Empirical Analysis of Isotach Networks

Paul F. Reynolds, Jr.
Craig Williams
Raymond R. Wagner, Jr.

Computer Science Report No. TR-92-19
June 1992

Empirical Analysis of Isotach Networks

Abstract

This paper presents the results of a simulation study designed to evaluate the performance of isotach networks. An isotach network is an interconnection network that provides hardware support for atomicity and sequential consistency in parallel computations. The results show conventional networks are more efficient than isotach networks under a workload with no atomicity or sequencing constraints, but that when such constraints are a significant factor in the workload, isotach networks outperform conventional networks. Isotach networks perform best in relation to conventional networks when execution is required to be sequentially consistent, data dependent operations are far enough apart that they do not restrict pipelining, contention for shared variables is high, and atomic actions are large. When two or more of these characteristics apply to the workload, isotach networks can outperform conventional networks by an order of magnitude or more.

1.0 Introduction

The isotach network [RWW89, WiR91] is a new type of interconnection network designed to provide hardware support for the task of coordinating and synchronizing processes in a parallel computation. The isotach network is so named because it implements a logical time system in which each message in the network progresses towards its destination at the same rate: one switch per logical time unit. This characteristic property of the isotach network, called the *velocity invariant*, makes the network a powerful coordinating mechanism. The velocity invariant makes it possible for a process to control the logical time at which its operations are received and executed through its control over when it emits the operations. This control over the logical time at which its operations are executed at memory enables processes to access multiple shared variables atomically without acquiring locks and to pipeline operations without risk that the operations be executed out of order, in contrast with conventional, i.e., non-isotach-based, systems that typically use locks and delays to achieve the same results.

The way an isotach network works is very similar to the way a conventional network of the same topology works except that the individual switching elements within an isotach network operate in a *locally synchronous* mode -- each switch applies a simple list-merge algorithm in determining the order in which to route incoming operations with the result that each switch stays loosely synchronized with its immediately neighboring switches. Local synchrony is the basis for the individual switch's ability to enforce the velocity invariant using only local information, but it does have a cost: an operation may be delayed at a switch in an isotach network when it would not be delayed in a conventional network. Thus the "raw power" of the isotach networks, i.e., the throughput and latency of the networks under a workload with no synchronization constraints, can be expected to be worse than that of a comparable conventional network.

This paper reports on a simulation study of the performance of isotach networks. Among the questions studied are the following:

1. How much lower is the raw power of an isotach network than a conventional network?
2. Under what conditions, if any, does an isotach network make up for the expected loss in raw power through more efficient support of synchronization?

The simulation also provided an opportunity to explore alternative implementations of isotach networks and to study the effect of various network and workload parameters, such as the network size, number of network buffers, average size of atomic actions, and traffic model.

All the networks we simulated, both isotach and conventional, are clocked, baseline networks that use store-and-forward routing. The simulated workloads assume a MIMD shared memory computation with no support for private caching of shared variables. The study does not cover the full range of isotach networks. Isotach networks can take other forms and can support other programming models.

The simulation compares isotach systems using isotach based synchronization techniques to conventional systems that enforce atomicity with two-phase locking (2PL) [EGL76] and that enforce sequential consistency by restricting pipelining. We ran the simulations under a variety of synthetic workloads to capture different atomicity, sequencing, and data dependence constraints among operations. The results of the simulation confirm that an isotach network has lower throughput and higher latency than a comparable conventional network, but that an isotach system is more efficient than a conventional system in executing computations with significant synchronization requirements. We believe the reason the conventional systems perform relatively poorly in these computations is that the means by which they enforce atomicity and sequencing constraints, i.e., locks and delays, slow execution by restricting concurrency and prevent the conventional system from taking advantage of its higher raw power.

This paper is organized as follows. Section 2 defines the isotach network and explains how the network supports synchronization. Section 3 describes the networks simulated. Section 4 describes the simulation and reports the results. Section 5 summarizes the results and describes ways in which we intend to extend our study of isotach network performance.

2.0 Isotach Networks

This section defines the isotach network, describes an implementation, and shows how an isotach network supports process coordination. The material presented in this section is a summary of previous work [RWW89, WiR89, WiR91], repeated to make this report self-contained.

2.1 A Logical Time System

The isotach network is defined in terms of the logical time system it implements. Informally, a logical time system is a set of rules for numbering events of interest, that is, assigning each event a *logical time*, such that the times assigned are consistent with causality, e.g., if event *a* causes event *b*, a logical time system assigns *a* a lower, i.e., *earlier*, time than it assigns *b*.

In the logical time system implemented by an isotach network, a time is assigned to each event of emitting a message into the network and of receiving a message from the network. Each logical time is, in general, an n-tuple of integers, in which the first and most significant component is called the *pulse* component. In the case of the isotach networks we simulated, each logical time is a 4-tuple of integers. Logical times are compared component-wise and are lexicographically ordered. The times assigned are consistent with the *happens-before* relation in the logical time system defined by Lamport [.Lam78.], i.e., the logical time assigned to the event of emitting a message into the network is less than or equal to the time assigned to the event of receiving the message, and for any two events occurring at the same node, the logical times assigned are consistent with the order in which the events actually occur.

The logical time system implemented by an isotach network differs from others [Lam78, Mat88, Fid91] in that it relates communication time with communication distance. In an isotach network each message is received exactly DIST pulses after it is emitted, where *DIST* is the number of switches through which the message is routed, i.e., a message emitted at time $t_{emit} = (i, v, j, k)$ is received at time $t_{receive} = (i + DIST, v, j, k)$. An isotach network maintains the following invariant, the *velocity invariant*:

$$\mathrm{DIST}/(t_{receive} - t_{emit}) = 1 \text{ switch/pulse}$$

In other words, all messages in an isotach network progress towards their destination at the same velocity -- one switch per pulse of logical time.

2.2 Local Synchrony

This logical time system can be implemented in a distributed way using a form of synchronization we call *local synchrony.* Each switch in a locally synchronous network stays loosely synchronized with its immediate neighbors through pulses of control signals called *tokens* that propagate through the network. In more vivid terms, the network pulses like a heart. These pulses supply the timing mechanism for a distributed logical clock. Local synchrony has been used by Ranade in a CRCW-PRAM emulation, by Awerbuch to support execution of SIMD graph algorithms on asynchronous networks [.Awe85.], and by Birk, et al., to support barrier synchronization [.BGS89.].

An isotach network can be implemented on many different types of networks and for many different reasons. We restrict this discussion to the type of system simulated: an equidistant network that supports process coordination in shared memory model (SMM) computations with no private caching of shared variables. In an equidistant network, sometimes called a *dance hall* network, the length of every routable path from a processing element (PE) to a memory module (MM) is the same. The implementation requires reliable FIFO communication links. We assume processes communicate only by accessing shared memory. Every message is either an *operation,* i.e., an instruction accessing a shared variable, or a *response*, i.e., a reply to an operation. A system of the type simulated requires that the velocity invariant be maintained only for operations, i.e., only in the forward, PE to MM, direction. We assume the reverse network is a conventional network.

For simplicity, each PE and MM is assumed to be connected to the network via a switch interface unit (SIU). The SIUs are responsible for assigning logical times to the events of interest -- the events of emitting and receiving operations. Each SIU maintains a local logical clock which it uses in assigning logical times. A local logical clock is simply a variable that records the current logical time, i.e., the time assigned the last local event.

The clocks at the SIUs are coordinated by local synchrony as follows. Initially each switch emits a token pulse, i.e., it emits a token on each output, including, for the switches in the first and last network stages, the output to each adjacent SIU. Each switch thereafter emits token pulse *i* after receiving token *i-1* on all inputs, including each adjacent SIU, if any. Thus the token pulses keep each switch loosely synchronized with its neighbors. The token pulses also drive logical clocks at the SIUs. The pulse component of the logical time at each SIU is the number of tokens that have passed through the SIU.

Between tokens the SIU for a PE (PE-SIU) may emit zero or more operations. Before emitting each pulse of operations, a PE-SIU first sorts the operations in increasing order by address of the variable accessed, using a stable sort to preserve the issue order among operations on the same variable. Each operation receives a timestamp. The timestamp for operation op_i, denoted $ts(op_i)$, is the 4-tuple (*pulse, var, pid, rank*), where *pulse* is the pulse in which op_i is emitted by the PE-SIU, *var* is the variable accessed, *pid* is the identifier of the issuing PE, and *rank* is op_i's issue rank within the pulse, i.e., the rank component of the timestamp for op_i is j if op_i is the *jth* operation emitted by that PE-SIU in the current pulse. Note that a PE-SIU emits operations in timestamp order and that timestamps are unique. This technique for assigning unique timestamps without centralized control is widely used in database concurrency control [RSL78, Ree83]. When a PE-SIU emits an operation it updates the local clock, setting the clock equal to the timestamp of the operation. Each PE-SIU emits operations in timestamp order so each PE-SIU's clock moves forward monotonically.

Between token pulses, each switch routes operations as usual except it chooses messages to route in timestamp order. Each 2x2 switch continuously merges the 2 sorted lists arriving on its inputs to produce 2 sorted output lists. As it routes each message, the switch increments the pulse component of the message's timestamp. Since each SIU emits operations in timestamp order and timestamp order is maintained at each switch and across each link, operations are received at each MM in timestamp order. Consider the tree of switches rooted at a given MM with leaves at each PE-SIU. A simple induction on the depth of the tree shows that operations arrive at the root MM in strictly increasing order by timestamp. Each SIU for an MM (MM-SIU) maintains a local clock in the same way as a PE-SIU, except an MM-SIU updates its clock for receive events.

The velocity invariant holds because a message with timestamp (i, v, j, k) arriving at a switch in pulse i (after the *ith* token received on the input on which the message arrives) leaves with timestamp $(i+1, v, j, k)$ in pulse $i+1$ (after the *i+1st* token pulse). Since traveling through a switch adds 1 to the pulse component of a message's timestamp and does not otherwise change the timestamp, a message emitted at time (i, v, j, k) is received at time $(i+DIST, v, j, k)$. The logical times at which operations are emitted and received are also consistent with Lamport's *happens-before* relation since an operation is always received at a later logical time than it is emitted and the logical clocks at all SIUs move forward monotonically. This isotach network implementation is also deadlock-free [ReW91].

The implementation described here is an abstract implementation intended to be useful in reasoning about the isotach network, but is not an implementation we recommend for an actual system. Though the tokens are necessary, the timestamps and logical clocks are not. Operations need carry only the information they carry in conventional networks. The algorithm actually simulated is described in section 3.

2.3 Atomicity and Sequential Consistency

The isotach network provides hardware support for the task of controlling the concurrency of parallel computations, in particular, in enforcing atomicity and sequential consistency.

Atomicity and sequential consistency are important properties of parallel executions that are expensive to enforce using existing techniques. The atomic action, a group of operations required to be executed as an indivisible step, was developed in the context of database concurrency control by Eswaran [EGL76]. Early proposals for using the atomic action as a device for structuring parallel programs include those of Owicki and Gries [OwG76] and Lomet [Lom77]. The atomic action is a group of one or more instructions issued by the same process that appears to be executed indivisibly, without interleaving with other instructions. In database concurrency control and in some programming languages designed for geographically distributed systems, the atomic action is also a unit of recovery, i.e., the instructions in an atomic action are executed on an all-or-nothing basis, even in the presence of hardware failures. In our research, we have assumed hardware reliability.

