Dual-ported Node Model

Using these dual-ported agents, a scalable multiple-port switch can be created. The second available switch model uses an SCI ring as the internal switch fabric with one port of each agent residing on this ring while the other port is connected to other ringlets. Again, with the appropriate routing tables, this connectionless switch passes the

appropriate packets between connected ringlets via the internal switch fabric. The top-level block diagram of the connectionless switch is shown in Figure 6.

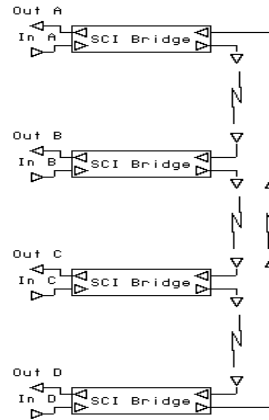


Figure 6. Dual-ported Nodes with an Internal Ringlet as the Switch Fabric

The third switch is the multi-service switch proposed by the Navy [KLIN93] which combines the multiple-port connectionless switch above with an internal crossbar and controller. The crossbar controller is addressable by any externally-connected node by residing on the internal switch fabric. To improve the performance of streaming data, the crossbar is switched to physically connect two external ringlets into a single larger ring. All of the switch models and descriptions are featured in [PHIP96]. Figure 7 shows the block diagram for the BONEs model of the multi-service switch. Future extensions to the SCI emulation model will include a shared-bus connectionless switch, broadcast support across switches, implementation of the initialization routines across switches, automatic routing table generation, and an Mbus host interface with DMA engine.

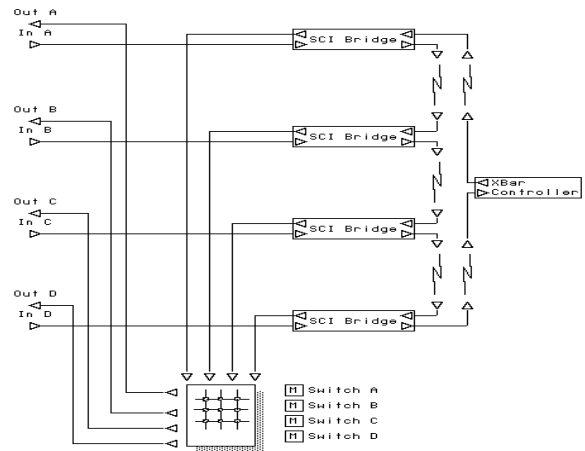


Figure 7. Multi-service Switch Model

In addition to modeling and simulation of base SCI, development is continuing on several real-time protocols based on SCI. Models have been developed and are currently being tested simulating the Preemptive Priority Queue [ANDE95] and TRAIN [SCOT95] protocols. Also, development is continuing on a Directed Flow Control protocol model [LACH96] and a 2-bit Priority protocol model [GUST96]. Preliminary results and detailed model descriptions are given in [TODD96].

3. Experimental Testbed Research

The HCS Lab is researching, developing, and studying new lightweight communication protocols and coordination languages to enhance portability and ease the writing of parallel programs. Some of these developments are: HCS_LIB, HCS_AM, and HCS_Linda. MPI was also used in development of parallel applications.

HCS_LIB

The lowest-level API available for use with the Dolphin SCI/Sbus-1 cards is a set of library routines called SCI_LIB [DOLP95]. This set of functions provides a level of abstraction to the programmer and facilitates the creation of shared-memory services and message-passing support without the programmer being overly concerned with the specific idiosyncrasies of the current revision of SCI drivers.

The SCI_LIB functions are shown in Table 4. The library consists of initialization, status, and closure functions, message-passing functions for creating, connecting, and releasing ports, and functions for shared-memory creation and destruction.

Table 4. SCI_LIB Summary

Initialization	Message-passing	Shared-memory
<i>sci_init_lib()</i>	<i>sci_create_receive_port()</i>	<i>sci_create_shm()</i>
<i>sci_close_lib()</i>	<i>sci_receive_port_connected()</i>	<i>sci_map_shm()</i>
<i>sci_lib_info()</i>	<i>sci_connect_transmit_port()</i>	<i>sci_unmap_shm()</i>
	<i>sci_read_msg()</i>	<i>sci_remove_shm()</i>
	<i>sci_write_msg()</i>	
	<i>sci_remove_receive_port()</i>	
	<i>sci_remove_transmit_port()</i>	

A higher-layer set of functions for accessing the SCI/Sbus-1 adapters in either shared-memory or message-passing fashion has been developed at the HCS Research Lab. Called HCS_LIB, it adapts the SCI_LIB functions to provide another level of abstraction for added convenience, flexibility, and dependability. Table 5 summarizes these functions.

Table 5. HCS_LIB Summary

Category	Function
Shared-memory	<i>safe_shared_memory_make()</i> <i>safe_shared_memory_map()</i>
Message-passing	<i>safe_create_receive_port()</i> <i>safe_connect_transmit_port()</i> <i>safe_wait_for_connection()</i> <i>send_block()</i> <i>get_block()</i> <i>send_broadcast()</i> <i>get_broadcast()</i>
Signaling	<i>service_signals()</i> <i>default_signal_handler()</i> <i>ignore_signal_handler()</i> <i>send_signal()</i> <i>send_signal_and_wait()</i> <i>set_signal_handler()</i> <i>barrier_sync()</i>
Initialization and Termination	<i>cluster_init()</i> <i>safe_exit()</i>

The majority of shared-memory and message-passing functions in HCS_LIB are simply extensions of SCI_LIB to provide more versatility and fault tolerance in the programming environment. These functions allow a programmer to reference an SCI node as an index into a node array rather than by specific node numbers while also providing time-out support on failed mapping and creation attempts. The HCS_LIB message-passing functions also include broadcast support which follows a spanning tree structure with the node initiating the broadcast transmitting to all nodes below. Figure 8 shows a broadcast tree for an eight-node configuration.

