

# Modeling and Simulative Performance Analysis of SMP and Clustered Computer Architectures

**Mark W. Burns, Alan D. George, and Brad A. Wallace**

*High-performance Computing and Simulation (HCS) Research Laboratory*  
ECE Department, University of Florida  
Gainesville, Florida 32611-6200

## **Abstract**

*The performance characteristics of several classes of parallel computing systems are analyzed and compared using high-fidelity modeling and execution-driven simulation. Processor, bus, and network models are used to construct and simulate the architectures of symmetric multiprocessors (SMPs), clusters of uniprocessors, and clusters of SMPs. To demonstrate a typical use of the models, the performance of ten systems with one to eight processors and the Scalable Coherent Interface interconnection network is evaluated using a parallel matrix-multiplication algorithm. Because the performance of a parallel algorithm on a specific architecture is dependent upon its communication-to-computation ratio, an analysis of communication latencies for bus transactions, cache coherence, and network transactions is used to quantify the communication overhead of each system. While low-level performance attributes are difficult to measure on experimental testbed systems, and are far less accurate from purely analytical models, with high-fidelity simulative models they can be readily and accurately obtained. This level of detail gives the designer the ability to rapidly prototype and evaluate the performance of parallel and distributed computing systems.*

**Keywords:** Performance analysis, architecture simulation, parallel computers, clusters, computer modeling

## **1. Introduction**

Advances in microprocessors and networking and the growing demand for high-performance computer systems have led to an increased interest in parallel and distributed computer architectures. These architectures range from symmetric multiprocessors (SMPs) to network-based, massively parallel computing systems. SMPs are constructed

using two or more processors connected via a shared bus, while distributed, network-based architectures leverage high-performance interconnection networks for increased scalability. Using modeling and simulation, the performance of a wide-variety of parallel and distributed computing system architectures can be analyzed without expensive prototypes or less flexible, purely analytical models.

Modeling and simulation allows for the design, debugging, and analysis of high-performance architectures and algorithms early in the design process. Zagar and Basch [1] identify that simulation of the conceptual solution, where no technological, layout, or implementation details are present, is the foundation for all other design phases. In the concept phase, the basic architectural characteristics of a system — number of processors, interconnection method, etc. — are described and analyzed. Correctness of the conceptual solution is especially important for the design of novel parallel computing systems where no hardware prototype or similar system is likely to exist.

In the design phase, a designer must be able to confidently verify the design of a system. In this work, a combination of high-fidelity processor, bus, and interconnection network models are leveraged to evaluate parallel and distributed computing systems. By using a modular approach, designers can easily combine any number of processor models connected by any number or type of bus and interconnection network models to rapidly prototype and analyze parallel architectures.

To simulate architectures with multiple processing nodes, a tradeoff must be made between model accuracy and the practicality of both simulation time and model complexity. The performance of a parallel algorithm is limited by its communication-to-computation ratio, which is dependent upon both the algorithm and the parallel architecture to which the algorithm is mapped. To simulate the components that have the greatest effect on communication and computation time, high-fidelity processor, bus, and network models are used. However, lower fidelity is used on components that have a less significant influence on execution time in order to obtain reasonable simulation times and allow for models of practical complexity.

High-fidelity modeling of the computer system components and simulation of the architectures are performed using the Block-Oriented Network Simulator (BONeS) [2] from Cadence Design Systems. BONeS provides a graphical interface for high-fidelity system modeling, an interactive simulator for debugging, a distributed simulation engine, and post-processing data tools for evaluation of results. The combination of detailed processor, bus, and/or network models and a graphical programming environment allows the designer to evaluate the performance of parallel and distributed computing systems with confidence and ease.

To demonstrate a typical simulation with the high-fidelity models, a performance evaluation of parallel architectures using symmetric multiprocessors and clusters with one to eight processors is discussed. Also, several hybrid configurations with clusters of SMPs are simulated and compared against the other architectures. A parallel matrix-multiplication algorithm operating on several matrix sizes is executed on each of the system architectures to provide execution times as well as statistics that are not easily and accurately measured with a prototype or analytical model. High-fidelity modeling and simulation allow statistics such as latencies for bus transactions, cache coherence, network transactions, and memory operations to be gathered easily. These results can be used to help determine which architecture performs best for a particular application. Simulation of parallel and distributed architectures in this manner is also useful for conducting investigations on the effect of processor interconnection strategies on the performance of a parallel algorithm, the impact of clustered processor system configurations on scalability, or the effect of local and global memories on parallel algorithm performance.

The remainder of this paper is organized as follows. In Section 2, related research in this area is discussed. The modeling approach and the simulation environment are explained in Section 3. In Section 4, the system modeling techniques for a variety of SMP, cluster, and hybrid architectures are detailed, while the performance results are presented and analyzed in Section 5. Finally, conclusions from this research are presented in Section 6.

## **2. Related Work**

Generally, simulators are created to maximize one of two properties — simulation speed or accuracy. Maximizing simulation speed typically involves custom solutions based on trace-driven and/or parallel simulation. Trace-driven simulation is based on the memory references exhibited by an application. Simulators that maximize accuracy usually employ execution-driven simulation. Execution-driven, or program-driven, simulators are based on the execution of the workload as in an actual system [3]. Simulators using either of these techniques [4-9] can provide useful information for system architecture design tradeoffs, instruction set decisions, program or operating system tuning, and compiler optimization.

Trace-driven simulation has been used to approximate the instruction flow of a processor or collection of processors [5-8]. Although trace-driven simulators are generally faster than execution-driven simulators, there are several disadvantages to the trace-driven approach. Meaningful traces may require large amounts of storage and memory. Traces must be created from an existing, similar system or a separate high-fidelity, execution-driven

simulation. The correctness of traces is also difficult to verify because no result is produced for comparison against the expected result. Also, data-dependent timing situations such as those that frequently occur in parallel computing systems cannot be modeled [7].

Trace sampling has been used to decrease the large amounts of storage and memory necessary for trace-driven simulation. The major tradeoff for using trace-sampling methods is that accuracy may be sacrificed because the sampling can make the trace less representative of the original workload [10]. Many simulators adopt a hybrid approach to allow the user to balance simulation performance with accuracy by providing a modular simulator that allows components of varying fidelity to be placed in the system [9, 11]. Parallel simulation for program-driven execution can also be used to retain a high degree of accuracy while allowing for high-performance simulation [4, 12-13].

Most computer system modeling and simulation efforts have concentrated on uniprocessor or shared-memory multiprocessor environments [1, 5-7, 9, 14-16]. Levels of detail and simulation performance are the primary differences between these systems. SimOS [11, 15, 17] provides what may be the greatest detail for a uniprocessor system; at its finest level of detail, SimOS supports a fine-grained processor with pipelining, a memory hierarchy, DMA, interrupts, I/O, an operating system, and a network interface. Other implementations, such as the simulations discussed by McCarron and Tung [6], disregard operating system references to minimize overhead and maximize simulation performance. Alternatively, models like the performance evaluation discussed by Qin and Baer [8] ignore the hardware internals and concentrate solely on access rates and system timing.