Atomic actions are typically specified by operations on locks or semaphores or by critical sections or monitors implemented with locks or semaphores. Existing hardware support for atomic actions is designed to make locking more efficient. The principal drawback to the use of locks in implementing atomicity is the unnecessarily restricted access to shared variables implied by locking. Variables cannot in general be partitioned so that each atomic action can, by acquiring a single lock, control all the variables it must access and no others. Atomic actions either acquire a single lock, and lock some variables unnecessarily, or acquire multiple locks. To avoid deadlock, an atomic action that requires multiple locks must typically obtain the locks sequentially. During this lock acquisition phase, the variables controlled by already acquired locks are unavailable to other processes.

The second property, sequential consistency, means that the order in which assesses are executed is consistent with the order specified by each individual process's sequential program [Lam79]. Maintaining sequential consistency is a problem in multiprocessors because stochastic delays in the network allow operations issued by the same process to arrive at the MM's in an order inconsistent with the order in which the operations were issued. The simplest solution, disallowing pipelining of memory accesses, is undesirable since pipelining is an important way to lessen effective memory latency. An isotach network enforces sequential consistency without restricting the pipelining of operations.

The isotach network was originally designed to support the *isochron*, a synchronization primitive derived from the *parallel operation* [Wag87, RWW89]. The isochron is an atomic, sequentially consistent multicast. In SMM terms, it is a group of one or more operations that appear to be executed as an indivisible step. A process can pipeline isochrons without risk that the operations be executed in the wrong order.

Given the velocity invariant, implementing isochrons on a equidistant isotach network is straightforward. We require only that

1. each MM execute operations in FIFO order;

2. each PE-SIU emit all operations from the same isochron in the same logical pulse; and
3. if a PE-SIU emits more than one operation on the same variable in the same pulse, it emit the operations in the order in which they were issued.

We have shown that these rules ensure atomic and sequentially consistent execution of isochrons by proving that for each parallel execution, an equivalent serial execution can be constructed from the logical times at which operations are received at memory, such that operations in each isochron are executed without interleaving and operations issued by the same process are executed in the order in which they were issued.

The isotach network supports atomic execution of isochrons, but isochrons represent only a limited class of atomic actions. Operations in an isochron must be issued as a batch, so operations with data dependencies cannot be executed in the same isochron. We have proposed techniques based on isochrons together with *access sequences* and *split operations*, defined in a previous report [WiR89], to support a broad class of atomic actions. An access sequence for a variable is the sequence of elements representing accesses made to the variable over time. Each element in the access sequence either records the value read or written by an access or reserves a position for an access. Split operations are the set of operations defined for the access sequence representation and are based on the idea of splitting an access into two steps -- a scheduling step that appends an element to the access sequence to reserve the context for an access and an assignment step that transfers a value. For writes, the transfer is from a process's local variable or register to the element appended by the scheduling step for the write. For reads, the direction of transfer is reversed, from the access sequence to the local variable or register. Splitting a write into two steps allows the write to be scheduled before the value to be written is known. A process schedules a write by issuing a SCHED operation and completes a previously scheduled write by issuing an ASSIGN operation. The steps can be collapsed into a single step (initiated by a WRITE operation) if the process knows the value to be written when it schedules the write. An MM executes a SCHED operation by appending an element with the special value nil denoted λ and returning the identifier *eId* of the element. When it determines the value to be written, the process executes an ASSIGN operation containing both the value and the *eId* returned in response to the SCHED. The *eId* enables the MM to assign the value to the element reserved for it. In the case of reads, the assignment step is initiated by memory, so the set of split operations contains only one operation relating to reading variables. A process schedules a read by issuing a READ operation and the MM executes the READ by 1) identifying the preceding write, i.e., the write whose SCHED reserved the element most closely preceding the read in the access sequence, and 2) if the assignment step for that write has been executed, i.e., if the value of the element reserved for the write is not λ, returning the value assigned. If the assignment step for the preceding write has not yet been executed when the MM executes the READ, the MM responds to the READ when the assignment is executed. As part of executing an ASSIGN operation, an MM sends the value assigned to all of the reads on the same variable scheduled immediately after the write.

A process executes an atomic action by issuing an isochron that schedules all the accesses required for the atomic action, executing the assignment steps for those accesses as it determines the values to be assigned. For example, a process executes the assignment $A=B+C$ atomically, where A, B, and C are all shared variables, by 1) issuing an isochron containing a SCHED operation on A and READ's on B and C; 2) waiting until it receives the value of B and C in response to the READ's and the *eId* of the element reserved for the write in response to the SCHED; and 3) issuing an ASSIGN operation containing both the *eId* and the value to be written, i.e., $B+C$.

Execution is atomic because the isochron used to schedule the accesses reserves a consistent "time slice" across the histories of the accessed variables. This technique, called the *scheduling isochron* technique, works for atomic actions with access sets that can be determined at the beginning of execution of the atomic action. We have proposed variations on the technique for atomic actions with data dependent access sets [WiR89]. The simulation described below is limited to atomic actions that can be executed using the scheduling isochron technique.

3.0 Simulated Networks

The architectural model assumed in this study is a MIMD shared memory parallel processor based on a multi-stage switching network. Four networks are simulated in the study, two conventional (C1 and C2) and two isotach (I1 and I2). Each network is composed of 2x2 switches interconnected in the same baseline network topology. A diagram of a multistage interconnection network with this topology is shown in Figure 1. The message transmission protocol is store-and-forward using a send-acknowledge protocol [REF87]. To allow the use of time-stepped simulation, we assume the networks are clocked, i. e., switches begin each cycle simultaneously. We believe the results are also applicable to self-timed networks.

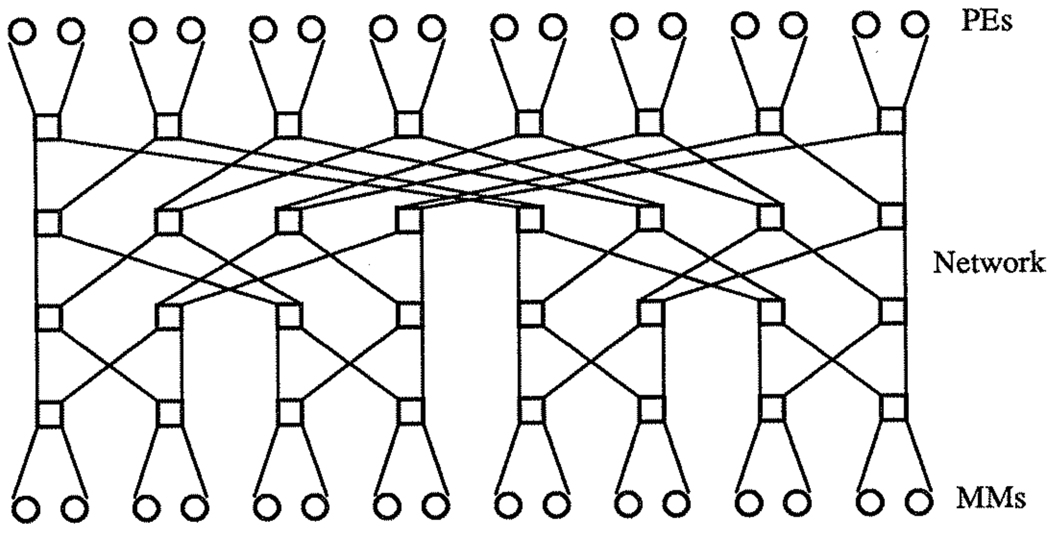


FIGURE 1. A 4 stage interconnection network.

The networks differ only in the design of the individual switches and in the algorithms the switches execute. Networks C1 and I1 are both composed of 2x2 crossbar switches with input buffer queues. Networks C2 and I2 use more sophisticated switches, called z-switches, that yield higher throughput at some cost in latency. We developed the z-switch to improve the throughput of the isotach network. When it proved successful, we translated it into a conventional switch design to enable us to compare the z-switch version of the isotach network (I2) to a conventional network with comparable routing advantages (C2). In the context of a conventional network, the z-switch is similar to switch designs with output buffers [KHM87] or internal buffers [KuJ84].

We assume switches in C1 and I1 have the same cycle time, i. e., we assume that the amount of time required in the absence of conflicts for a message to travel through a switch in C1 is the same as in I1. All performance data is measured in units of this cycle time. The switches in networks C2 and I2 are assumed to have twice the latency of those in C1 and I1.

3.1 Conventional Networks Simulated

This section describes the two conventional networks, C1 and C2, included in the study for comparison with the isotach networks.

3.1.1 Conventional network C1

The switches in network C1 consist of two input queues, two output buffers, and a routing unit that can route a message from either of the two inputs to either of the two outputs. Each buffer can hold a single message and each queue consists of one or more buffers. A diagram of the switch architecture appears in Figure 2.

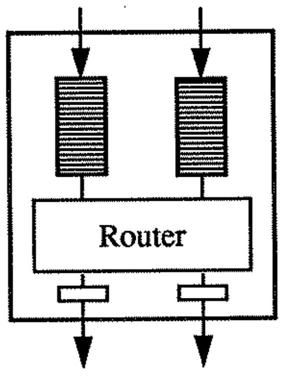


FIGURE 2. Simple switch design (C1 and I1)

The switching cycle of switches in network C1 consists of two steps: a "route" step in which each switch attempts to route messages on its inputs to its outputs, and a "push" step in which each switch attempts to send the messages on its outputs to the inputs of switches at the next stage using a send-acknowledge protocol. The route and push steps at each switch are as follows:

Route Step:

1. Choose an input randomly such that each input is chosen with equal probability.
2. If the input holds a message, determine the output on which the message should be routed.
3. If the output is available, route the message, i.e., move it to the correct output.
4. Repeat steps 2 and 3 with the other input. (Note that a switch is capable of routing two messages in a single route step.)

Push Step:

1. For each non-empty output, send a copy of the message on the output to the next stage switch connected to the output.
2. For each input on which a new message arrives, if room in the input queue is available, add the message to the queue and send an ACK to the sender.
3. For each output for which an ACK is received, mark the output empty.

3.1.2 Conventional network C2

Network C2 is a high-throughput conventional network based on the z-switch. In network C2, switch outputs are decoupled from switch inputs to provide better throughput at some cost in network latency. Decoupling inputs from outputs prevents a blocked message from blocking another message merely because it arrives on the same input. In a z-switch, a message whose output buffer is available can make progress even though a message arriving previously on the same input is blocked due to a full output buffer.

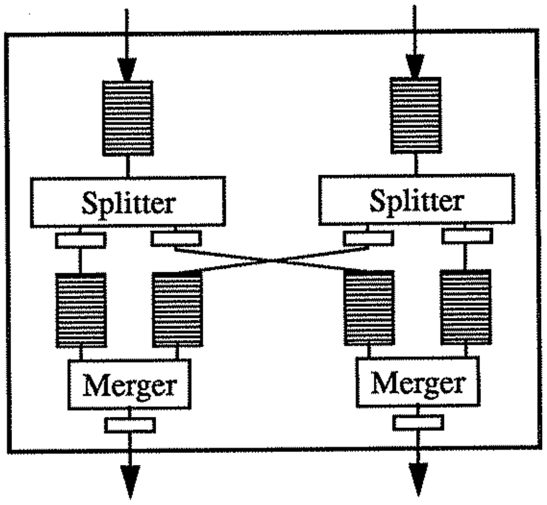


FIGURE 3. Z-switch high-throughput design.

Each switch element consists of four components: a "splitter" for each switch input and a "merger" for each switch output. Each splitter has one input queue and two output buffers, one for each merger. Each merger has two input queues, one for each splitter, and one output buffer. A diagram of the switch design appears in Figure 3.