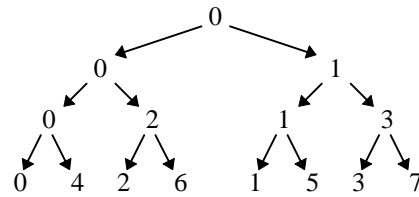


Figure 8. Broadcast Tree Configuration

One of the most important features of HCS_LIB is the signaling support that allows a programmer to have interactive communication and semaphores between master and worker nodes with latencies in the single-digit microsecond range. These functions use small shared-memory maps to transfer integer signals from worker to master while allowing customization of signals by the *set_signal_handler()* command. Via special signals, the *barrier_sync()* function allows synchronization of all nodes in a particular parallel program such that a pseudo-synchronous execution may follow. This construct is particularly important in heterogeneous SCI clusters where correct timing is essential. *P* and *V* semaphore constructs

are implemented via the *send_signal_and_wait()* function (used by the worker) and the *service_signals()* function (used by the master). This allows the master to arbitrate such high-level concepts as mutual exclusion.

Finally, the initialization and termination commands in HCS_LIB allow for the construction of a common environment for SCI-based parallel programs. The *cluster_init()* function creates all of the shared-memory maps for signaling while also fully connecting the network via message-passing ports. Once this is done, a barrier synchronization is performed to insure all initialization has completed before control is returned to the user's program. The *safe_exit()* function simply catches all signals from child processes for debugging purposes while also safely terminating all shared-memory areas, message-passing connections and ports, and freeing all structure memory used for internal SCI accounting.

A second version of HCS_LIB, currently in development, bypasses Dolphin's API but is still restricted to the kernel interface to the adapter. Collaboration with Dolphin will allow a direct, user-level interface to the adapter thereby further reducing the latency. The next version will also integrate other network interfaces into a single software package. HCS_LIB has been adapted as an interface for SMP communication. Future extensions are planned for FORE ATM and ANCOR Fibre Channel interfaces. These enhancements will offer a simple, lightweight and common user-level interface for writing parallel applications over a variety of interconnects and environments.

HCS_AM

To build upon the successes of other universities and in the spirit of simple, portable and low-latency communications, the HCS Lab is actively pursuing an Active Message (AM) implementation over SCI called HCS_AM. This ongoing work will be compatible with Berkeley's *Generic Active Message* [MAIN95] specification to enable SCI clusters access to the newest applications designed for Active Messages. AM offers a low-latency communication method which allows remote procedure execution and bulk data transfers in a simple, unified package. The current implementation of HCS_AM is built using HCS_LIB v.2 and implements the full active message specification. Interaction with developers at Berkeley will help to validate this implementation.

HCS_Linda

Linda, developed at Yale, is a simplistic yet powerful parallel coordination language that offers a content-addressable virtual shared memory across a network of workstations [SCA95] [CARR92]. Researchers at the HCS Lab have begun the design and implementation of a multithreaded, lightweight version of Linda. This is implemented using active messages due to the similarities of

remote procedure call execution and synchronization that are inherent within Linda. In contrast to existing Linda implementations, this version offers a completely dynamic network size and offered load. Development is also taking place to expand upon the HCS_Linda paradigm to Piranha or Paradise network models, which extend the basic Linda memory model to either adaptive parallelism or to fault-tolerant shared-memory servers, respectively [JUST96].

The Message Passing Interface (MPI)

The MPI specification was created by the Message Passing Interface Forum [MPIF93] with the goal of providing a portable parallel API that allows for efficient communication, heterogeneous implementations, convenient C and FORTRAN-77 language bindings, and consistency with current message-passing paradigm practices (such as PVM, NX, p4, etc.). Portability is achieved with the creation of machine-specific implementations of MPI which consist of "black box" libraries of MPI functions that adhere to the syntax and semantics of the MPI specification. The manner in which these functions perform the necessary operations are not specified by MPI and are transparent to the application programmer. The general categories of functions are: point-to-point communication (blocking and non-blocking), collective communication, group management, process topology management, environmental management, and profiling interfacing. MPICH is a sockets-based TCP/IP implementation of the MPI specification which can be ported to any number of platforms available from Argonne National Laboratory and Mississippi State University. MPICH (used over ATM, Fibre Channel, and Ethernet) and a version of MPI developed by Dolphin Interconnect Solutions that operates over SCI (MPISCI), were used in all parallel processing experiments as a comparison to HCS_LIB.

4. Throughput and Latency Tests

In order to gauge the potential performance benefits of the SCALE environment, a number of basic benchmarking programs have been developed and their results measured and collected. In [GEOR95] these measurements were presented for a ring of SS5/85 workstations running SunOS 4.1.3. In [GEOR96] the same measurements were shown using a heterogeneous cluster of SS20/85, SS20/50, and SS5/85 workstations. In this section we present the latest performance measurements for an SCI ring of eight dual-CPU SS20/85 workstations running Solaris 2.5 and similar results using ATM, Ethernet and Fibre Channel as the interconnect. The ATM portion of the cluster was constructed using FORE SBA200E/UTP5 Sbus card adapters with a FORE ASX200BX ATM switch producing an eight node 155-Mbps ATM network. Ethernet results were obtained us-

ing a standard 10-Mbps 10BASE-2 bus segment that is logically isolated using a Cisco Catalyst 3000 multi-port switch. Finally, the 1-Gbps Fibre Channel testbed consisted of two ANCOR FCS 1062 Sbus adapters. The results of these basic experiments are provided in the form of both latency and effective throughput measurements. All TCP/IP-based throughput and latency statistics were obtained using a public domain benchmarking utility by Hewlett-Packard called *netperf* [HPC96]. Throughput metrics were obtained using the TCP_STREAM test and latency metrics were obtained using the TCP_RR test.