The major weakness of many simulators is a lack of support for parallel systems that do not exclusively use shared memory. Several simulators have been tailored to multicomputer systems. In PROTEUS [18], multiple processors are connected via a bus, direct network, or indirect network; the execution of an application and the architecture simulation are time-multiplexed on a single host processor. Talisman [19] also simulates multiple processors by switching program execution with timing semantics embedded in the simulator. However, simulators such as PROTEUS and Talisman provide little support for high-fidelity network simulation. Given the important role played by the interconnection network for communication between processors and clusters, and its inherent complexity, an accurate network model is a necessity for the simulation of distributed systems.

Rapid virtual prototyping and profiling of parallel algorithms, architectures, and systems are possible with the Integrated Simulation Environment (ISE) [20]. ISE simulates parallel and distributed architectures using a

combination of high-fidelity models and existing hardware-in-the-loop to execute real parallel programs on virtual prototypes. However, ISE is only effective for prototyping algorithms on testbed processor architectures.

In this paper, processor subsystem, bus, and network hardware models are presented that are useful for building a wide array of system models ranging from a simple uniprocessor to a loosely coupled, distributed parallel system. Execution-driven simulation is used to provide sufficient accuracy, where the simulated processors execute the *actual* machine-language code from the application. High-fidelity modeling and simulation provide, but are not limited to, any of the statistics for performance evaluation available from existing simulators such as execution times, utilization percentages, and cache or memory access rates.

### 3. System Structure

The modeling and simulation environment in BONEs is block-oriented in nature. Components are specified graphically as entities with strongly cast inputs and outputs. Internally, components are defined as C++ code primitives or as a hierarchy of block components; all components at the lowest level of the hierarchy are eventually specified as primitives. Information passes between blocks using wired I/O connections or through global and local parameters.

The block-oriented property of the simulator allows components to be easily swapped between simulation iterations. Variable levels of detail are modeled to balance simulation performance and accuracy. By using a parameter-based switch that is activated on system initialization, one of several variations of a component is simulated in a given instance. For example, the DLX processor discussed in Section 4 contains a variable Level 1 (L1) data cache, and a user-controlled parameter determines which component is used in the system — no cache (pass-through), a simple direct-mapped cache, a cache with victim buffer, or a cache with victim buffer and write buffer. Due to the event-driven simulation engine, components not being used add no performance penalty to the simulation.

BONEs is a viable platform for high-fidelity simulation because of its block-oriented style and debugging tools. Both a background simulator and an interactive debugger are provided. The interactive debugger is invaluable for determining system hot spots and evaluating the effect of subtle model changes. Fast-forwarding mechanisms are also available within the debugger to allow a designer to reach the steady state of a system before beginning an analysis.

In this study, three major components are used in modeling a shared-memory multiprocessor or distributed-memory multicomputer system:

- A processor subsystem, including a CPU or multiple CPUs in a symmetric multiprocessor configuration, a cache hierarchy, and a memory bus interface;
- A bus hierarchy, which includes both memory and I/O buses;
- An interconnection network, including links, switches, buffers, and an I/O bus interface.

Structuring the system in a modular manner allows the designer to create any system architecture easily with little debugging. The system models described in this paper include a DLX [21-22] processor subsystem, system buses, and a Scalable Coherent Interface (SCI) [23-24] interconnection network. A common interface among these components allows interchangeability that provides the designer with a powerful set of models for simulation.

#### **4. Processor and System Modeling**

An advanced version of a DLX uniprocessor or multiprocessor is used as the processor subsystem. DLX is an academic design that embodies many of the architectural features of existing RISC processors [21]. Each uniprocessor contains a CPU and two levels of cache hierarchy. The DLX CPU employs a load/store architecture with a five-stage pipeline; the DLX pipeline stages are instruction fetch, instruction decode, execution, memory access, and write back. All integer components in the CPU are 32-bits wide, and floating-point instructions for 32- and 64-bit words are supported.

Separate L1, or on-chip, instruction and data caches are used. The user may define the L1 instruction cache size and the L1 data cache size. Both of the L1 caches are direct-mapped, and the data cache implements a write-through with no write-allocate policy. The user may also select one of several variations for each L1 cache including no cache, standard, with a victim buffer, non-blocking, and, for the data cache, with a write buffer. The unified Level 2 (L2), or off-chip, cache is fully associative and implements a write-back with write-allocate policy. Like the L1 caches, the L2 cache size is variable. At simulation time, the user may choose between no L2 cache, a standard cache, or a cache with a variable-size victim buffer.

The processor subsystem uses parameters to specify timing operations. The CPU clock frequency and L2 cache access time are variable; the L1 caches have a fixed access time of one processor cycle. Within each processor, the instruction latencies for multiplication and division operations are also adjustable.

Each processor executes real code. The DLX instruction set architecture, as well as several special instructions including non-coherent reads and writes and atomic exchanges, is supported. The processors operate on formatted S-records for DLX binary code generated by compilation and assembly. A public-domain DLX compiler, *dlxcc*, and assembler, *dlxasm*, allow the designer to create the S-records from C or assembly-language code, respectively.

Operating system calls are not supported by the DLX processor subsystem. In testbed analyses, it is common practice to begin timing the execution of a program after initialization, therefore operating system overhead, such as I/O mapping, etc., does not play a significant role. In order to obtain consistent performance results, the parallel algorithm is assumed to be the only process running on each processor. This assumption further removes the need to support operating system calls.

Symmetric multiprocessors are configured as two or more processors sharing a single memory bus, I/O bus, and local memory. Each processor in the SMP contains its own L1 and L2 caches. A snoopy cache coherence protocol is used to keep data consistent between caches and main memory. The coherence scheme enforces an invalidate protocol, and each processor keeps a three-stage ownership field for cache blocks — *shared* (read-only), *exclusive* (read-write), and *invalid*. Cache coherence causes a minimum of one additional cycle of overhead for L2 accesses in a multiprocessor. Although the snoopy protocol monitors all transactions on the memory bus, a separate control bus is used to manage the invalidate protocol for the L2 caches in the SMP. Using a separate control bus reduces the latency for cache coherency and reduces contention for the memory bus. Figure 1 displays the DLX processor subsystem and bus hierarchy, where  $N$  is a user-specified value for the number of processors in a node.

The DLX bus hierarchy contains a split-transaction memory bus as well as a connected-transaction I/O bus. A generalized bus model with separate address, data, and control lines is used for both the memory and I/O buses. Centralized bus controllers manage access to each bus, and bus requests are serviced in FIFO order. Operations on the I/O bus are blocking; a transaction does not relinquish access to the I/O bus until both a request and response are completed. The clock frequency and bus width of the memory and I/O buses are parameterized. For improved performance, the bridge between the memory and I/O buses combines transactions when consecutive read or write operations share the same memory address.

Memory modules in each processing node are interfaced to the memory bus. The access time for the memory modules is user-specified at simulation time. Similar to the memory-I/O bus bridge, multiple consecutive read

requests to the same memory location are combined by the memory controller and serviced with only one memory access for improved performance.

The SCI interconnection network model connects nodes via their I/O bus interfaces. *IEEE Standard 1596-1992 Scalable Coherent Interface* is a high-performance, point-to-point interconnect that supports a shared-memory model scalable to systems with up to 64k nodes [23]. Simulation is done at the packet level and assumes error-free communication and control. The SCI model emulates the IEEE standard with the exception of broadcasts and distributed cache coherency, which are not necessary for the simulations performed in this paper. Parameters are used to specify the following network attributes: SCI data rate, input queue size, bypass queue size, and output queue size. Figure 2 demonstrates a typical interconnection of several processing nodes via SCI using these models.