As in C1, each switch cycle in C2 consists of a route step and a push step. Each splitter and merger participates in both steps of the cycle:

Route Step - Splitter:

1. If the input is empty, skip the route step.
2. Determine the output on which the incoming message should be routed.
3. If that output is available, route the message.

Route Step - Merger:

1. If the output is full, skip the route step
2. Randomly choose an input.
3. If the input holds a message, route it.
4. Otherwise, route the message, if any, on the other input.

Push Step - Splitter:

1. For each non-empty output, send a copy of the message on the output to the merger connected to the output.
2. If a new message arrives on the input and room in the input queue is available, add the message to the queue and send an ACK to the sender.
3. For each output for which an ACK is received, mark the output empty.

Push Step - Merger:

1. If the output holds a message, send a copy of the message to the next stage switch connected to the output.
2. For each input on which a new message arrives, if room in the input queue is available, add the message to the queue and send an ACK to the sender.
3. If an ACK is received for the output, mark the output empty.

The latency time for C2 is assumed to be two cycles because the splitter and merger each have one cycle latency time.

3.2 Isotach Networks Simulated

The two isotach networks simulated, I1 and I2, correspond to C1 and C2, respectively. In each isotach network the structure of the switches is similar to that of the switches in the corresponding conventional net, but the switch algorithm is different. The isotach switches apply the local synchrony algorithm described in section 2. The local synchrony algorithm requires that the switch route messages arriving in the same pulse in a given order. In isotach networks I1 and I2, each switch routes operations in non-decreasing order by "route-tag". The "route-tag" of an operation is the ordered pair "(dest, source)", where "dest" is the variable accessed by the operation and "source" is the pId of the source PE. Route-tags are lexicographically ordered.

Isotach networks carry two types of messages in addition to operations: tokens and ghosts. As described in section 2, tokens are control signals that divide pulses. A ghost [Ran87] is a copy of an operation with a bit set to indicate it is not a real operation. In other respects, ghosts look like operations. In particular, ghosts have route-tags. When a switch sends an operation on one output it sends a ghost with the same route-rag as the operation on the other. A switch receiving a ghost knows that all further operations it receives on the same input in the same pulse will have a larger route-tag than the ghost. This knowledge may enable the switch to route an operation on its other input. Ghosts improve network performance and are necessary in some networks for deadlock freedom [ReW91].

3.2.1 Isotach network I1

In network I1 the switches have the same structure as the switches in C1, i.e., each switch has two input queues, two output buffers, and a router.

Each switch in I1 records the route-tag of the last message it routed as "last_tag". The value of "last_tag" represents the best conservative guarantee the switch can make about the route-tag of the next message it will route. Initially, and at the beginning of each pulse, the value of "last_tag" is reset to (-1,-1).

Each switch in I1 always chooses the minimum message for routing, i.e., the message with the minimum route-tag. Identifying the minimum message requires that both inputs hold a message. In identifying the minimum message, tokens are treated as having a route-tag greater than that of any other type of message and operations and ghosts are treated identically. If both inputs hold tokens, the minimum message is chosen arbitrarily. If only one input holds a token, the message on the other input is the minimum message. If neither holds a token, the minimum message is the message with the lowest route-tag.

Whenever it has no operation or token to route on an output, a switch routes a ghost. Ghosts take up only unused bandwidth and unused buffers. A ghost can always be overwritten by a newer message. Therefore an output buffer is "available" if it is empty or contains a ghost, and room in an input queue is "available" if the queue is not full or if the message at the tail is a ghost.

The route and push steps in I1 are as follows:

Route Step:

1. Determine the input holding the minimum message.
2. If the minimum message is a token and both outputs are available, remove the tokens from both inputs, place tokens on both outputs, and reset last_tag. (Note that if the minimum message is a token then the other input also holds a token.)
3. Otherwise, set last_tag equal to the route-tag of the minimum message. If the minimum message is an operation, determine the output on which it should be routed. If the output is available, route the operation.
4. Emit a ghost with route-tag = last_tag on each available output.

Push Step:

Same as the push step in C1.

Network I1 has low throughput in relation to C1 because the switches in I1 can route only one message per cycle. A switch in an isotach network that can see only the operations at the head of each of its inputs has insufficient information to route more than one operation per cycle given the requirement that a switch must route operations in sorted order. The z-switch used in network I2 was designed to overcome this throughput limitation.

3.2.2 Isotach network I2

The switches in network I2 have the same structure as those in C2. However the units called splitters in C2 are called multiplexors in I2 to better represent their function in I2. The function of a multiplexor is to copy each operation received on its input onto both outputs. The function of a merger is to route the message streams received from both multiplexors in order by route-tag. Each multiplexor and merger records the last_tag as in I1.

The route and push steps in I2 are as follows:

Route Step - Multiplexor:

1. If the input holds a token and both outputs are available, consume the token, put tokens on both outputs, and reset last_tag.
2. Otherwise, set last_tag equal to the route-tag of the message on the input. If the input holds an operation, determine the output on which it should be routed, and move the message to that output if it is available.
3. Emit a ghost with route-tag = last_tag on each available output.

Route Step - Merger:

1. If the output is not available, skip the route step.
2. Determine the input holding the minimum message.
3. If the minimum message is a token, remove the tokens from both inputs, place a token on the output, and reset last_tag.
4. Otherwise, set last_tag equal to the route-tag of the minimum message and, if the minimum message is an operation, move it to the output.
5. If the output is available, emit a ghost with route-tag = last_tag.

Push Step - Multiplexor:

Same as the push step for the splitter in C2.

Push Step - Merger:

Same as the push step for the merger in C2.

The simulation data presented in the next section compares each of the four networks presented here. Isotach networks, as simulated, are simple variations of the networks described here, modified to allow piggybacking of tokens. Piggybacking of tokens occurs when a switch knows that a token follows the current message, and sends the message with a 'token bit' set, instead of sending the message followed by a token. Piggybacking thus reduces traffic in the network.

4.0 Performance Results

This section presents results from simulations of the four networks described in the previous section. Our assumption is that networks C1 and I1 have the same switch cycle time, and that C2 and I2 have latency double that of C1 and I1.

Our simulation environment allows us to present data for each network and to vary a number of operational parameters and workload models. The simulation parameters include the following:

Network Size - The number of PE/MMs.

Variables/MM - The number of shared variables per MM.

Network Switch Queue Size - The maximum number of messages that a switch input queue can hold. In C2 and I2, the queue size applies to the input queues of both the splitter/multiplexer and the mergers.

READ probability - The probability that any given access is a READ, as opposed to a WRITE.

Atomic Action Mean - In workload models that include atomic actions, the average number of operations in an atomic action. The sizes of atomic actions are from an exponential distribution with a mean equal to the mean supplied as a parameter and a range from 1 to (10 * (mean-1)) + 1.

Forced Singleton Probability - A user defined probability p. The probability that an atomic action is a singleton, i.e., that it is of size 1, is $p + q(1-p)$, where q is the probability of choosing 1 from the distribution of atomic action sizes described above.

Atomic Action Cap - In workload models that include atomic actions, the cap on the number of atomic actions that any PE may have started and not completed at any given time.

Traffic Model - A set of parameters that describe the distribution of operations on variables. The simulation uses three different types of traffic models: a uniform model, in which all variables are accessed with equal probability; a hot-spot model, in which one variable is accessed with greater probability; and a warm-spot model, in which several variables are accessed with greater probability.

Workload Model - A set of rules governing the production and execution of operations. The simulation uses many different workload models, ranging from a "raw power" workload, in which PE's produce a saturation or probabilistic load of generic operations with no constraints on the order in which operations are executed, to a workload with atomicity, sequencing, and data-dependency constraints among operations.

The study is divided into six series of simulations, each focusing on a different workload model, or, in one case, traffic model. For each series, a 'base case' is defined giving the default parameter settings. The base case for each parameter and the range of values tested is presented in table 4.1. In the remainder of this section, we describe each series and its results.

Parameter	Base Case Value	Other Values Tested
Network Size	5 stages (32 PE/MMs)	4, 6, 8, 10
Variables/MM	32	-
Network Switch Queue Size	1	2, 3, 4, 5, 6
READ Probability	.75	0, .25, .50, .90, 1.0
Atomic Action Mean	3	1-10, 16
Atomic Action Cap	-	1, 3, 6, 12, 16, 20, 24, unlimited
Forced Singleton Probability	0	0, .1, .25, .5, .75
Traffic Model	Uniform	Hotspot, Warmspot

TABLE A - Value Ranges for Simulation Parameters

All data points represent results from at least 10 independent runs. For data points at which runs give results with a high variance, we report 99% confidence intervals based on 16 runs.

The following legend applies to all graphs presented. The parenthetical descriptions of the differences between data lines apply only to series 3 and 6.1. In other series, only the first legend for each network will appear.

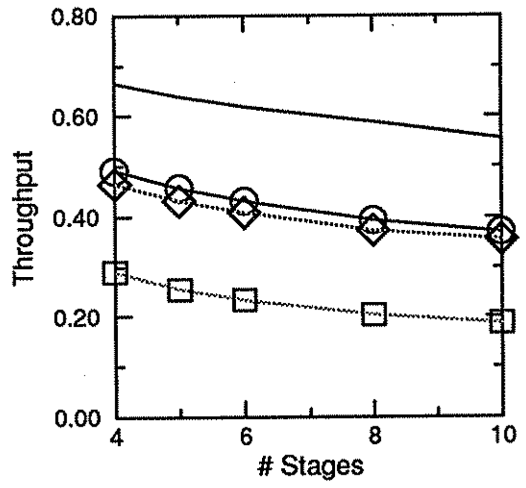
⊖——⊖	Network C1
⊡┈┈⊡	Network I1 (No AA Cap)
◇┈┈◇	Network I2 (No AA Cap)
■┈┈■	Network I1 (AA Cap = 3)
◆┈┈◆	Network I2 (AA Cap=3)
——	Network C2
|	99% Confidence Interval

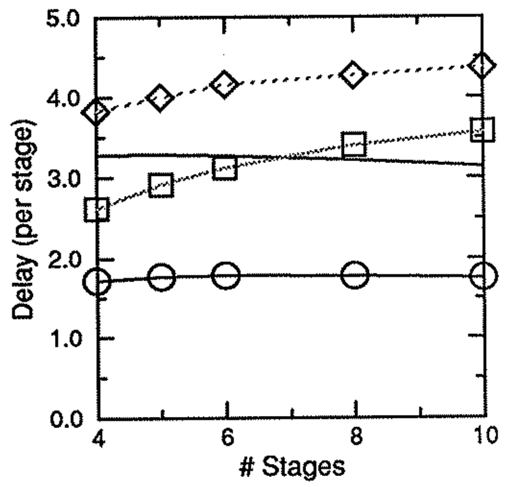
4.1 Raw Power Data

The first series measures the "raw power" of each of the networks. We use the term "raw power" to mean network throughput and delay under a workload of operations with no atomicity or sequencing constraints. The workload model consists of generic, independent operations -- reads and writes are not distinguished and operations may arrive at memory in any order. In the base case the load on the network is a saturation load. A PE generates a new operation whenever its output buffer is empty. Only the forward, i.e., PE->MM, network is simulated in this series. Each MM is a sink for operations. We assume the memory cycle time is the same as the switch cycle time -- each MM can consume one operation per cycle.

In I1 and I2 enforcement of atomicity is relaxed by specifying an isochron size of one. No comparable way exists in I1 or I2 to relax enforcement of sequential consistency. Thus series 1 actually compares isotach networks that enforce sequential consistency with conventional networks that do not.