The shared-memory benchmark using SCI examines the latency between two machines for an n -byte payload transfer. Initialization of this test is done by creating a shared-memory area on each machine with a size of n bytes. These areas are then mapped by both the remote and local nodes to create a fully-accessible, distributed shared-memory system consisting of two nodes. The actual latency is found by measuring the time it takes for node A to write a value n -bytes long to node B's buffer, which in turn reads the value and writes it back to node A's buffer. On TCP/IP tests, the latency is found by timing the round-trip of an n -byte request-response transaction through sockets. This two-way latency time t is averaged among m iterations and the one-way latency is found by

$$\text{latency} = \frac{t}{2m}$$

The message-passing benchmark examines the maximum sustained throughput between two nodes. On SCI, this test is started by creating and connecting a transmit port on one node and a receive port on the second node. The blocking I/O functions are enabled using the appro-

prate *ioctl()* signal and the data is aligned to the nearest 64-byte boundary for easy transfer to the Sbus subsystem. Using *read()* and *write()* functions, data is pushed through this "pipe" as fast as possible in n -byte payloads. The total number of bytes transferred (i.e. n bytes * m iterations) divided by the total transfer time is the sustained effective throughput. Similarly, for TCP/IP tests a simple TCP socket is constructed for streaming traffic from source to destination. All benchmark tests for SCI were compiled using the GNU C 2.7.2 compiler with level *-O4* optimization on the Solaris 2.5 operating system.

Figure 9 shows the one-way latency of HCS_LIB over SCI connected in a 2-node ring, HCS_LIB over SCI connected in an 8-node ring, ATM, Fibre Channel, and Ethernet, the latter three using TCP/IP which is typically the case for these networks. As shown, SCI has a distinct advantage showing a lowest latency of approximately 4 microseconds for smaller message sizes. Because ATM, Fibre Channel, and Ethernet all follow approximately the same track, it may be assumed that the bulk of this latency is due to TCP/IP overhead although each network adapter has its own latency characteristics.

Figure 10 illustrates the effective throughput for the four interconnects tested. Here, SCI, ATM, and Fibre Channel all increase to similar values. One explanation for this behavior is the current limitations of the Sbus in transferring data from memory to the interface card. ATM throughput tends to peak before any of the others simply because the maximum transmission unit for FORE ATM adapters is 9188 bytes. This means that for sizes larger than this, message segmentation must be performed in software.

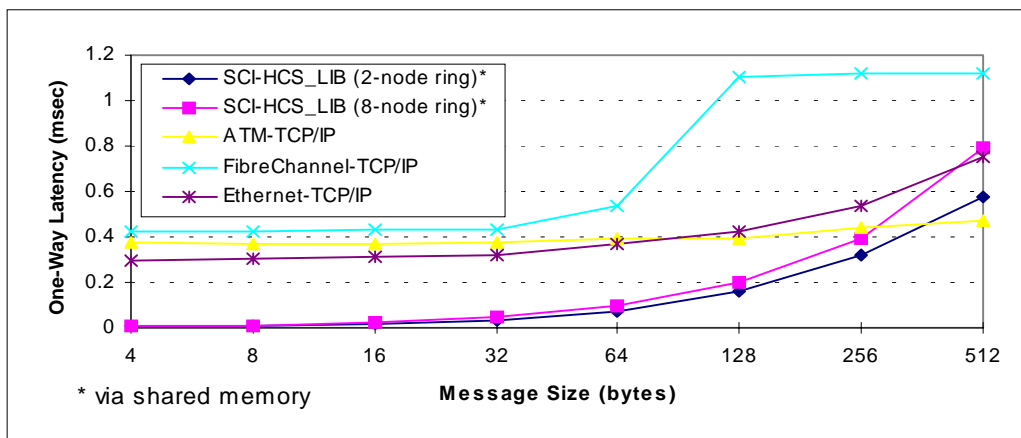


Figure 9. One-way Latency Measurements

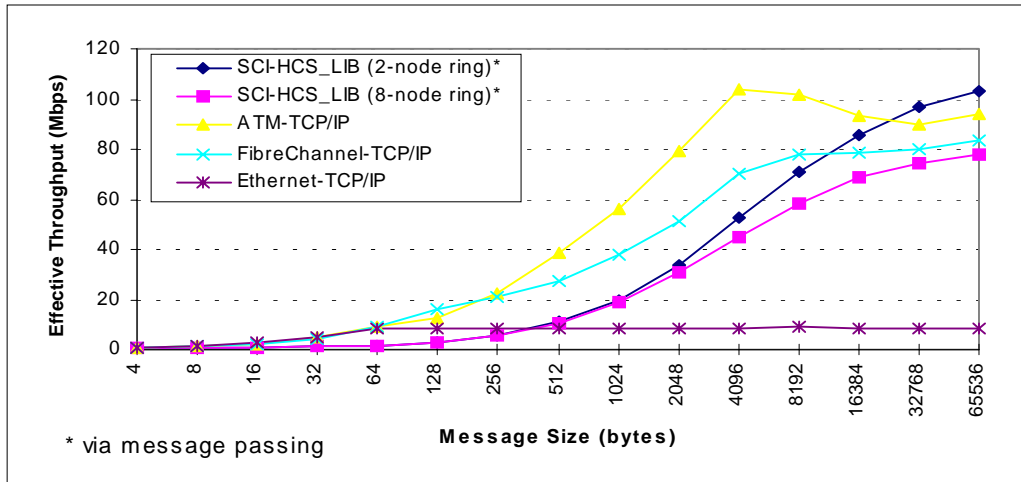


Figure 10. Effective Throughput Measurements

5. Parallel Processing Experiments

The parallel programs developed for this study were designed and implemented in C using HCS_LIB or MPI function calls over SCI. In order to compare the results with conventional workstation clusters, a comparable set of programs was also developed using MPICH. In [GEOR96], the matrix multiply and parallel sort applications were run on a heterogeneous cluster of workstations. The applications described in this section, including the matrix multiply, parallel sort, and a parallel Fast Fourier Transform, take advantage of a homogeneous cluster of

workstations shown in Figure 11 with each different interconnect. This allows the behavior of applications to be more readily interpreted without inferences as to the relative computing power of each machine on a heterogeneous network. All results are given as both execution times for individual tests and parallel efficiency. Parallel efficiency is computed by the following equation:

$$Efficiency = \frac{\frac{\text{sequential time}}{\text{parallel time}}}{\text{number of nodes}} = \frac{\text{speedup}}{\text{number of nodes}}$$