The SCI node interface connects directly to the I/O bus of each processor subsystem. All transactions through the SCI node interface are delayed to account for hardware and software overheads in the network adapter. Remote memory request and response transactions to the node interface incur a one-way delay of 2 $\mu$ s. With the interface delay included, the average round-trip remote transaction latency for a 64-byte read transaction, which includes the time required to traverse the SCI links, intermediate SCI node interfaces, and the memory and I/O buses of the source and destination nodes, on an unloaded, 2-node ring is approximately 6 $\mu$ s. These results are comparable to those measured on an SCI testbed in the laboratory.

To study and compare the performance of the various systems through execution-driven simulation, one or more benchmarking applications are required. One common application for performance benchmarking of parallel architectures is a matrix-multiplication algorithm where the overall execution time is measured. Simulations with the processor, bus, and network models use the matrix-multiplication algorithm shown in Figure 3. In the figure,  $N$  is the total number of processors,  $p$  represents the processor number (from 0 to  $N - 1$ ), and the size of the matrices is  $n \times n$ . The algorithm performs the integer matrix operation  $C = A \times B$ , where each matrix is composed of 4-byte elements.

The use of the memory modules by the algorithm differs for SMP and cluster simulations. SMP architectures contain only one memory module, therefore the  $A$ ,  $B$ , and  $C$  matrices are accessible as local memory addresses by all processors. For clustered systems, the entire  $B$  matrix is distributed to the memory module in each node before simulation timing begins. The  $A$  matrix in clustered systems is initially stored in one node's memory module, and each processor fetches portions of the  $A$  matrix using remote transactions as needed by the algorithm during

execution. For cluster simulations, the resultant  $C$  matrix is written to the node that also stores the  $A$  matrix, and the simulations are considered finished when all processors have completed the computation of their portion of the resultant matrix and written the results to memory. The matrix results are written to memory with a non-coherent instruction so that the results are not cached locally and a cache block is not fetched because of the L2 cache's write-allocate policy.

The processor subsystem, bus hierarchy, and interconnection network models are used to create three parallel architectures which are evaluated using the parallel matrix-multiplication algorithm: a symmetric multiprocessor, a cluster of uniprocessors, and a hybrid of the two approaches — a cluster of SMPs. Ten systems using one to eight processors and these architectures are used to demonstrate the effectiveness of the models. Each system is represented by an  $(M, N)$  pair, where  $M$  is the number of processing nodes in each cluster and  $N$  is the number of processors in each node. For example, the  $(4, 2)$  system represents a cluster of four dual-processor SMPs.

#### *4.1 Symmetric Multiprocessors*

Symmetric multiprocessors with one, two, four, and eight processors are simulated. The configuration for each SMP is similar to Figure 1 with  $N = 1, 2, 4,$  and  $8$ . The uniprocessor system ( $M = 1, N = 1$ ) executes a sequential matrix-multiplication algorithm. SMPs have one memory module that contains the matrix-multiplication routines for each processor as well as the segments for the  $A, B,$  and  $C$  matrices. Figure 4 displays a system-level block diagram of the BONEs four-processor SMP model.

Process initialization is accounted for in all simulations. For the uniprocessor system, simulation timing and algorithm execution begin at time zero. For SMPs, simulation timing begins at time zero when the master processor begins execution, and the overhead to spawn processes on the other processor(s) in an SMP is included. Two memory bus cycles are required to initialize a process on each processor in an SMP, therefore all processors are executing the algorithm after a delay of approximately  $2 \times (N - 1)$  memory bus cycles.

#### *4.2 Uniprocessor Clusters*

A cluster of uniprocessors is constructed as a ring of  $M$  SCI nodes with one processor at each node. The number of processors is varied as in the SMP case; because each cluster node contains only one processor,  $M$  is set to 2, 4, and 8, respectively. The single processor case is not included because the configuration would be identical to the uniprocessor discussed previously. Figure 2 shows the general configuration of the DLX cluster architecture, and a

BONeS system-level block diagram of the four-node cluster is shown in Figure 5. Each node includes a local memory module that contains the matrix-multiplication instructions and the  $B$  matrix. The memory module in node 0 also contains a global memory segment shared by all nodes. The global memory contains data for the  $A$  matrix as well as space for the resultant  $C$  matrix. Process initialization is performed in a manner similar to the SMP case. The initialization of each process in a cluster architecture requires a remote transaction over the SCI ring, and a remote one-way transaction has a latency of approximately  $3\mu\text{s}$ . Therefore, an additional delay of approximately  $(M - 1) \times 3\mu\text{s}$  is incurred before all nodes are executing the algorithm.

### 4.3 Clusters of Symmetric Multiprocessors

A natural alternative for parallel architectures is a hybrid between a symmetric multiprocessor and a clustered network of uniprocessors. A cluster of SMPs attempts to garner the benefits of locality and cost-effectiveness available in SMPs while also benefiting from the removal of the bottleneck and limits on scalability associated with a shared bus. This configuration is similar to Figure 2, however each node contains more than one processor. For the simulation experiments, three hybrid cluster architectures are simulated: a cluster of two dual-processor SMPs ( $M = 2, N = 2$ ), a cluster of four dual-processor SMPs ( $M = 4, N = 2$ ), and a cluster of two quad-processor SMPs ( $M = 2, N = 4$ ). A BONeS block diagram of the ( $M = 2, N = 2$ ) SMP cluster is shown in Figure 6. The memory configuration for clusters of SMPs is identical to that of the uniprocessor clusters. Process initialization is also executed in the same fashion as in the above cases. Remote SCI transactions are required to initialize the processes in all nodes except the master node, and all processes are executing after a delay of approximately  $(M - 1) \times N \times 3\mu\text{s}$ .

## 5. Performance Results

Simulations with the models discussed in Section 4 were verified by inspection, sanity checks, and reference to testbed results. Initially, the resultant matrix from the simulation was checked for correctness. Sanity checks were performed on the results with each architecture to ensure that the speedup of an  $N$ -processor system (with respect to the uniprocessor system) was not greater than  $N$ . The execution time for the basic uniprocessor DLX system was also verified against a comparable SPARC workstation using both the matrix-multiplication routine and several other test algorithms. Similarly, the SCI model was validated against an SCI testbed present in the laboratory.

Two system settings are discussed throughout this section — a basic and an advanced system. The two configurations provide insight into the effect of processor and bus choice on system architecture performance.

Table 1 displays the parameters for the two configurations.

The basic settings configure the uniprocessor model to be roughly equivalent to a SPARCstation-5 workstation. The memory and I/O bus parameters are configured to represent MBus and SBus, respectively. The advanced simulation settings represent a more modern workstation, roughly comparable to an Ultra-1 workstation with a UPA memory interconnect and a second-generation SBus for memory and I/O, respectively. However, it is important to note that the processor and bus models are not intended to directly emulate any commercially available workstation. The parameters are adjusted in this manner to provide a good approximation of system operation for variations in the processor subsystem and bus hierarchy.

The remainder of this section presents performance analyses of the basic and advanced systems executing the parallel matrix-multiplication algorithm with matrix sizes of  $16 \times 16$ ,  $32 \times 32$ ,  $64 \times 64$ , and  $128 \times 128$  integer elements. Speedup is referenced against the uniprocessor execution time for each matrix size. A breakdown of the major components of communication time for the  $64 \times 64$  matrix-multiplication algorithm is also shown, with the components defined as follows:

*Memory Bus Transaction* — The average time per processor for local request and response (if necessary) transactions on the split-transaction memory bus. This time includes delays caused by bus contention.