Unless otherwise stated, the delay reported in series 1 is the network delay per stage. The delay does not include source queueing or other non-network delay such as delay at the network interface. More specifically, the delay is the average number of cycles between the time an operation is sent from the PE-SIU to the first-stage switch until the time it is sent from the last stage switch to the MM-SIU for the destination MM, divided by the number of network stages. In this series the throughput reported, unless otherwise stated, is the average number of operations arriving at memory per cycle, divided by the number of MMs.



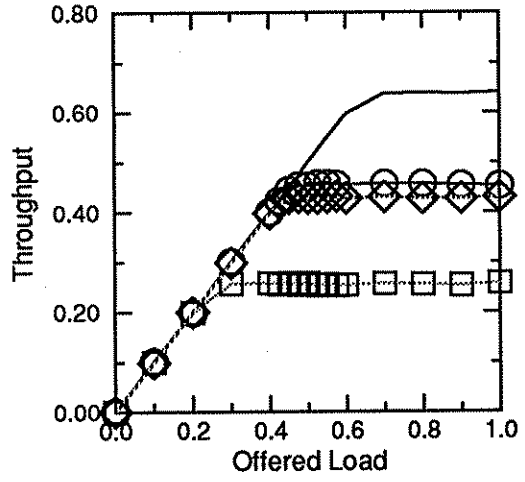


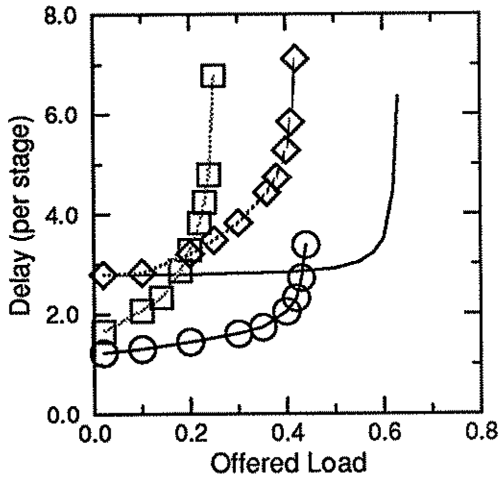
GRAPHS 1-Scalability - Data for series 1, variation of network size.

Base Case - Base case performance of the networks is shown in several of the series 1 graphs, e.g. in Graphs 1-scalability at stages = 5. Each isotach network has lower throughput and higher delay in the base case than the corresponding conventional network. I2's throughput is two-thirds that of C2, and its delay is one-fourth longer than C2's. I1's throughput is slightly over half C1's and its delay is two-thirds longer than C1's. Although the difference in the raw power of each pair of corresponding networks is significant, the faster of the isotach networks has delay in the base case comparable to that of the slower of the conventional networks and the high throughput isotach network has throughput comparable to that of the low throughput conventional network.

In each part of the series, one parameter is varied while each of the other parameters remains at its base case setting.

Scalability - Graphs 1-scalability show the effect of increases in the number of network stages on network throughput and delay. The throughput of each network decreases slowly at about the same rate for each network. The delay per stage increases slowly for I1 and I2 and stays roughly constant for C1 and C2.

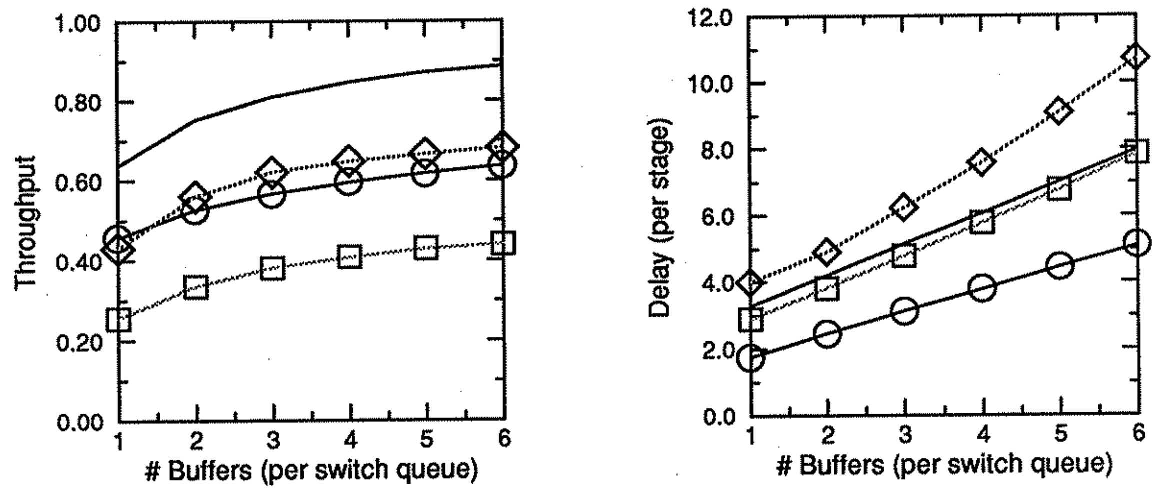




GRAPHS 1-prob_load - Series 1 data for variation in offered load.

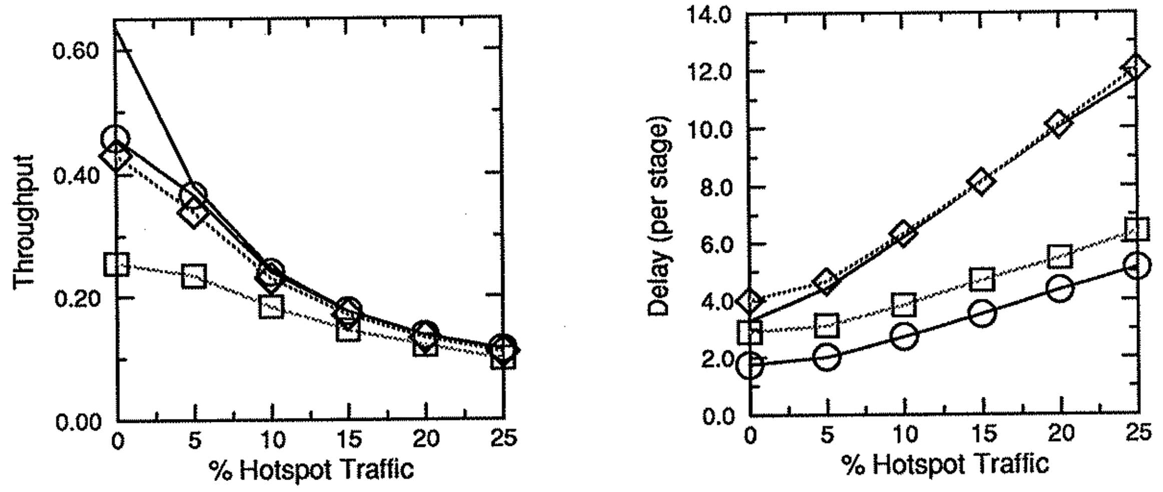
Probabilistic Load - Graphs 1-prob_load show throughput and delay data for specified offered loads. Instead of generating a new operation whenever its output buffer is empty, a PE generates a new operation each cycle with probability *r*. As *r* increases, throughput for each network increases up to the network's saturation point. Delay for each network increases sharply near the saturation point.When load is generated independently of the state of a PE's output buffer, source queueing can account for a significant portion of total delay. Therefore, in this part of series 1 source queueing is included in delay. Delay is measured from the time the operation is generated until it leaves the MM-SIU. Each of the conventional networks can handle a higher load with less delay than the corresponding isotach network. The request rate at which C2 saturates is the highest, 0.65; and I1 the lowest, 0.25. C1 and I2 saturate at about the same request rate, 0.45 and 0.43, respectively.

Switch Input Queue Size - Graphs 1-switch_queues show the effect of increases in the size of the switch input queues. As the number of buffers in each switch input queue increases, throughput improves for all networks, but delay worsens. The graphs indicate that performance is best when the switch input queues are small, able to queue only 1 or 2 operations at most. The result applies only to saturation loads in which source queueing is not a factor.



GRAPHS 1-switch_queues - Series 1 data for variation of the number of buffers per switch queue.

Hot-Spot Traffic - Graphs 1-hot_spot give performance results for the hot-spot traffic model. The hot-spot traffic model [PhN85] is a simple variation on the uniform traffic model in which one variable, the hot-spot, is accessed more often than the other variables. The hot-spot is chosen randomly such that for each trial, each global variable may be the hot-spot with equal probability. The probability that a given operation accesses the hot-spot is the hotspot probability *r* plus *(1-r)* divided by the total number of variables. As the probability of generating operations accessing the hot-spot increases, throughput decreases and the differences in throughput among the networks narrows. The delay of the z-switch networks, C2 and I2, increases at a significantly faster rate than the low-throughput networks. Again, the result applies only to saturation loads in which source queueing is not a factor.



GRAPHS 1-hot_spot - Series 1 data for various hotspot loads.

Warm-spot Traffic - Table1 gives results from a warm-spot traffic model. A warm-spot traffic model has several "warm" variables instead of a single hot variable. We defined the warm-spot traffic model in an attempt to identify a model that captures contention for shared variables more realistically than either the uniform or the hot-spot traffic models. The warm-spot traffic model is based on the 80/20 rule (see e.g. [Knu73]), a rule of thumb widely used in describing the pattern of accesses to a pool of shared objects. The 80/20 rule says that 80% of the accesses are to 20% of the variables. The rule is applied recursively, i.e., 80% of the initial 80% of accesses are to 20% of the initial 20% of the accesses, and so on until the number of objects in the set of most accessed objects reaches one. The warm-traffic model we use is a modified version of the standard 80/20 rule that supports ratios other than 80/20. For simplicity, it truncates the recursion at 4 levels and it assumes that the probability of assessing the most frequently accessed variables at level *i* is the probability of accessing the most frequently accessed variables at level *i-1* cubed. The effect of this modification is to lessen the contention for the most frequently accessed variables. Table 2 shows performance under three different warm-spot traffic models: light contention (60/30 rule); medium contention (70/20); and heavy contention (80/10). The table indicates that increased contention affects the relative performance of the networks in the warm-traffic model in the same way as in the hot-spot model: the differences in throughput narrow and delay in the z-switch networks C2 and I2 grows faster than delay in C1 and I1.

Access Throughput	Low Sharing	Medium Sharing	Heavy Sharing
C1	0.442	0.396	0.156
C2	0.585	0.487	0.163
I1	0.247	0.233	0.124
I2	0.413	0.362	0.147
Normalized Access Delay	**Low Sharing**	**Medium Sharing**	**Heavy Sharing**
C1	1.805	1.964	3.952
C2	3.305	3.970	8.957
I1	3.001	3.164	5.396
I2	4.135	4.546	9.529

TABLE 1. Series 1 Warm-spot data.

The results from Series 1 show that each conventional network outperforms the corresponding isotach networks for workloads with no atomicity, data dependence, or other sequencing constraints. Each of the remaining series compares the performance of the networks under a workload that includes some combination of atomicity and sequencing constraints. In addition to including constraints on execution order, series 2-6 differ from series 1 in other ways:

1. Processes - Each process is independent, executing its own program on its own PE, and each process's program consists of a sequence of atomic actions, each containing one or more operations on shared variables.
2. Reverse network - Each network simulation includes traffic in both the forward (PE->MM) and reverse (MM->PE) directions. All of the reverse networks are conventional networks. I2 uses C2 as the reverse net and the other networks use C1. In each cycle each MM (PE) can both receive one operation (response) from the network and issue one response (operation).
3. Throughput - The throughput reported is the atomic action throughput instead of the access throughput. Unless otherwise stated, the throughput reported is the average number of atomic actions completed per cycle divided by the number of MMs.
4. Delay - The delay reported is the atomic action delay. Unless otherwise stated, delay is the average number of cycles from the time an atomic action is generated until it is completed. Note that delay is not normalized for the number of network stages as it is in series 1.
5. Networks - Results are shown only for C1, I1, and I2. We simulated all four networks, including C2. Although C2 outperforms all the other networks in series 1, it performs more poorly than C1 in the remaining series. The results show that at every data point, C1 provides more throughput with less delay than C2. The reason for C2's poor performance relative to C1 is that in the later series C2 retains its high latency but can't take advantage of its high throughput. Both conventional networks are under-utilized in series 2-6 because higher level synchronization constraints prevent processes from issuing operations fast enough to saturate the network. For simplicity, we omit further discussion of C2.