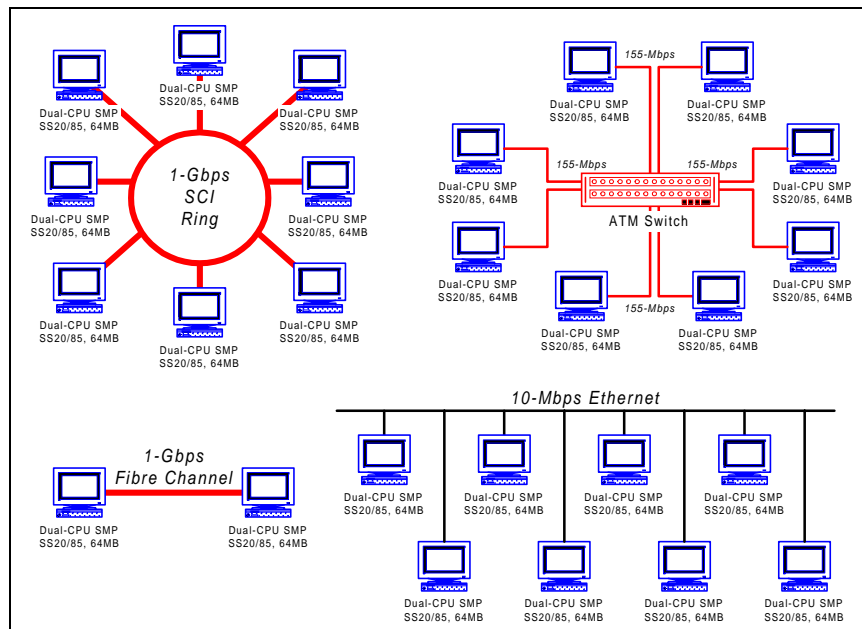


Figure 11. SCI, ATM, Fibre Channel, and Ethernet Cluster Topologies for Parallel Processing Experiments

Matrix Multiply

The first parallel application program developed for these experiments is a medium-grain implementation of an $N \times N$ matrix multiply. This algorithm is designed in a master-slave fashion in which the master passes out rows of the first matrix for the slaves to multiply. The slaves, when passed an input row, perform N dot products to produce the corresponding output row. This output row is passed back to the master and the slave is ready to request another row. This behavior typifies the concept of opportunistic load balancing (OLB).

There are three types of communication in this algorithm: broadcast of the second input matrix, scattering of the first input matrix, and collection of the output matrix. Granularity can also be varied in the matrix multiply by passing more than one row at a time in response to a request from a slave. When varying grain size from one to eight rows it was found that no significant, sustained performance increase or decrease took place. The overall best granularity only performed better than its competitors by nominal margins in a majority of cases. The test cases for this experiment were a 256x256 matrix, a 512x512 matrix, and a 1024x1024 matrix. Major and minor page faults were monitored and cache misses were minimized by assuming that the broadcast matrix had already been transposed before the calculation takes place, thus avoiding vertical indexing and dramatically improving calculation times.

Parallel Sort

The second parallel application developed performs a sort on an array of N double-precision, floating-point numbers. The algorithm used is based on a hybrid combination of the popular *Quicksort* and *Mergesort* algorithms. While *Quicksort* is among the fastest sorting algorithms on uniprocessor machines, it contains elements which are difficult to parallelize. To circumvent a considerable amount of overhead inherent in this "divide-and-conquer" scheme, an algorithm was devised such that the sort vector was divided into P equal segments, where P is the number of processors or workstations in the system. The segments are distributed, one per processor, and each processor (including the master) performs a *Quicksort* using the native C *qsort()* function. Each individually sorted segment is then passed back to the master which performs a *Mergesort* to produce the list *final_vector* containing the sorted list of N elements. Figure 12 illustrates this sequence of events.

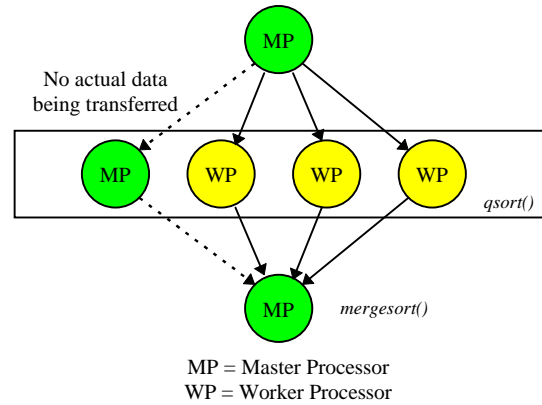


Figure 12. Parallel Sorting Algorithm

Fast Fourier Transform

The final parallel application developed is a decimation in time Fast Fourier Transform (FFT). The approach this program uses is the radix-2 FFT that was developed by J. Cooley and J.W. Tukey [STRM89]. A divide-and-conquer algorithm is used which places the restriction that the number of data points N is an integer power of two.

Figure 13 represents this algorithm for a 32-point vector. The parallel FFT is an example of agenda parallelism where a master divides the vector into blocks that are sent to the workers. All workers participate in the first and second stage before dropping out in powers of two. After each stage, the results are sent to the master who then distributes them to the remaining active workers. This process continues until the last stage in which only the master performs calculations which produce the FFT results [SARW96].

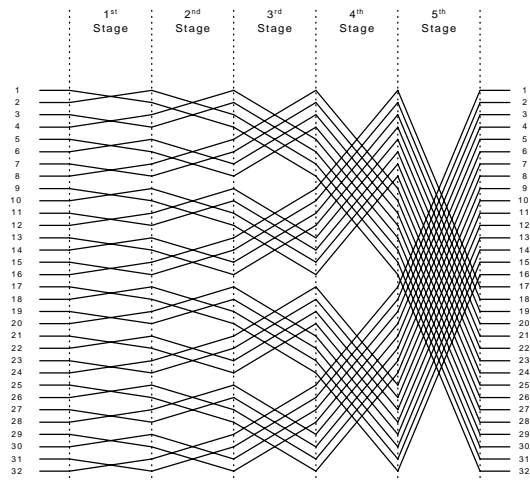


Figure 13. Diagram of a 32-point FFT

All of the parallel programs written in C function calls were compiled using the GNU version 2.7.2 compilers at level $-O4$ optimization.