*Cache Coherence* — The average time per processor a memory bus transaction is delayed to enforce cache coherency.

*SCI Transaction* — The average time per processor for remote transactions over the SCI interconnection network. A round-trip read transaction includes the time required for the request and response to pass through the memory bus, bus bridge, I/O bus, SCI node interface, and SCI links between the local and remote nodes. Write transactions are similar to reads, however writes are unidirectional from the local to the remote node and do not block the local I/O bus because no response is required. The SCI transaction time also includes queuing delays caused by contention.

*Memory Operation* — The average time for each memory request to be serviced. This time includes the main memory access time and delays from contention to shared-memory locations.

### 5.1 Analysis of the Basic Systems

Table 2 displays the execution times for all basic systems, and Figure 7 shows the speedup of the eight-processor basic architectures for all matrix sizes.

The SMP architectures have the lowest execution times, and thus display the highest performance, of the two- and four-processor basic systems (e.g. execution times of 347.385ms and 174.290ms, respectively, for matrix sizes of  $128 \times 128$  elements). For the eight-processor basic systems, the cluster of uniprocessors exhibits the largest performance for matrix sizes larger than  $16 \times 16$  elements, followed by the cluster of dual-processor SMPs, the eight-processor SMP, and finally the cluster of quad-processor SMPs.

Figure 7 displays several key characteristics of the basic systems. The  $16 \times 16$  matrix-multiplication results do not demonstrate the same pattern as the larger matrix sizes. This characteristic is due to the overhead of process initialization and communication. Also, the difference in speedup between each architecture is reduced as the matrix size is increased from  $32 \times 32$  to  $128 \times 128$  elements. As the matrix size increases, the communication-to-computation ratio decreases since communication in a matrix-multiplication algorithm scales as  $O(n^2)$ , but computation increases as  $O(n^3)$  with problem size. This property causes the computational portion of the algorithm to dominate execution time as matrix size is increased.

Table 3 details an analysis of the key communication components for the  $64 \times 64$  matrix multiplication and reveals the causes of disparity between the performance of each architecture. Due to contention, the average memory bus transaction time increases exponentially as the number of processors per node,  $N$ , increases (e.g. 72.34ns, 83.71ns, 175.87ns, and 533.02ns for  $M = 1$  and  $N = 1, 2, 4,$  and  $8$ , respectively). Delays for cache coherence in SMPs also increase with  $N$ , however the increase is relatively insignificant due to the separate control bus used for the snoopy protocol. An examination of the average SCI transaction time for reads and writes on clustered systems reveals that the values increase primarily with  $N$  and to a lesser extent with the number of nodes,  $M$ . For example, the average SCI read transaction measurements increase from  $6.65\mu\text{s}$  to  $6.83\mu\text{s}$  as  $M$  is increased from the (4, 1) to the (8, 1) configuration, however, as  $N$  is increased from the (4, 2) to (2, 4) configuration, the average SCI read transaction measurements increase from  $7.59\mu\text{s}$  to  $11.33\mu\text{s}$ . This trend occurs because each I/O bus uses connected transactions — a remote transaction blocks the bus transactions of the other processor(s) on the same node until a response to the first request is received. The average memory operation time is similar to the memory bus transaction time and

increases with  $N$  (e.g. 60.00ns, 60.06ns, 66.61ns, and 90.30ns for  $M = 1$  and  $N = 1, 2, 4,$  and  $8,$  respectively) due to contention for the shared-memory modules.

The results detailed in Table 3 further explain the patterns exhibited in Table 2 and Figure 7. Because clusters of uniprocessors have only one processor per node, the remote requests from other processors on the same node are not delayed as in the hybrid configurations. With the basic settings, clusters of uniprocessors therefore have a higher performance than the hybrid configurations with the matrix-multiplication algorithm. Clusters of dual-processor SMPs also outperform clusters of quad-processor SMPs because more transactions are delayed with each remote request in the latter case.

## 5.2 Analysis of the Advanced Systems

The execution times for all the advanced systems are shown in Table 4, and the speedup for the eight-processor advanced systems is shown in Figure 8.

Because the width and clock frequency of the memory bus are higher in the advanced systems, memory bus contention is greatly reduced. SMP architectures benefit the most from reduced memory bus contention, and thus the two-, four-, and eight-processor SMPs achieve the lowest execution times for all three multiprocessor system sizes as a result (e.g. execution times of 190.632ms, 95.450ms, and 48.130ms, respectively, for the  $128 \times 128$  element matrices). Although the SMP configurations show higher speedups than the cluster and hybrid configurations, the speedup of the networked architectures becomes closer to the speedup of the SMPs as the matrix size is increased from  $32 \times 32$  to  $128 \times 128$  elements because of the decrease in the communication-to-computation ratio of the matrix-multiplication algorithm.

Table 5 displays the communication composition for the advanced systems with the  $64 \times 64$  element matrix multiplication. The trends displayed in the advanced systems are similar to those of the basic systems, however one significant difference is noted. Because memory bus contention is reduced, more demand is placed on the memory modules in the advanced systems. The average memory operation measurements rise more significantly as  $N$  is increased in the larger advanced systems due to increased contention for the memory module (e.g. the average memory operation delay almost doubles from 60.15ns to 116.56ns between the two- and eight-processor SMPs).

As with the basic systems, the (2, 4) configuration displayed the lowest speedup of the advanced eight-processor systems. As reflected in Table 5, remote transactions in the advanced systems suffer from the same contention bottlenecks as the basic systems.

### 5.3 Discussion

Using simulative analyses, the impact of the architecture, processor subsystem, and bus hierarchy on bottlenecks and performance can be evaluated for various systems and algorithms. Although the basic and advanced system settings yield different results, the patterns exhibited can be used to infer the performance of the architectures for larger matrix sizes. As the matrix size is further increased, a matrix multiplication performed in this manner becomes more coarse in granularity and approaches an embarrassingly parallel problem where all computation consists of a number of tasks that execute more or less independently without extensive communication requirements. Because each processor fetches instructions, a portion of the  $A$  matrix, and the entire  $B$  matrix, the L2 caches will experience capacity misses for matrix sizes larger than  $128 \times 128$  elements. In this scenario, the dominating factor of communication will be memory bus transactions caused by capacity misses in the L2 caches. Also, the average memory bus transaction times increase with the number of processors per node. Therefore, as problem size increases, the speedup of the purely clustered architectures can be expected to dominate for the basic and advanced systems, followed by the hybrid and SMP architectures, and the performance gap between architectures will expand as problem size grows.

## 6. Conclusions

In this paper a collection of high-fidelity models for simulation of parallel and distributed computer architectures has been presented. The models can be useful for analyzing single-processor systems or symmetric multiprocessors, however the power of the processor, bus, and network models lies in the ability to allow designers to rapidly configure and simulate a large array of system architectures at a sufficient level of detail. A designer may simply choose a processor model (or set of models for a heterogeneous environment), bus hierarchy, and one or more interconnection networks before connecting the modules and beginning a simulation.