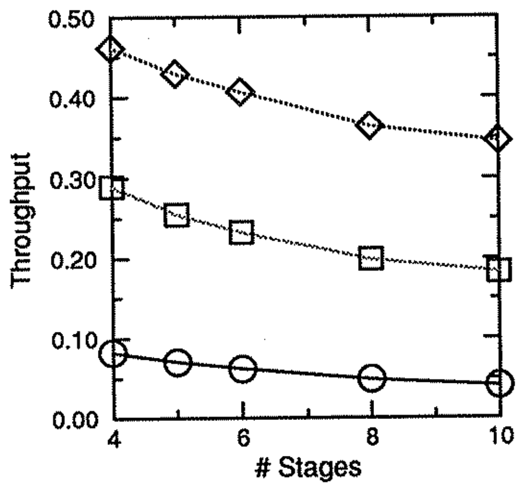
4.2 Sequential Consistency Only Data

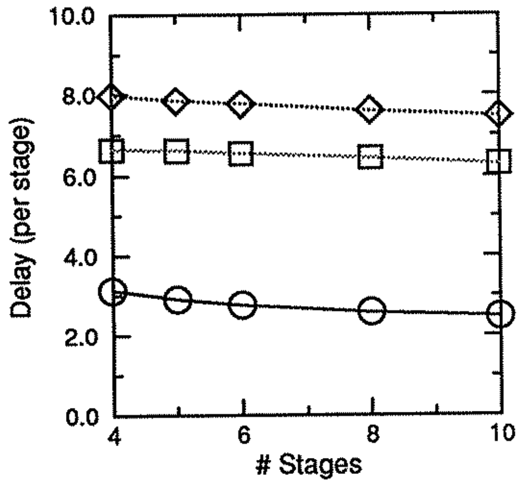
Series 2 compares the networks under a workload model requiring sequential consistency. In this series, all atomic actions are singletons (size 1) and no data dependencies constrain the issuing of operations, but each process's operations must appear to be executed in order. In I1 and I2, operations can be pipelined without risk of violating sequential consistency, whereas C1 must enforce sequential consistency by limiting each PE to one outstanding operation at a time. In C1, each PE repeatedly issues an operation, waits for a response from memory, and then issues its next operation (i. e., the simulation parameter "aa_cap" equals 1). In I1 and I2, each PE issues a new operation whenever its output buffer is empty (aa_cap = unlimited). Note that the isotach network simulations in series 1 and 2 are the same except for the inclusion of the reverse networks and the difference in the way results are reported. Delay in series 2 is the number of cycles between the time an operation is placed in the PE's output buffer until the time the PE receives the response to the operation.

Base Case - Throughput for C1 in the base case is only about 15% of its series 1 throughput, reflecting the high cost of enforcing sequential consistency in a conventional network. For the isotach networks the throughput in series 1 and 2 is the same. Because they can pipeline operations, the isotach networks outperform C1 in terms of throughput but the delay per operation is longer than C1's delay. In the base case I1 can execute 3.6 times as many operations as C1, but each operation takes 2.3 times as long. I2 can execute 6 times as many operations, but each takes 2.7 times as long. Delay for all networks is higher than in series 1 because delay in series 2 includes delay due to the reverse network and non-network delay.

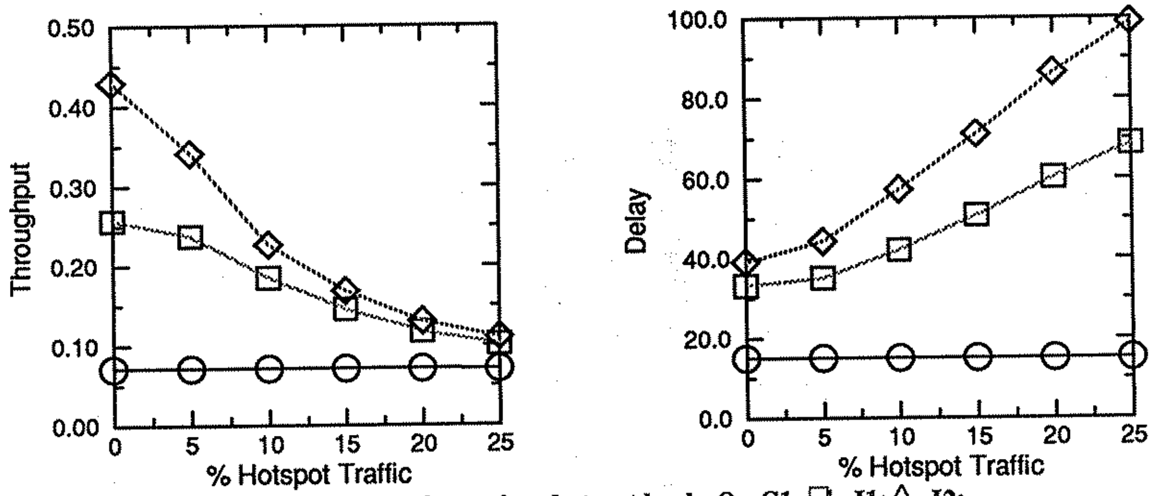
In each part of the series, one parameter is varied while each of the remaining parameters remains at its base case setting.

Scalability - Graphs 2-scalability show the effect of varying the network size from its base case setting of 5. As network size increases from 4 to 10 stages, throughput slowly decreases and delay per stage remains roughly constant. Delay per stage in I1 and I2 is relatively constant as the number of network stages increases because the delay due to the forward network is only one component of the delay reported in series 2. The other components, the reverse network and non-network delay either remain constant on a per stage basis or decrease. Delay per stage due to the reverse network is roughly constant since the reverse networks are conventional networks. Non-network delay declines on a per stage basis as the delay is amortized over more stages. Throughput decreases for each network at about the same rate but for different reasons. Throughput per MM decreases in the isotach networks because in isotach networks throughput per MM decreases as the number of stages increases. Throughput per MM decreases in the conventional networks because a PE can have only one outstanding operation at a time and delay for each operation increases as the number of stages increases.





GRAPHS 2-scalability - Series 2 data for variation of network size. O - C1; ❑ - I1; ◊ - I2;



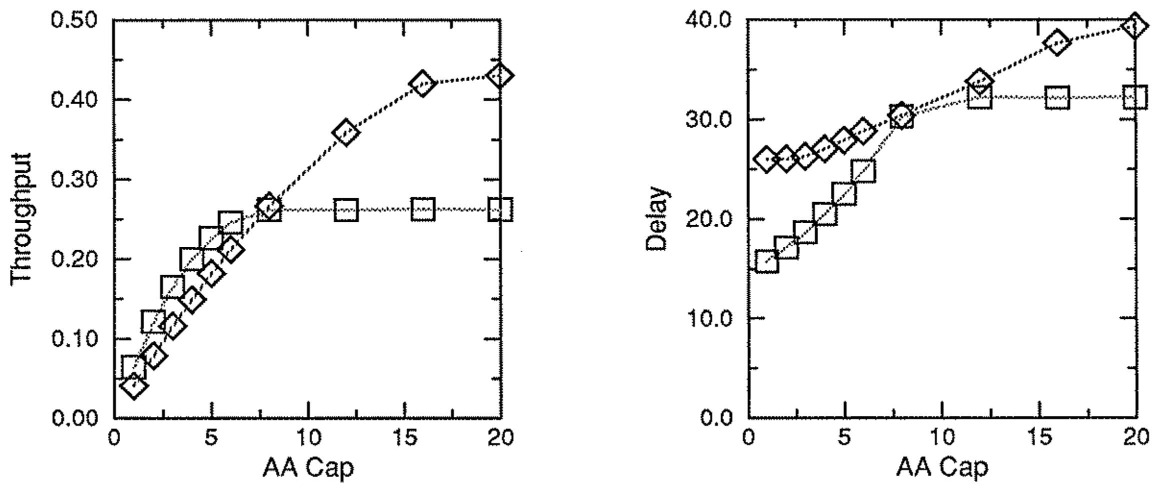
GRAPHS 2-hot_spot - Series 2 data for various hotspot loads. O - C1; ❑ - I1; ◊ - I2;

Hot-spot and Warm-spot Traffic - Graphs 2-hot_spot and Table 2 show network performance for the hot-spot and warm-spot traffic models, respectively. As in series 1, networks I1 and I2 show significant decreases in throughput and increases in delay as contention increases. Network C1 has low throughput and low delay. Over the range of traffic model parameters tested, C1 is unaffected by increases in the probability of hot-spot and warm-spot traffic. Throughput and delay are unaffected in C1 because the network is operating at only about 28% capacity due to the restriction on pipelining.

Access Throughput	Low Sharing	Medium Sharing	Heavy Sharing
C1	0.071	0.071	0.071
I1	0.249	0.229	0.128
I2	0.408	0.367	0.254
Access Delay	**Low Sharing**	**Medium Sharing**	**Heavy Sharing**
C1	15.000	15.000	15.002
I1	34.032	36.038	56.861
I2	40.344	42.887	76.587

TABLE 2. Series 2 warmspot data.

Data Dependencies - Graphs 2-aa_cap show the effect of changes in the data dependence distance on system performance. Data dependencies diminish throughput of isotach networks by an amount that depends on the distance between data dependent operations. Until now, we have assumed data dependencies either do not exist or are between operations that are so far apart that the data dependencies do not effect pipelining. In this part of series 2 we relax this assumption. With a data dependence distance of 1 (modeled by imposing an aa_cap of 1), the throughput of the isotach networks is lower and the delay higher than that of C1, as can be predicted from the results of series 1. Beginning at a data dependence distance of 2 (aa_cap of 2), the throughput of the isotach systems is higher than C1's although the delay also continues to be higher. C1 requires an aa_cap of 1 in order to maintain sequential consistency and so is unaffected by the data dependence distance. I2 requires a dependence distance of about 16 to take full advantage of its throughput. For I1, a dependence distance of about 5 is sufficient. I2 performs less well than I1 at small dependence distances because I2 cannot take advantage of its higher throughput but is hurt by the higher response turnaround time due to its higher latency.



GRAPHS 2-aa_cap - Series 2 data for variation of data-dependence distance. ❑ - I1; ◊ - I2;

4.3 Atomicity Only Data

Series 3 compares the performance of isotach and conventional systems under a workload requiring atomic access to multiple shared variables. The atomic actions are assumed to be *flat*, i.e., no data dependencies exist among operations from the same atomic action, and independent, i.e., no data dependencies among different atomic actions inhibit pipelining. The atomic actions must appear to be executed as an indivisible step, but may appear to be executed in any order, i.e., the workload does not require enforcement of sequential consistency. Although sequential consistency is not required, the isotach networks guarantee sequential consistency by their design.