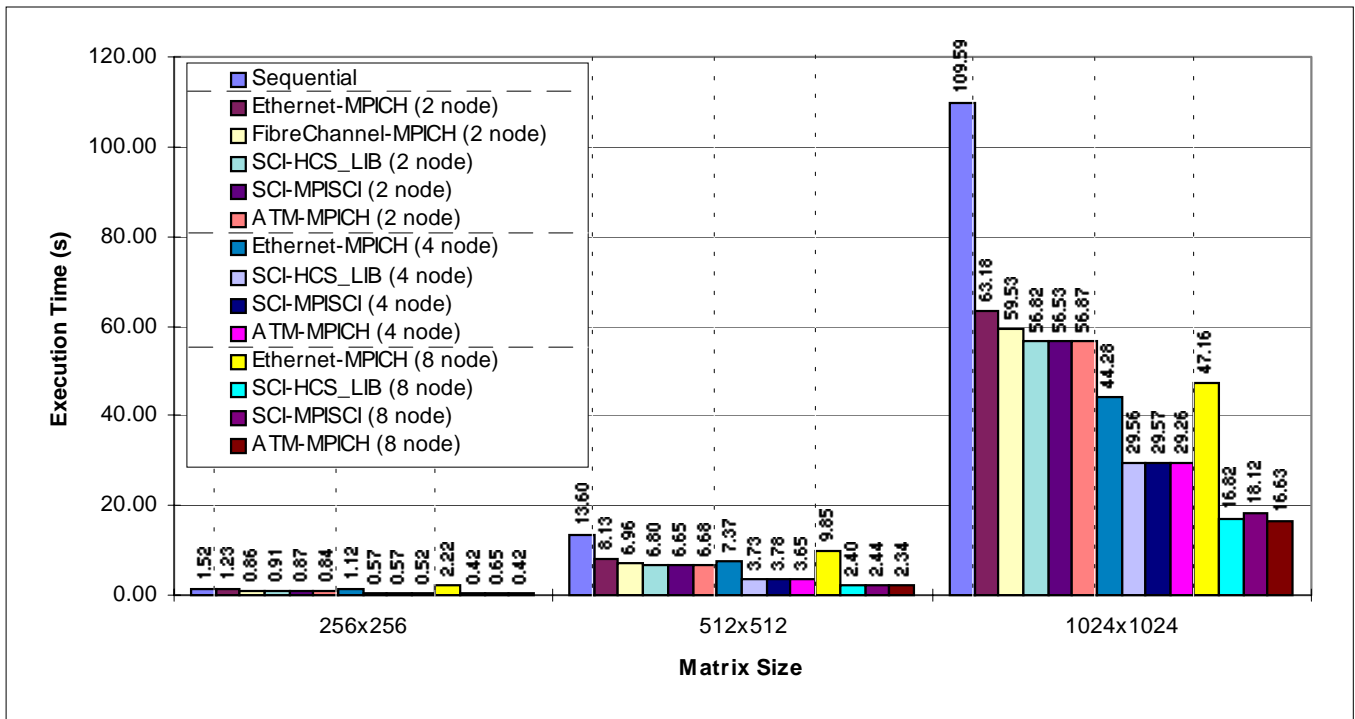


Figure 14. Parallel Matrix Multiply

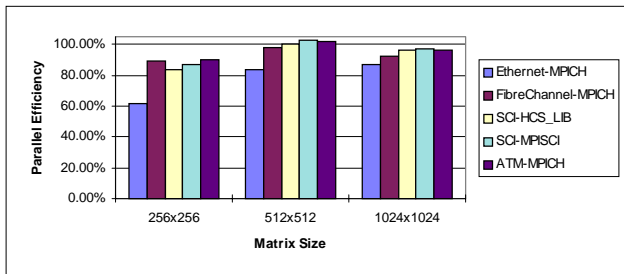


Figure 15. Parallel Matrix Multiply Efficiency (2-nodes only)

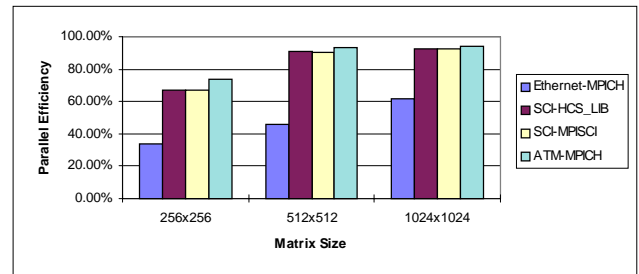


Figure 16. Parallel Matrix Multiply Efficiency (4-nodes only)

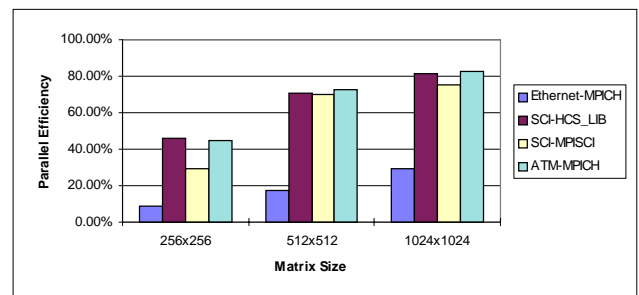


Figure 17. Parallel Matrix Multiply Efficiency (8-nodes only)

Figure 14 shows the results of the parallel matrix multiplication for several different interconnects and processor configurations. Figures 15, 16, and 17 break the test down by the number of nodes in the test. Here, the matrix multiply algorithm, although relatively fine grain, is not latency variant enough to show the benefits of using SCI as opposed to an interconnect such as ATM. The throughputs of ATM and Fibre Channel, even with the TCP/IP overhead, allow them to transport at a rate equal to or higher than SCI and thus give equal or higher performance in this parallel application. The best speedup achieved on the largest matrix size tested was approximately 6.5 for eight nodes using HCS_LIB over SCI and MPICH over ATM.