As a case study, ten separate architectures were evaluated using a matrix-multiplication algorithm. The processor, bus, and interconnection models discussed provide accurate timing mechanisms for performance evaluations. The

results demonstrate the extent to which computer architecture performance is dependent upon factors such as design choices on processor subsystem and bus hierarchy, type and size of problem, etc. For the basic system settings, SCI-based clusters of uniprocessors exhibit the highest performance for the eight-processor systems with matrix sizes greater than  $32 \times 32$  elements. However, for the more modern system settings, eight-processor SMP architectures provide the highest performance with matrix sizes up to  $128 \times 128$  elements. By examining the key communication components of the matrix-multiplication algorithm, the “crossover point” where SCI-based clusters of uniprocessors exhibit the highest performance for the advanced systems begins with matrix sizes near  $256 \times 256$  integer elements.

This suite of high-fidelity models minimizes the need for costly hardware prototypes and is invaluable for rapid and accurate analyses on the complex issues associated with the performance of parallel and distributed systems. The system executes real programs with a high degree of fidelity at the processor, bus, and interconnection network level. The use of execution-driven models yields results less quickly than other simulators designed primarily for simulation performance. However, at some stage in the design cycle an accurate simulation model is still a necessity. The models discussed herein balance accuracy and design complexity to allow designers to make important decisions on issues such as instruction set selection, system design tradeoffs, algorithm adaptation for a specific architecture, and architecture verification and validation. Researchers and designers may readily employ these models in the BONEs simulation environment to conduct a variety of performance and fault-injection experiments with virtual prototypes and thereby study various forms of cluster and multiprocessor systems.

One limitation in the current environment is simulation time. Due to the use of execution-driven simulation with high-fidelity models, the experiments conducted in this paper require simulation times that range from several minutes to several days on a conventional workstation. One direction of future research will focus on the incorporation of techniques such as distributed simulation, parallel simulation, and hardware-in-the-loop components to keep simulation times reasonable. Another direction for future research will focus on adding new models and features to the suite, including additional processor subsystem and network models. Finally, and perhaps most importantly, extended case studies using these models and extensions are needed to gain insight into the many design choices in the system architecture and determine, for a larger class of problems, the key tradeoffs in performance, dependability, scalability, cost, etc. for emerging systems.

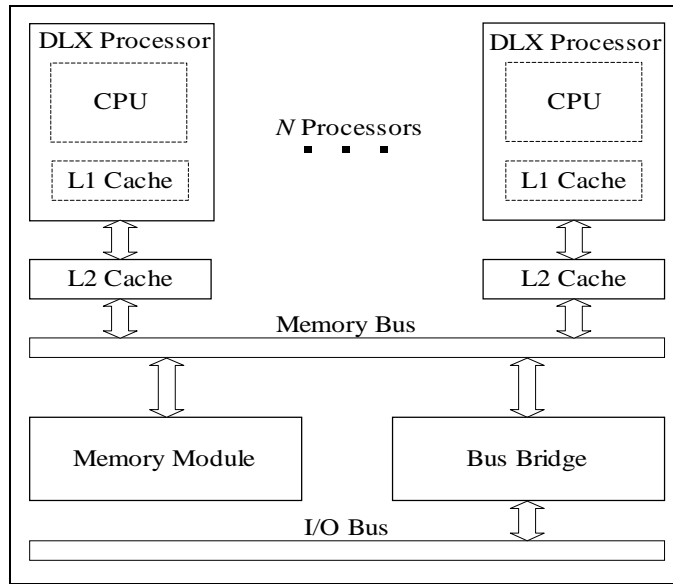
## 7. Acknowledgements

This research was supported in part by the National Security Agency. SCI equipment support for model validation was provided by Dolphin Interconnect Solutions and Scali AS. Finally, special thanks are extended to Matthew Chidester, Robert Todd and other members of the HCS Research Lab for assistance with the bus and SCI models and related topics.

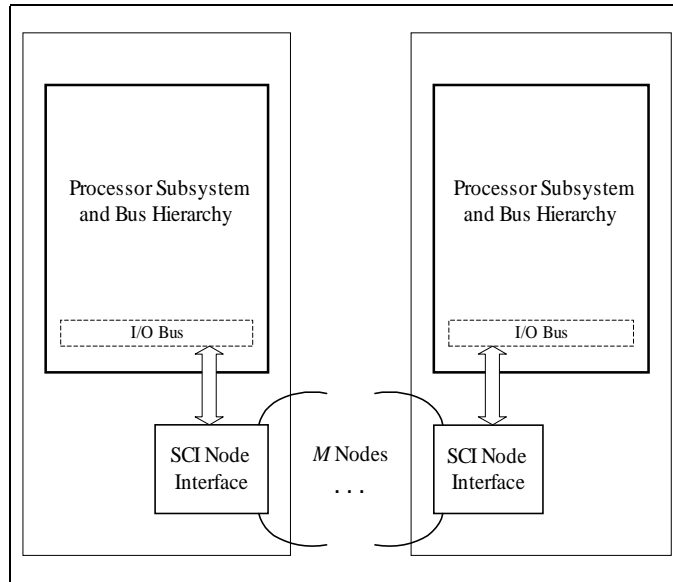
## 8. References

- [1] Zagar, Mario and Danko Basch, "Microprocessor Architecture Design with ATLAS," *IEEE Design & Test of Computers*, Vol. 13, No. 3, July-September, 1997, pp. 104-112.
- [2] Shanmugan, K. Sam, Victor S. Frost, and William LaRue, "A Block-Oriented Network Simulator (BONeS)," *Simulation*, Vol. 58, No. 2, February, 1992, pp. 83-94.
- [3] Dahlgren, Fredrik, "A Program-driven Simulation Model of an MIMD Multiprocessor," *Proceedings of the 24<sup>th</sup> Annual Simulation Symposium*, New Orleans, Louisiana, April, 1991, pp. 40-49.
- [4] Ayani, R., Y. Ismailov, M. Liljenstam, A. Popescu, H. Rajaei, and R. Rönngren, "Modeling and Simulation of a High Speed LAN," *Simulation*, Vol. 64, No. 1, January, 1995, pp. 7-13.
- [5] Edmondson, John and Matt Reilly, "Performance Simulation of an Alpha Microprocessor," *IEEE Computer*, Vol. 31, No. 5, May, 1998, pp. 50-58.
- [6] McCarron, Christopher W. and Cheng-Hsien Tung, "Simulation analysis of a Multiple Bus Shared Memory Multiprocessor," *Simulation*, Vol. 61, No. 3, September, 1993, pp. 169-175.
- [7] Poursepanj, Ali, "The PowerPC Performance Modeling Methodology," *Communications of the ACM*, Vol. 37, No. 6, June, 1994, pp. 47-55.
- [8] Qin, Xiaohan and Jean-Loup Baer, "A Performance Evaluation of Cluster Architectures," *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Seattle, Washington, June 15-18, 1997, pp. 237-247.
- [9] Rosenblum, Mendel and Emmett Witchel, "Embrea: Fast and Flexible Machine Simulation," *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Philadelphia, Pennsylvania, May 23-26, 1996, pp. 68-79.
- [10] Bose, Pradip and Thomas M. Conte, "Performance Analysis and Its Impact on Design," *IEEE Computer*, Vol. 31, No. 5, May, 1998, pp. 41-48.
- [11] Rosenblum, Mendel, Edouard Bugnion, Scott Devine, and Stephen A. Herrod, "Using the SimOS Machine Simulator to Study Complex Computer Systems," *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 1, January, 1997, pp. 78-103.
- [12] George, Alan D., "Simulating Microprocessor-Based Parallel Computers Using Processor Libraries," *Simulation*, Vol. 60, No. 2, February, 1993, pp. 129-134.
- [13] George, Alan D. and Stephen W. Cook, Jr., "Distributed Simulation of Parallel DSP Architectures on Workstation Clusters," *Simulation*, Vol. 67, No. 2, August, 1996, pp. 94-105.
- [14] Dwarkadas, S., J. R. Jump, and J. B. Sinclair, "Execution-Driven Simulation of Multiprocessors: Address and Timing Analysis," *ACM Transactions on Modeling and Computer Simulation*, Vol. 4, No. 4, October, 1994, pp. 314-338.