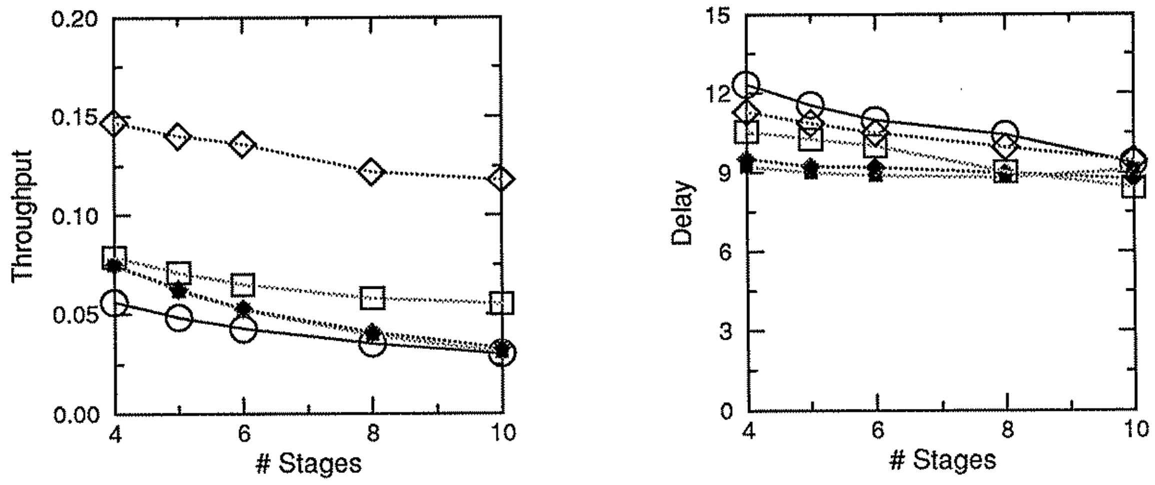
Techniques for enforcing atomicity in isotach systems were described in section 2. Isotach networks enforce atomicity by issuing all the operations in an atomic action in the same isochron. The conventional systems we simulate enforce atomicity using 2PL. A process does not release any lock acquired in an atomic action until it acquires all locks for the atomic action. To avoid deadlock, each process acquires the locks it needs for each atomic action in a predetermined linear order. Since execution need not be sequentially consistent, a process may execute more than one atomic action concurrently.

The algorithm for lock acquisition and release simulated is more efficient than that used in most systems. Instead of spinning on a lock, processes queue lock requests at memory. The algorithm distinguishes read locks from write locks and only the latter are exclusive. Several processes can concurrently hold a read lock on the same variable. Instead of sending a lock request for an operation, a process sends the operation itself. Each operation implicitly carries a request for a lock of the type indicated by the operation. When it receives an operation, the MM enqueues it. If no conflicting operation is enqueued ahead of it, the operation is executed and a response returned to the source PE. A PE knows it has acquired a lock when it receives the response. Eliminating explicit lock requesting and granting messages reduces traffic and eliminates the roundtrip delay from memory to the process and back when a lock is granted. When the PE has acquired all the locks for the atomic action, it sends a lock release for each lock it holds.

Since execution need not be sequentially consistent, locks are not obtained for operations from singleton (size 1) atomic actions. The MM's execute operations from singleton atomic actions as soon as they are received even if another process holds an exclusive lock on the variable accessed. In the base case, with an atomic action mean size of 3, about 22% of atomic actions are of size 1.

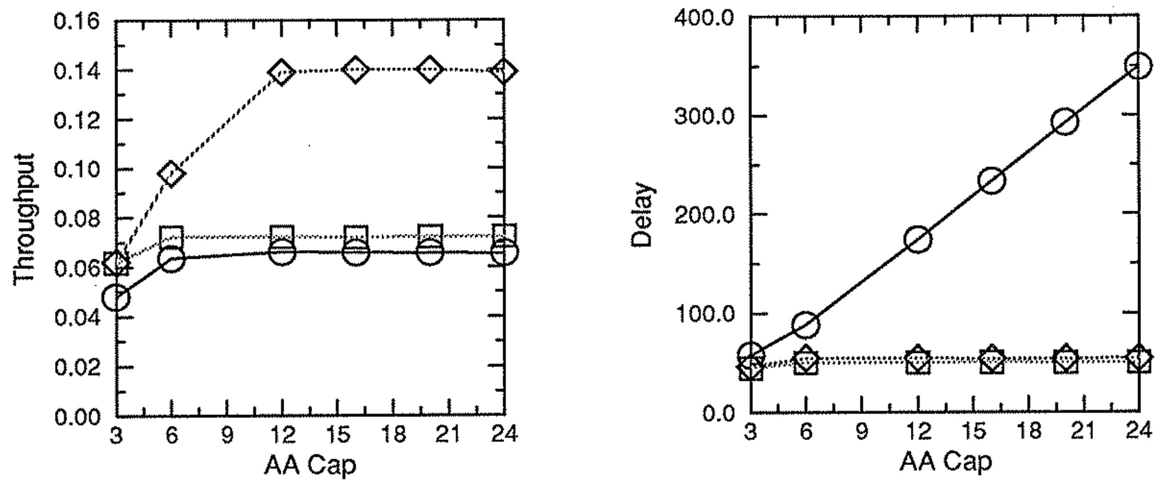
Delay in series 3 is the number of cycles from the time the atomic action is generated until the last response is received. Note that for conventional networks, the time required to release locks is not included in delay. A process generates a new atomic action whenever its output buffer is empty and the number of atomic actions it has started but not completed is fewer than a given number, aa_cap, supplied as a parameter. Initially we set aa_cap = unlimited for both isotach and conventional systems to allow unrestricted pipelining. We found however that allowing unlimited production of atomic actions in the conventional networks is counterproductive due to contention for locks. Latency rises steeply with no gain in throughput. This effect does not occur in the isotach systems, where locking is not necessary. We present data for C1 with aa_cap = 3. For I1 and I2, we show results for aa_cap = 3 and for aa_cap = unlimited.

Base Case - When each network is subject to an aa_cap of 3, throughput in the isotach networks is slightly higher in the base case and delay slightly lower than in C1. The performance of the uncapped isotach networks is better than the capped networks. When I2 is uncapped, its throughput is almost 3 times that of C1 and its delay remains lower than C1's.



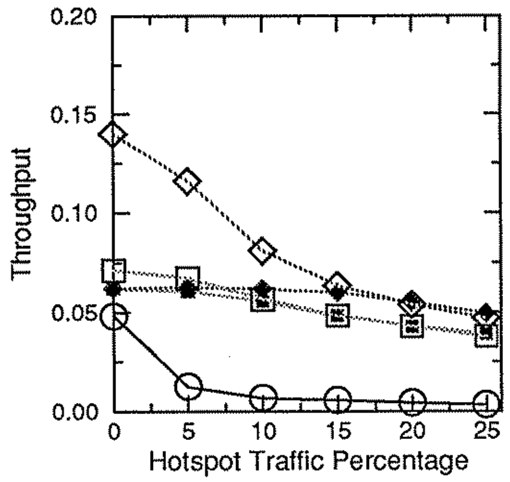
GRAPHS 3-scalability - Series 3 data for variation of network size. O - C1; ❑ - I1; ◊ - I2; ■ - I1 (AA Cap = 3); ◆ - I2 (AA Cap = 3);

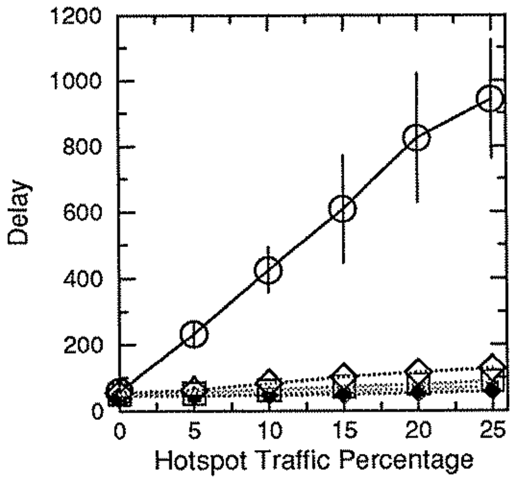
Scalability - Graphs 3-scalability show the effect of varying the number of network stages. With aa_cap = 3 for all networks, the graphs indicate that the performance of I1 and I2 becomes worse than that of C1 for large systems (stages > 9), but that the uncapped isotach networks continue to outperform C1. The graphs show that as the number of stages increases, throughput and delay gradually decrease. The delay reported in this part of series 3 is the delay normalized for the number of stages. The decrease in throughput and delay is attributable to the factors discussed under the scalability part of series 2. The graphs show an anomalous result for delay in large isotach networks: delay becomes worse when a cap is placed on the number of active atomic actions.



GRAPHS 3-AA_cap - Series 3 data for variation of network size. O - C1; ❑ - I1; ◊ - I2;

Atomic Action Cap - Graphs 3-AA_cap show the effect of varying the cap on the number of atomic actions a PE can have active at any one time. The graphs show C1 performs very poorly with a large aa_cap. C1 does not benefit from increasing the aa_cap because increasing the aa_cap also increases contention for locks. This contention is reflected in the delay for C1, which rises steeply as the aa_cap increases. Throughput for C1 levels off at about 6. For I1 and I2, throughput and delay rise as the aa_cap increases until the network is saturated (at about 6 for I1 and about 12 for I2) and thereafter remain constant.





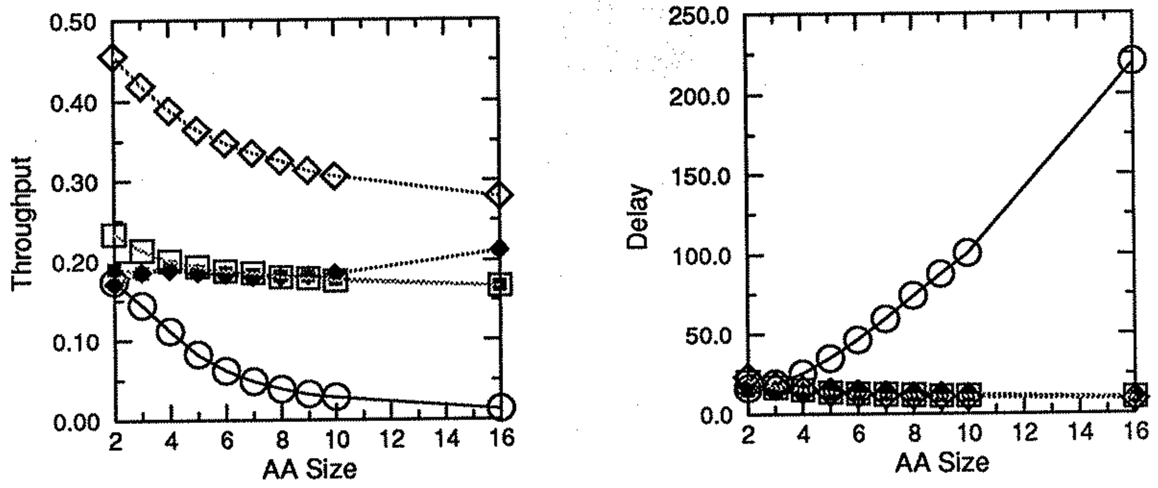
GRAPHS 3-hot_spot - Series 3 data for variation of hotspot load.
O - C1; ❑ - I1; ◊ - I2; ■ - I1 (AA Cap = 3); ◆ - I2 (AA Cap = 3);

Hot-spot and Warm-spot Traffic - Graphs 3-hot_spot and Table 3 show network performance for the hot-spot and warm-spot traffic models, respectively. The results show that C1 performs increasingly poorly relative to I1 and I2 as the percentage of traffic going to hot and warm variables increases. C1 is more adversely affected by nonuniform access patterns because the use of locking as the means for enforcing atomicity magnifies the effect of contention. The throughput graph shows that until the percentage of hotspot traffic is about 15%, throughput for I1 and I2 is lower when aa_cap = 3 than when aa_cap = unlimited, indicating that the cap itself is the limiting factor. Thereafter the capped and uncapped networks have similar throughput, indicating that the hotspot traffic has become the limiting factor. We attribute the large confidence intervals exhibited by C1 to the fact that placement of the hot-spot variable changes with each trial, while the order in which locks are acquired remains the same.