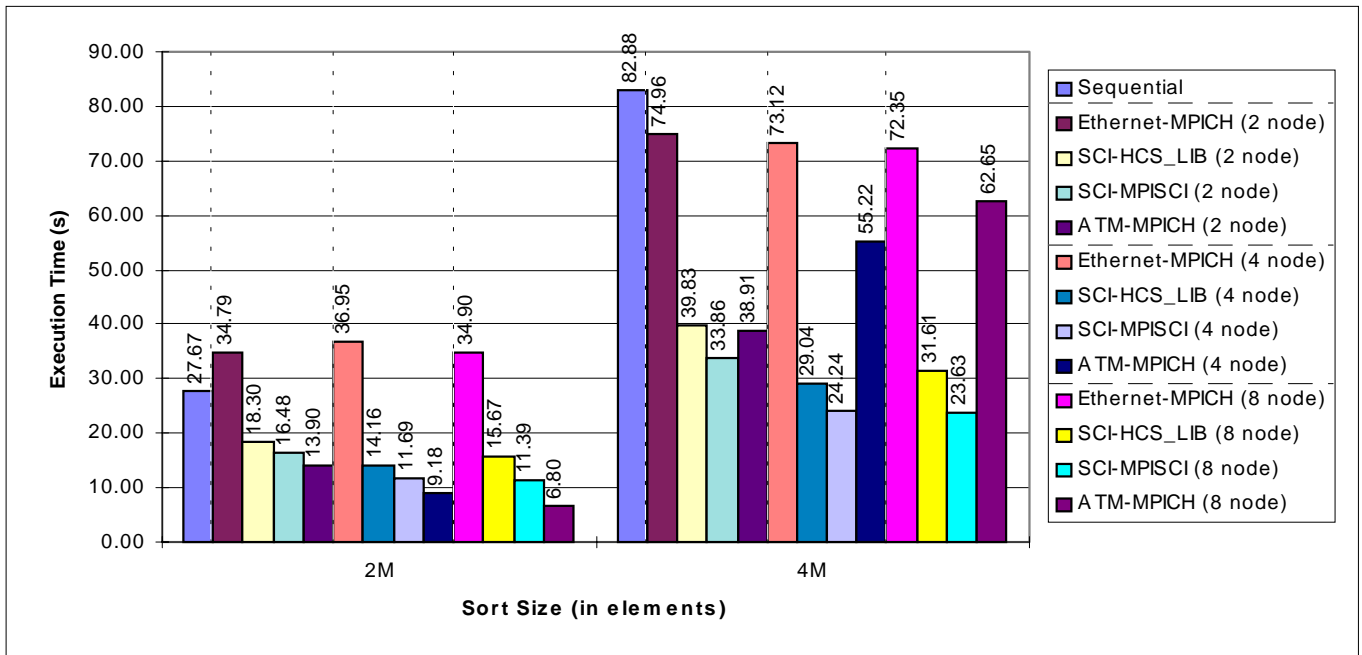


Figure 18. Parallel Sort

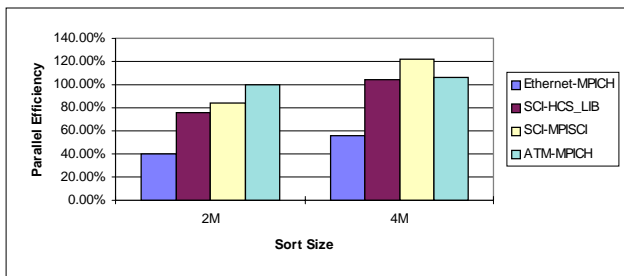


Figure 19. Parallel Sort Efficiency (2-nodes only)

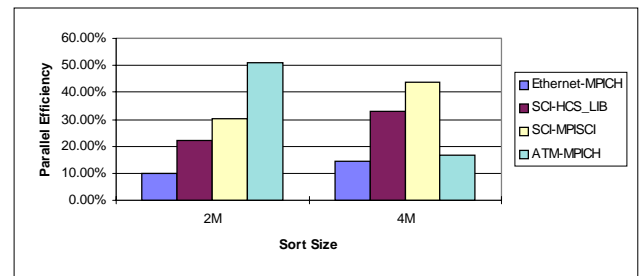


Figure 21. Parallel Sort Efficiency (8-nodes only)

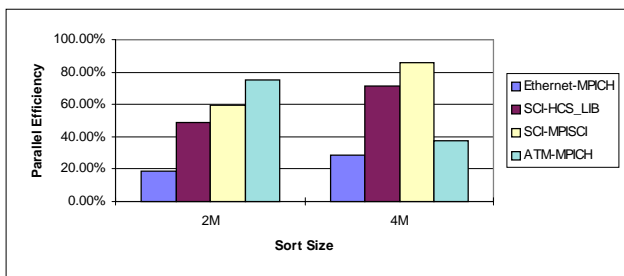


Figure 20. Parallel Sort Efficiency (4-nodes only)

Figure 18 illustrates the results of the parallel sort application and Figures 19, 20, and 21 categorize the results by the number of nodes used in the test. Superlinear speedup, as shown in Figure 19, was achieved because this is a comparison with a sequential *Quicksort* which is not able to take advantage of the smaller memory segments used in the parallel sort. Here SCI, using MPI and HCS_LIB as coordination languages, does fairly well with the former reaching a speedup of 3.5 for a 4-million element sort on eight processors. ATM does poorly on this test because of the heavy amount of message segmentation that must be done to support such large bulk transfers. The TCP/IP portion of this fragmentation is performed in software and thus CPU cycles are wasted on communication rather than on computation. Because MPICH does not use a TCP stream to transfer data, the large transfer involved in the parallel sort must be broken up into many smaller messages increasing the initial and final transfer times dramatically.