- [15] Rosenblum, Medel, Stephen A. Herrod, Emmett Witchel, and Anoop Gupta, "Complete Computer System Simulation: The SimOS Approach," *IEEE Parallel and Distributed Technology*, Vol. 3, No. 4, Winter, 1995, pp. 34-43.
- [16] Yan, Yong, Xiaodong Zhang, and Qian Ma, "Software Support for Multiprocessor Latency Measurement and Evaluation," *IEEE Transactions on Software Engineering*, Vol. 23, No. 1, January, 1997, pp. 4-16.
- [17] Herrod, Stephen A., "Using Complete Machine Simulation to Understand Computer System Behavior," Dissertation, Department of Computer Science, Stanford University, February, 1998.
- [18] Brewer, Eric A., Chrysanthos N. Dellarocas, Adrian Colbrook, and William E. Wehl, "PROTEUS: A High-Performance Parallel-Architecture Simulator," Technical Report TR-516, MIT/LCS, Laboratory for Computer Science, Cambridge, Massachusetts, September, 1991.
- [19] Bedichek, Robert C., "Talisman: Fast and Accurate Multicomputer Simulation," *Proceedings of SIGMETRICS '95*, Ottawa, Ontario, Canada, May, 1995, pp. 14-24.
- [20] George, Alan D., Ryan B. Fogarty, Jeff S. Markwell, and Michael D. Miars, "An Integrated Simulation Environment for Parallel and Distributed System Prototyping," *Simulation*, Vol. 75, No. 5, May 1999, pp. 283-294.
- [21] Hennessy, John L. and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, CA, 2<sup>nd</sup> Edition, 1995.
- [22] Kaeli, David R. and Philip M. Sailer, *The DLX Instruction Set Architecture Handbook*, Morgan Kaufmann, San Francisco, CA, 1996.
- [23] IEEE Standard 1596-1992 Scalable Coherent Interface (SCI), IEEE Publications Office, 1-800-678-4333.
- [24] George, Alan D., William Phipps, Robert Todd, David Zirpoli, Kyla Justice, Mark Giacoboni, and Mushtaq Sarwar, "SCI and the Scalable Cluster Architecture Latency-hiding Environment (SCALE) Project," *Proceedings of the 6th International SCI Workshop*, Santa Clara, California, September, 1996, pp. 9-24.



**Figure 1.** DLX processor subsystem and bus hierarchy



**Figure 2.** Clusters with multiple processing nodes connected via SCI

```
for(i=p*n/N; i<(p+1)*n/N; ++i)
  for(j=0; j<n; ++j)
  {
    C[i][j]=0;
    for(k=0; k<n; ++k)
      C[i][j]=C[i][j]+A[i][k]*B[k][j];
  }
```