Atomic Action Throughput	Low Sharing	Medium Sharing	Heavy Sharing
C1 (AA Cap=3)	0.0269 (.0267 - .0271)	0.0127 (.0125 - .0129)	0.0035 (.0034 - .0036)
I1 (AA Cap=3)	0.0611 (.0606 - .0616)	0.0599 (.0595 - .0603)	0.0415 (.0405 - .0427)
I2 (AA Cap=3)	0.0625 (.0615 - .0635)	0.0622 (.0615 - .0629)	0.0561 (.0539 - .0583)
I1 (No AA Cap)	0.0693 (.0682 - .0705)	0.0661 (.0649 - .0673)	0.0390 (.0371 - .0409)
I2 (No AA Cap)	0.1359 (.1341 - .1376)	0.1238 (.1214 - .1263)	0.0565 (.0527 - .0602)
Atomic Action Delay	**Low Sharing**	**Medium Sharing**	**Heavy Sharing**
C1 (AA Cap=3)	108.393 (107.537 - 109.250)	232.193 (228.971 - 235.414)	849.973 (817.718 - 882.228)
I1 (AA Cap=3)	45.751 (45.460 - 46.042)	46.711 (46.252 - 47.170)	67.186 (65.508 - 68.864)
I2 (AA Cap=3)	46.312 (45.635 - 46.989)	46.558 (46.004 - 47.111)	52.123 (49.918 - 54.328)
I1 (No AA Cap)	52.291 (51.664 - 52.918)	54.602 (51.664 - 52.918)	86.775 (82.979 - 90.570)
I2 (No AA Cap)	55.702 (55.156 - 56.249)	60.092 (59.208 - 60.976)	112.505 (105.842 - 119.168)

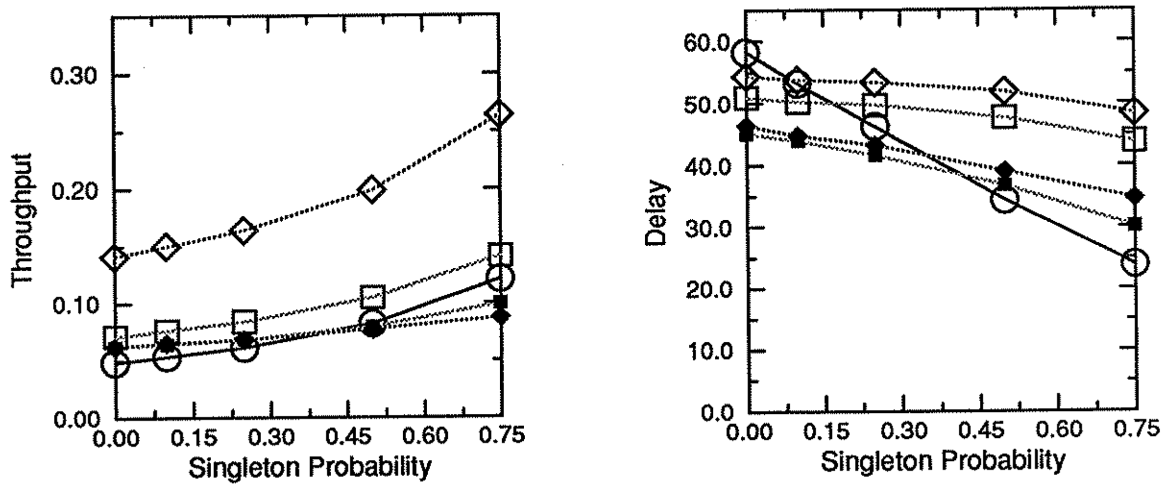
TABLE 3. Series 3 warm-spot data. 99% confidence intervals are shown in parentheses.

Atomic Action Size - Graphs 3-AA_size show the effect of varying the average size, aa_mean, of atomic actions. Throughput as reported in this part of series 3 is normalized for atomic action size, i.e., reported throughput is atomic action throughput times the average number of operations per atomic action. Thus reported throughput is the average number of operations arriving at each MM. Delay is also normalized. Delay is the atomic action delay divided by the average number of operations per atomic action. The results show C1 performs poorly for large atomic actions. As the atomic action mean size increases, C1's throughput drops and its delay rises steeply. The isotach networks, both capped and uncapped, outperform C1. The drop in throughput is more gradual and, instead of increasing, normalized delay actually decreases. The decrease is attributable to the fact that operations in the same atomic action are delivered and executed concurrently in an isotach system, so delay per operation declines as the number of operations per atomic action increases. For large atomic actions, the isotach systems perform markedly better than the conventional systems. When the average atomic action size is 16, the delay in the conventional systems is greater than the delay in the isotach systems by a factor of about 22 and the throughput is less than the throughput in the isotach systems by a factor of about 12 (I1) to 16 (I2). The steep rise in delay in the conventional systems is attributable to the locking protocol. A process waiting to acquire a lock retains the locks it has already acquired for operations in the same atomic action. As atomic actions grow larger, not only does the number of operations contending for access grow, but also the length of time each lock is held.



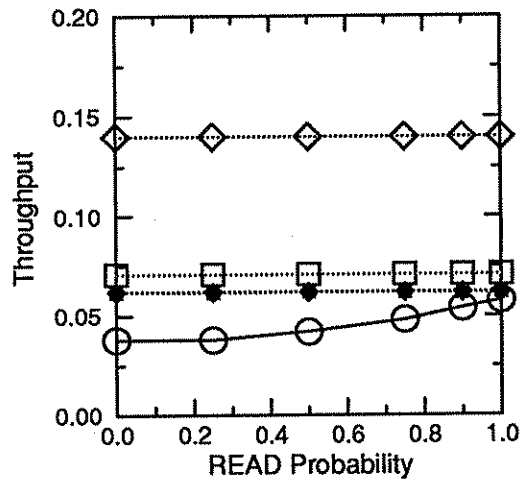
GRAPHS 3-AA_size - Series 3 data for variation of atomic action size.
O - C1; ❑ - I1; ◊ - I2; ■ - I1 (AA Cap = 3); ◆ - I2 (AA Cap = 3);

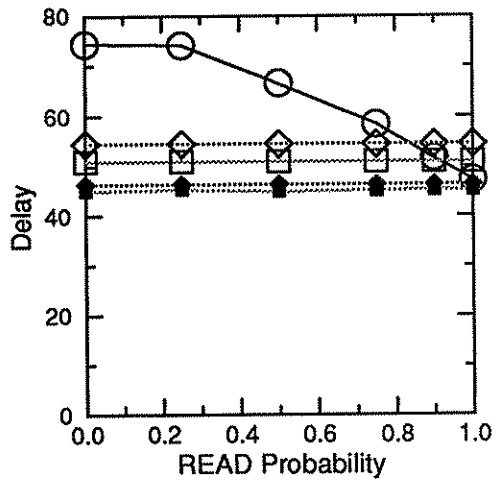
The capped versions of I1 and I2 have lower throughput than the uncapped versions when the atomic actions are too small to saturate the network.



GRAPHS 3-force_sing - Series 3 data for variation of singleton probability.
O - C1; ❑ - I1; ◊ - I2; ■ - I1 (AA Cap = 3); ◆ - I2 (AA Cap = 3);

Forced Singletons - Graphs 3-force_sing show the effect of varying the probability of forced singletons. Recall that the actual percentage of singleton atomic actions, i.e., atomic actions of size 1, is larger than the forced-singleton probability because some singleton atomic actions may be drawn from the exponential distribution of atomic action sizes. With an atomic action mean of 3, a forced singleton probability of 50% corresponds to a total probability of singleton atomic actions of about 61%. The results show that network C1 performs well when the percentage of singletons is high. Since C1 does not enforce sequential consistency in series 3, or acquire any locks for singleton atomic actions, as the percentage of singletons increases, its performance becomes increasingly similar to its performance in series 1. The isotach networks are less sensitive to the percentage of singleton atomic actions. As the singleton percentage increases, their performance improves, but the rate of improvement in delay is less than that of C1 and, except for the uncapped version of I2, the rate of throughput is about the same as C1's. As a result, the isotach networks, with the exception of the uncapped version of I2, perform less well than C1 when the percentage of singletons is high.





GRAPHS 3-read_prob - Series 3 data for variation of READ probability.
O - C1; ❑ - I1; ◊ - I2; ■ - I1 (AA Cap = 3); ◆ - I2 (AA Cap = 3);

Read/Write Probability - Graphs 3-read_prob show the effect of varying the percentage of accesses that are READ operations from the base case setting of 75%. The conventional networks perform better as the percentage of read accesses increases because the read-locks required for READ's can be shared, whereas WRITE's require exclusive locks. The isotach networks are unaffected by the percentage of reads.

Series 3 shows that for a workload model that requires atomicity but not sequential consistency, the isotach networks outperform the conventional networks. Even with the handicap of also insuring sequential consistency, the isotach networks are able to perform better than the conventional networks.

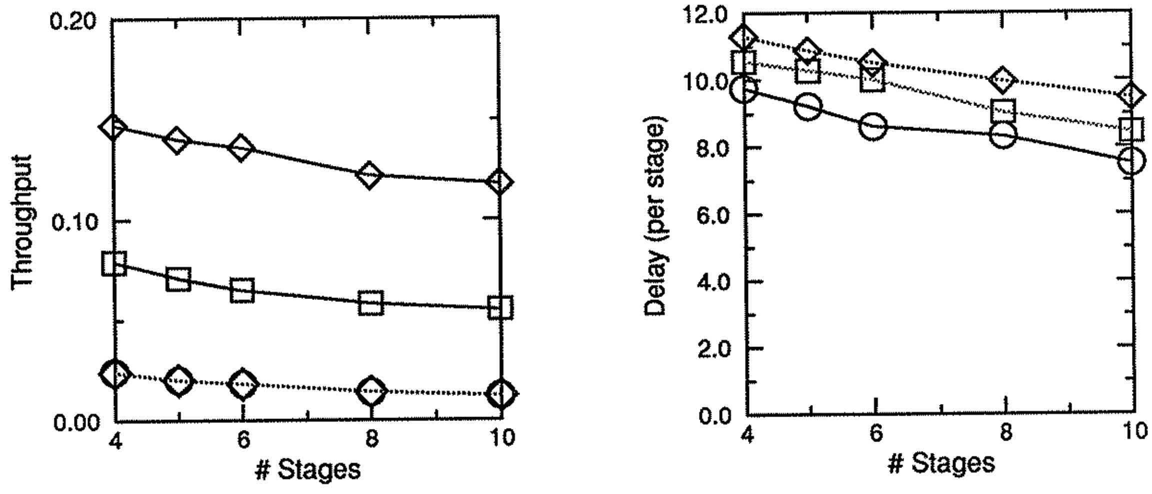
4.4 Flat Atomic Action Data

Series 4 compares the performance of the networks under a workload which requires both atomicity and sequential consistency. As in series 3, the atomic actions are assumed to be independent and flat, but in this series, the atomic actions must not only be executed atomically but also must appear to be executed in order.