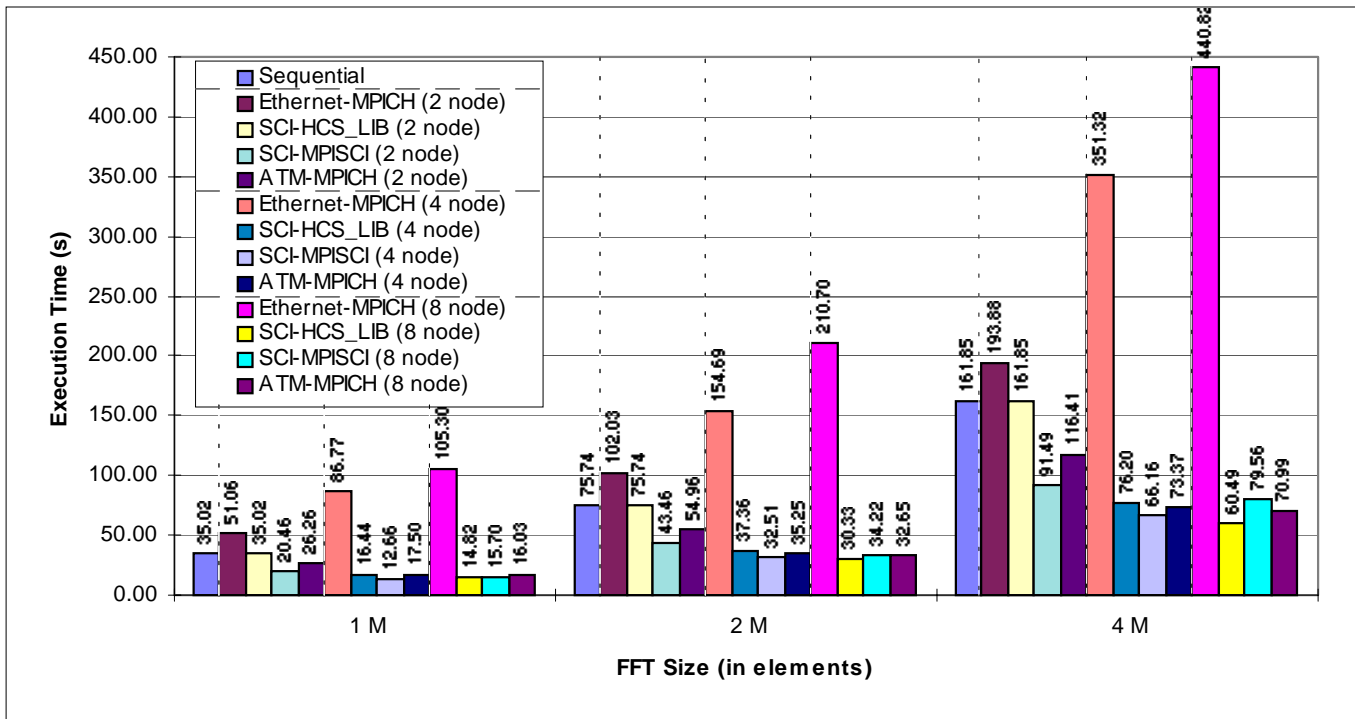


Figure 22. Parallel Fast Fourier Transform

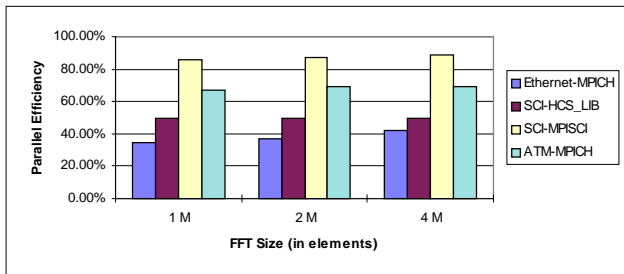


Figure 23. Parallel Fast Fourier Transform Efficiency (2-nodes only)

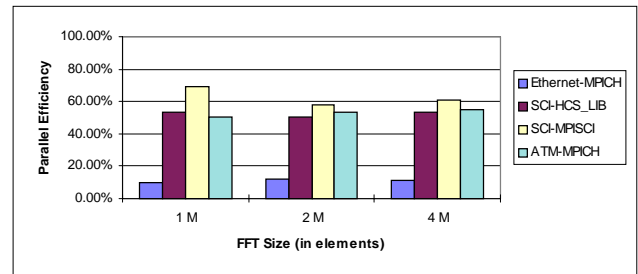


Figure 24. Parallel Fast Fourier Transform Efficiency (4-nodes only)

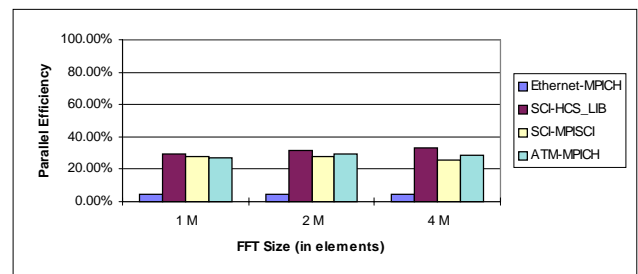


Figure 25. Parallel Fast Fourier Transform Efficiency (8-nodes only)

Figure 22 shows the parallel FFT results over MPICH, MPISCI, and HCS_LIB while Figures 23, 24, and 25 show the 2-node, 4-node, and 8-node tests respectively. Because this application transfers large amounts of data quite often, the throughput of the interconnection system has a strong impact on the speedup of the application. Here, SCI using HCS_LIB, SCI using MPISCI, and ATM all perform similarly with SCI using HCS_LIB obtaining a speedup of 2.6 for a 4-million point FFT on eight processors. This algorithm does not parallelize as well as the others simply because as it completes only the master node is performing computations while the worker nodes are sitting idle.

6. Conclusions and Future Research

This paper has presented an overview of the SCALE project in the *High-performance Computing and Simulation (HCS) Research Laboratory*. With SCALE we hope to develop methods by which workstation clusters can be extended beyond their current limitations to provide parallel and distributed processing solutions for a wide breadth of applications in a fashion that is open, portable, high-performance, distributed, fault-tolerant, and scalable. Clusters of workstations are by far the most accessible and affordable of all potential methods for high-performance computing, and a family of techniques and tools that make it possible to construct complete parallel systems around them in a modular and scalable fashion is particularly promising.

There are a number of building blocks that will play a pivotal role for these systems. High-performance workstations form the basic computational building blocks of the system. New implementations of standards-based interconnects like SCI and ATM are the backbone of the system and the method by which they are interfaced to the processors is critical although in some respects disappointing thus far in their commercial development due to the limitations of current node-chips (e.g. queue size) and I/O bus adapters (e.g. bandwidth, latency, and lack of global shared memory or cache coherency). Portable, lightweight, and low-latency communication protocols are needed to harness the potential of these high-speed interconnects and yet provide a reliable and functional access point for higher-layer software. High-level language tools for parallel programming and coordination are needed to make the distributed and parallel machines accessible, portable, effective, and efficient for real-world problems and for programmers not possessing a great deal of expertise in the world of parallel computing. Finally, libraries of key operations (e.g. matrix algebra, FFT, sorting, etc.) designed with inherent granularity knobs can help developers of grand-challenge and other applications establish a firm foothold in the world of scalable, parallel, and distributed computing in short order.