**Figure 3.** C representation of the matrix-multiplication algorithm executed by each processor

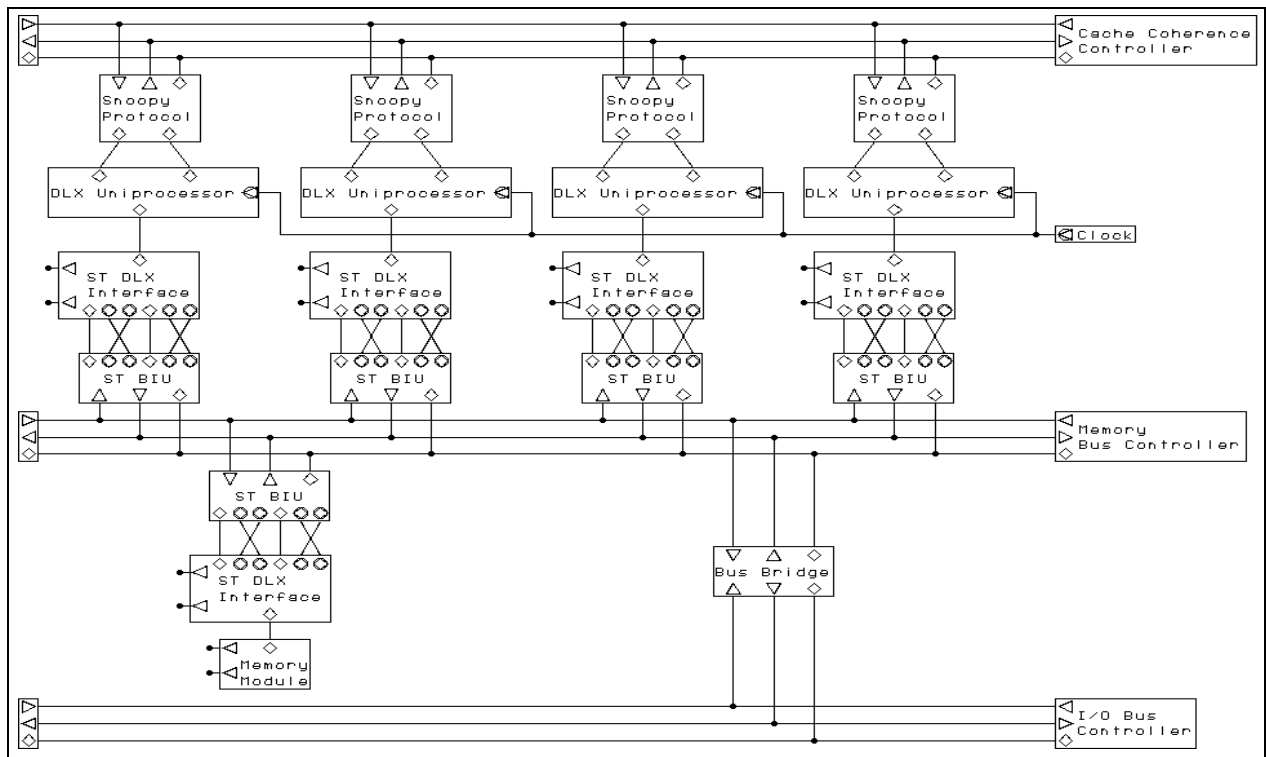
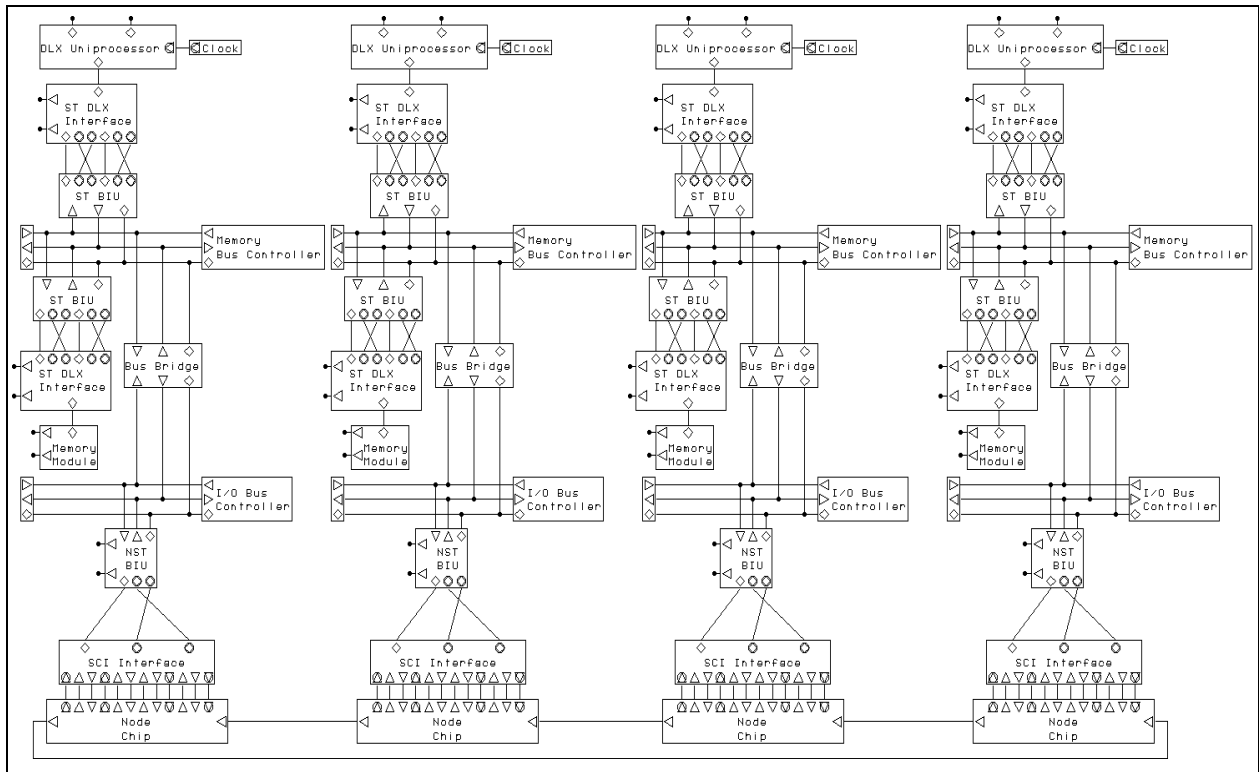
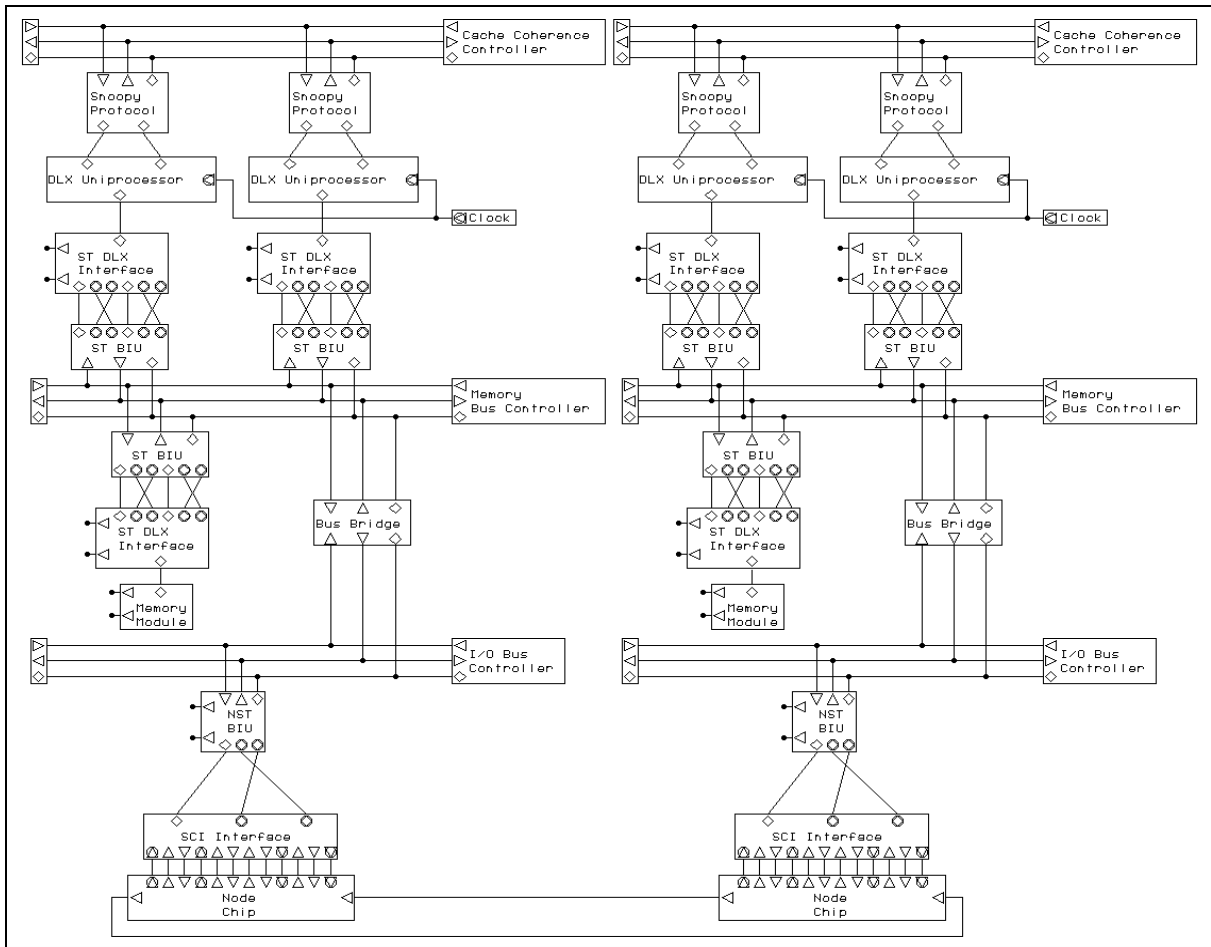


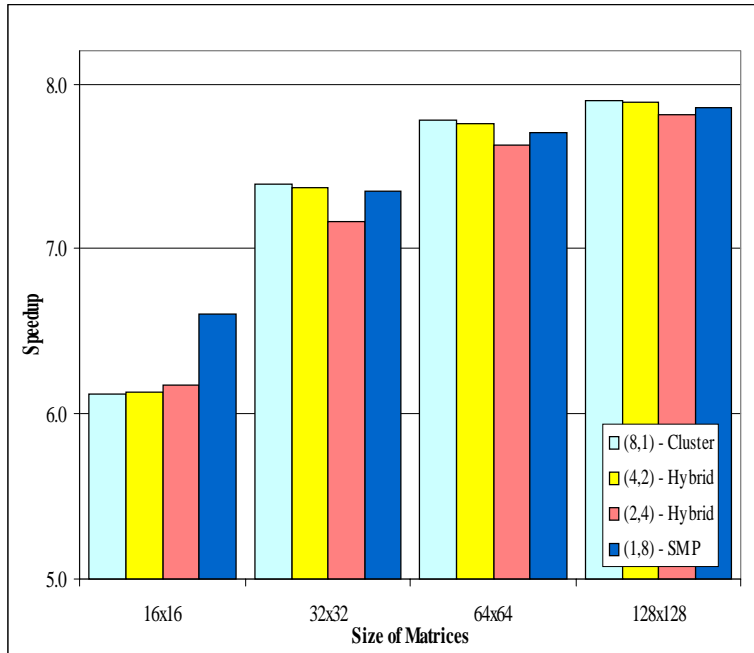
Figure 4. SMP model ( $M = 1, N = 4$ )