Ongoing tasks in the SCALE project team are centered around efforts in modeling and simulation research and those in experimental testbed research and are mutually supportive. Testbed results help to design and validate the models, and the models help to forecast future strengths and weaknesses in terms of both performance and dependability. In this study results to date have been presented which show some of the current strengths and weaknesses of first-generation SCALE systems constructed with current I/O bus adapters. Future interfaces will move closer to the processor thereby making the peak performance characteristics of the underlying communication fabric more attainable. In collaboration with effective and efficient communication protocols and program-

ming tools, these interfaces may lead to a scheme of system archetypes which can be fashioned in a building block manner to meet the demands of a particular problem.

Future research lies in the continuation of the modeling and testbed efforts currently underway. The development of a software hierarchy consisting of Linda-based parallel programs and programming tools operating over Active Messages has begun, and these along with HCS_LIB are being designed for portability across different UNIX workstations, interconnects, and adapters for both uniprocessor and SMP workstation configurations. We are hopeful of collaborating in the development of a high-speed, processor-bus interface of SCI that is open, accessible, and capable of becoming COTS technology, and the development of switches to support varying communication fabrics of SCI, ATM, and others. Finally, modeling and simulation efforts will continue and soon begin to step beyond basic network performance modeling to include both dependability characteristics as well as distributed and parallel performance studies.

Acknowledgments

We gratefully acknowledge our sponsors at the National Security Agency, the Office of Naval Research, and the Naval Air Warfare Center, Aircraft Division for their support.

References

- [ANDE95] Anderson, Duane, "Proposal to the P1596.6 (SCI/RT) Working Group for A Preemptive Priority Queue Protocol," White Paper for IEEE P1596.6 WG, April 5, 1995.
- [CARR92] Carriero, N. and Gelernter, D., *How to Write Parallel Programs: A First Course*. The MIT Press, Cambridge, MA. 1992.
- [DOLP95] Dolphin Inc., "1 Gbit/sec SBus-SCI Cluster Adapter Card", White Paper, Dolphin Interconnect Solutions, March 1995.
- [FREU93] Freund, R.F. and Siegel, H.J., "Heterogeneous Processing," *IEEE Computer*, June 1993, pp. 13-17.
- [GEOR95] A.D. George, R.W. Todd, and W. Rosen, "A Cluster Testbed for SCI-based Parallel Processing," *Proceedings of the 4th International Workshop on SCI-based High-Performance Low-Cost Computing*, November 1995, pp. 109-114.

- [GEOR96] A.D. George, R.W. Todd, W. Phipps, M. Miars, and W. Rosen, "Parallel Processing Experiments on an SCI-based Workstation Cluster," *Proceedings of the 5th International Workshop on SCI-based High-Performance Low-Cost Computing*, March 1996, pp. 29-39.
- [GUST95] Gustavson, D.B. and Q. Li, "Local-Area Multi-Processor: the Scalable Coherent Interface," *Proceedings of the Second International Workshop on SCI-based High-Performance Low-Cost Computing*, pp. 131-154, March, 1995.
- [GUST96] Gustavson, D.B., "RealTime Architecture for the Scalable Coherent Interface," White Paper for IEEE P1596.6 WG, March 1996.
- [HPC96] *Netperf: A Network Performance Benchmark*, Information Networks Division, Hewlett-Packard Company. Revision 2.1, February 15, 1996.
- [KLIN93] Kline, T., Rosen, W., and Harris, J., "Application of IEEE 1596-1992 Scalable Coherent Interface to Sensor/Video Interconnect Subsystems." White Paper, April 1993.
- [LACH96] Lachenmaier, R. and Stretch, T., "A Draft Proposal for a SCI/Real-Time Protocol using Directed Flow Control Symbols," White Paper for IEEE P1596.6, draft 0.21, April 12, 1996.
- [MAIN95] Mainwaring, A.M., "Active Message Applications Programming Interface and Communication Subsystem Organization," Draft Technical Report. Computer Science Division, University of California, Berkeley, 1995.
- [MPIF93] Message Passing Interface Forum, "MPI: A Message Passing Interface," *Proceedings of Supercomputing 1993*, IEEE Computer Society Press, pp. 878-883, 1993.
- [PHIP96] W. Phipps, "Switching and Parallel Processing Techniques for the Scalable Coherent Interface," B.S.E.E. Honors Thesis, Major Professor: A. George, Spring 1996.
- [JUST96] K. Justice and W. Phipps, "Design and Implementation of Linda's TupleSpace over Active Messages," HCS Research Laboratory Technical Report, August 1996.
- [SARW96] M.A. Sarwar and J.A. Herbert, "Fast Fourier Transform over Scalable Coherent Interface," HCS Research Laboratory Technical Report, April 1996.
- [SCA95] *Linda User's Guide and Reference Manual*, Scientific Computing Associates. Manual version 3.0, January 1995.
- [SCI93] *Scalable Coherent Interface*, ANSI/IEEE Standard 1596-1992, IEEE Service Center, Piscataway, New Jersey, 1993.
- [SCOT95] Scott, Tim, "Full Performance TRAIN Protocol for SCI/RT," White Paper for IEEE P1596.6 WG, draft 0.11, September 30, 1995.
- [STRM89] Strum, R.D. and Kirk, D.E., *Discrete Systems and Digital Signal Processing*, Addison-Wesley Publications, New Your, pp. 493-512, 1989.
- [TODD96] R.W. Todd, "A Simulation Case-Study of Proposed Real-Time Protocols Based on the Scalable Coherent Interface," M.S.E.E. Thesis, Major Professor: A. George, Summer 1996.
- [ZIRP96] D. Zirpoli, "Parallel Computing on Workstation Clusters via the Scalable Coherent Interface," M.S.E.E. Thesis, Major Professor: A. George, Fall 1996.