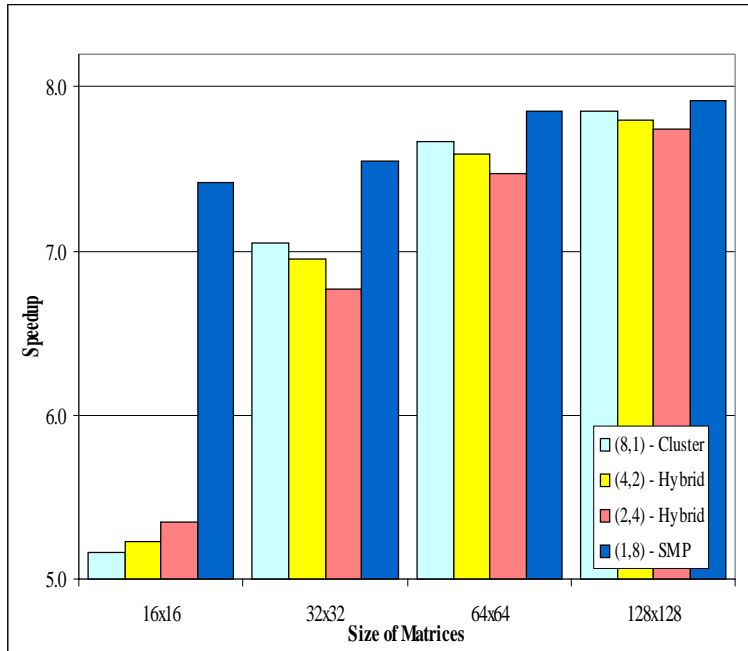
**Figure 5.** Cluster model ( $M = 4, N = 1$ )



**Figure 6.** Clustered SMP model ( $M = 2, N = 2$ )



**Figure 7.** Matrix-multiplication speedup for the eight-processor basic systems



**Figure 8.** Matrix-multiplication speedup for the eight-processor advanced systems

**Table 1.** Simulation parameters

<b>Parameter</b>	<b>Basic Value</b>	<b>Advanced Value</b>
CPU Clock Frequency	85 MHz	166 MHz
Integer Multiplication Latency	6 clock cycles	
L1 Instruction Cache Configuration	Non-blocking Cache with Victim Buffer	
L1 Instruction Cache Size	16 kB	
L1 Instruction Cache Victim Buffer Size	4 entries	
L1 Data Cache Configuration	Cache with Victim Buffer	
L1 Data Cache Size	8 kB	
L1 Data Cache Victim Buffer Size	4 entries	
L2 Cache Configuration	Cache with Victim Buffer	
L2 Cache Size	256 kB	512 kB
L2 Cache Victim Buffer Size	16 entries	
L2 Cache Access Time	20 ns	
Main Memory Access Time	60 ns	
Memory Bus Clock Frequency	40 MHz	83 MHz
Memory Bus Width	64 bits	132 bits
I/O Bus Clock Frequency	20 MHz	25 MHz
I/O Bus Width	32 bits	
SCI Data Rate	1 GB/s	
SCI Input Queue Size	5 packets	
SCI Bypass Queue Size	5 packets	
SCI Output Queue Size	5 packets	

**Table 2.** Execution times of the basic systems for all four matrix sizes

Configuration		Execution Times (ms)			
$(M, N)$	Description	16x16	32x32	64x64	128x128
(1,1)	Uniprocessor	1.307	10.237	87.230	694.439
(2,1)	Cluster	0.716	5.240	44.502	350.741
(1,2)	SMP	0.658	5.131	43.657	347.385
(4,1)	Cluster	0.373	2.696	22.304	175.553
(2,2)	Hybrid	0.372	2.706	22.350	175.750
(1,4)	SMP	0.341	2.607	21.981	174.290
(8,1)	Cluster	0.214	1.384	11.220	87.968
(4,2)	Hybrid	0.213	1.389	11.243	88.082
(2,4)	Hybrid	0.212	1.428	11.440	88.857
(1,8)	SMP	0.198	1.394	11.330	88.376

**Table 3.** Communication composition of the basic systems for the 64×64 matrix multiplication

Configuration ( $M, N$ )	Memory Bus Transaction (ns)	Cache Coherence (ps)	SCI Transaction (us)		Memory Operation (ns)	Total Execution Time (ms)
			Read	Write		
(1,1)	72.34	0.00	0.00	0.00	60.00	87.23
(2,1)	75.66	0.00	6.61	2.49	60.61	44.50
(1,2)	83.71	9.53	0.00	0.00	60.06	43.66
(4,1)	84.47	0.00	6.65	2.54	61.45	22.30
(2,2)	89.08	9.64	7.40	2.56	60.72	22.35
(1,4)	175.87	9.79	0.00	0.00	66.61	21.98
(8,1)	100.04	0.00	6.83	2.63	60.18	11.22
(4,2)	104.30	9.86	7.59	2.64	60.79	11.24
(2,4)	178.75	10.16	11.33	3.15	70.10	11.44
(1,8)	533.02	10.74	0.00	0.00	90.30	11.33

**Table 4.** Execution times of the advanced systems for all four matrix sizes

Configuration		Execution Times (ms)			
$(M, N)$	Description	16x16	32x32	64x64	128x128
(1,1)	Uniprocessor	0.671	5.251	47.875	381.119
(2,1)	Cluster	0.391	2.731	24.726	193.702
(1,2)	SMP	0.337	2.631	23.957	190.632
(4,1)	Cluster	0.209	1.431	12.394	96.933
(2,2)	Hybrid	0.207	1.455	12.530	97.585
(1,4)	SMP	0.173	1.326	12.013	95.450
(8,1)	Cluster	0.130	0.745	6.240	48.562
(4,2)	Hybrid	0.128	0.755	6.304	48.880
(2,4)	Hybrid	0.125	0.776	6.408	49.244
(1,8)	SMP	0.090	0.696	6.094	48.130

**Table 5.** Communication composition of the advanced systems for the 64×64 matrix multiplication

Configuration ( $M, N$ )	Memory Bus Transaction (ns)	Cache Coherence (ps)	SCI Transaction (us)		Memory Operation (ns)	Total Execution Time (ms)
			Read	Write		
(1,1)	29.48	0.00	0.00	0.00	60.00	47.88
(2,1)	30.28	0.00	6.02	2.23	60.22	24.73
(1,2)	32.02	9.53	0.00	0.00	60.15	23.96
(4,1)	32.39	0.00	6.08	2.25	60.60	12.39
(2,2)	33.38	9.64	8.23	2.26	63.17	12.53
(1,4)	42.84	9.79	0.00	0.00	74.86	12.01
(8,1)	36.14	0.00	6.17	2.31	60.27	6.24
(4,2)	36.89	9.86	8.25	2.29	62.10	6.30
(2,4)	44.19	10.16	10.16	2.60	70.22	6.41
(1,8)	96.24	10.74	0.00	0.00	116.56	6.09