The conventional networks ensure atomicity through the locking protocol discussed above. Sequential consistency is enforced by limiting the number of outstanding atomic actions per PE to one (aa_cap = 1) and by requiring processes to obtain locks for singleton atomic actions. In the isotach systems, by contrast, a PE may have any number of active atomic actions and it need not obtain locks.

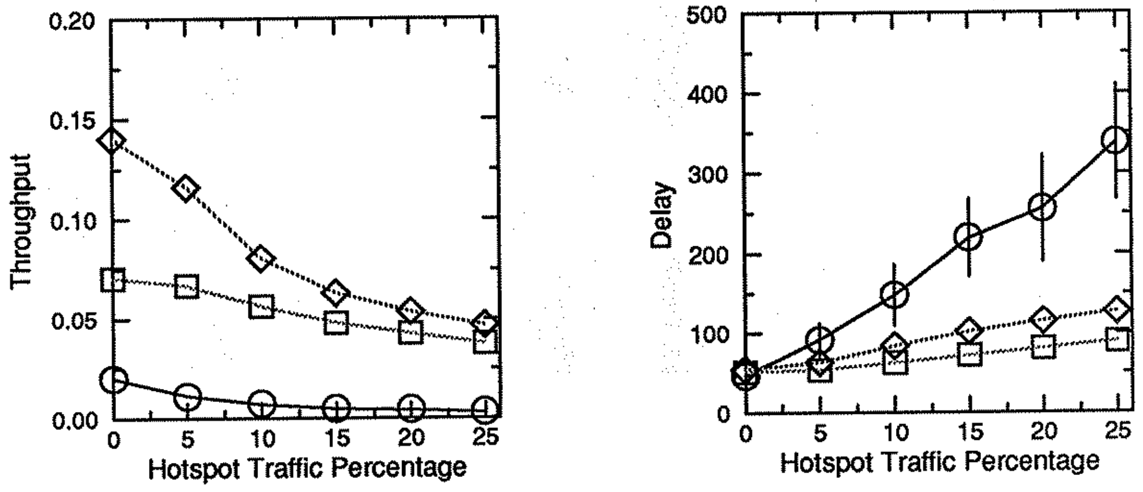
This series is very similar to series 3. Throughput and delay are measured and reported in the same way as in series 3. In terms of the simulation parameters, the only difference is in the setting of the aa_cap. For the conventional networks, aa_cap is reduced from 3 to 1. For the isotach networks, only the uncapped versions are relevant in this series. The results reported for I1 and I2 are identical to those reported in series 3 as the results for the uncapped versions of I1 and I2. Our report for series 4 is limited to the differences between series 3 and series 4.

Base Case - C1 has lower throughput and lower delay in series 4 than in series 3 because the cap on the number of atomic actions is lower. The lower cap reduces delay by reducing the load on the network and by reducing contention for locks. As a result, C1's delay is better in relation to I1 and I2 in this series than in series 3 and its throughput is worse. C1's per stage delay in the base case is slightly better than that for I1 and I2 (9.2 for C1 as opposed to 10.3 for I1 and 10.9 for I2), but its throughput is much worse: in the case of I1, by a factor of 3.5 and in the case of I2 by a factor of 7.



GRAPHS 4-scalability - Series 4 data for variation of network size. O - C1; ❑ - I1; ◊ - I2;

Scalability (Graphs 4-scalability) - As in series 3, throughput and delay per stage drop off gradually in both the isotach and conventional systems as the number of stages increases from 4 to 10.



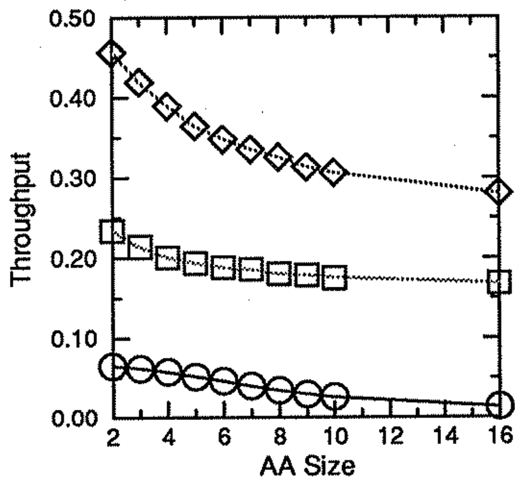
GRAPHS 4-hot_spot - Series 4 data for variation of hotspot load. O - C1; ❑ - I1; ◊ - I2;

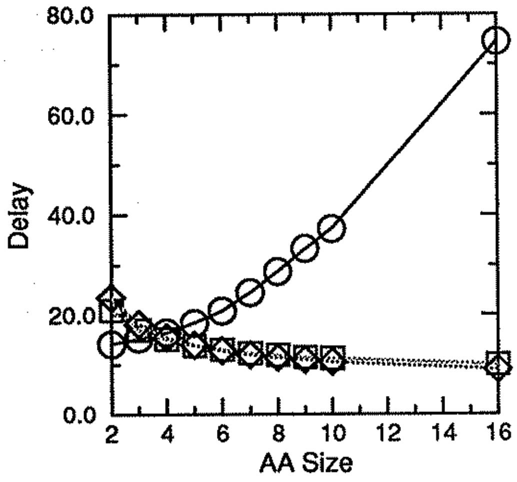
Hot-spot and Warm-spot Traffic (Graphs 4-hot_spot and Table 4) - As in series 3, the results show that the advantage of isotach networks over conventional networks grows as the probability of hot spot and warm spot traffic increases.

Atomic Action Throughput	Low Sharing	Medium Sharing	Heavy Sharing
C1	0.0175 (.0175 - .0176)	0.0115 (.0115 - .0116)	0.0032 (.0031 - .0033)
I1	0.0693 (.0682 - .0705)	0.0661 (.0649 - .0673)	0.0390 (.0371 - .0409)
I2	0.1359 (.1341 - .1376)	0.1238 (.1214 - .1263)	0.0565 (.0527 - .0602)
Atomic Action Delay	**Low Sharing**	**Medium Sharing**	**Heavy Sharing**
C1	54.086 (53.907 - 54.264)	83.755 (83.185 - 84.325)	308.015 (296.555 - 319.475)
I1	52.291 (59.208 - 60.976)	54.602 (53.784 - 55.421)	86.775 (82.979 - 90.570)
I2	55.702 (55.156 - 56.249)	60.092 (59.208 - 60.976)	112.505 (59.208 - 60.976

TABLE 4. Series 4 warm-spot data. 99% confidence intervals are shown in parentheses.

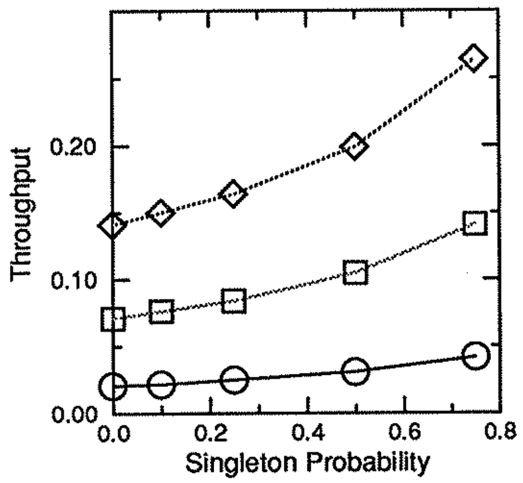
Atomic Action Size (Graphs 4-AA_size) - The delay in C1 is lower than is series 3, so the difference in delay between the conventional and isotach systems is less in this series. The delay in C1 is higher than that of I1 and I2 by a factor of about 7.5 instead of 22 for aa_size = 16.

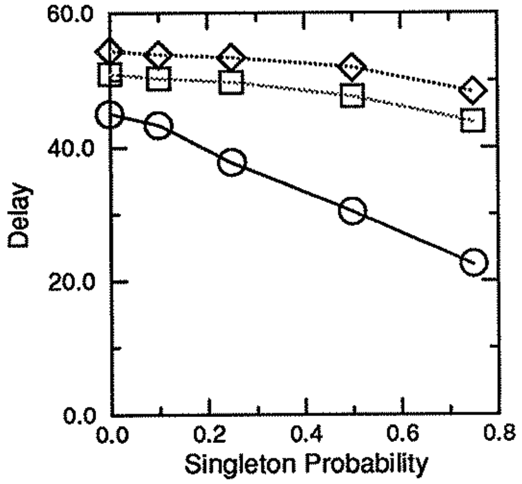




GRAPHS 4-AA_size - Series 4 data for variation of atomic action size. O - C1; ❑ - I1; ◊ - I2;

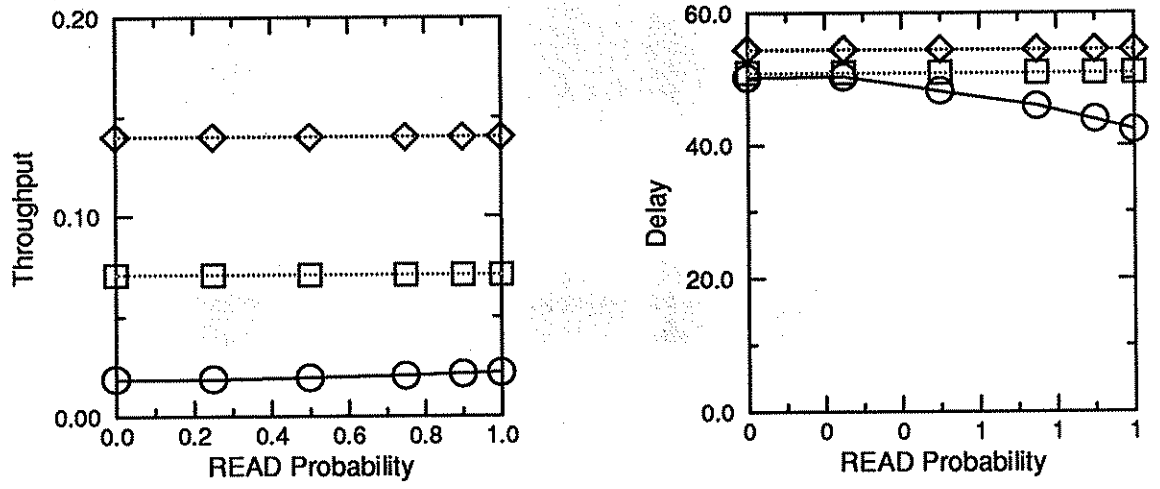
Forced Singletons. (Graphs 4-force_sing) - C1 does not show as great an improvement as the percentage of singletons increases in this series as it does in series 3. The reason C1 derives less benefit from singletons in series 4 is that singletons are subject to more restrictions. In series 4, singletons cannot be pipelined by C1 and must respect locks in the same way as ordinary accesses.





GRAPHS 4-force_sing - Series 4 data for variation of the singleton probability. O - C1; ❑ - I1; ◊ - I2;

Read/Write Probability (Graphs 4-read_prob) - C1 is less sensitive to the proportion of reads and writes than in series 3. Reducing the aa_cap from 3 to 1 means fewer operations are contending for access to the same variable. In the base case of series 4 (25% WRITES) the percentage of lock requests that are granted immediately is 97 as opposed to 91% in series 3. Concurrently active operations access the same variable so infrequently in this series that increasing the percentage of writes has little effect on performance.



GRAPHS 4-read_prob - Series 4 data for variation of READ probability. O - C1; ❑ - I1; ◊ - I2;

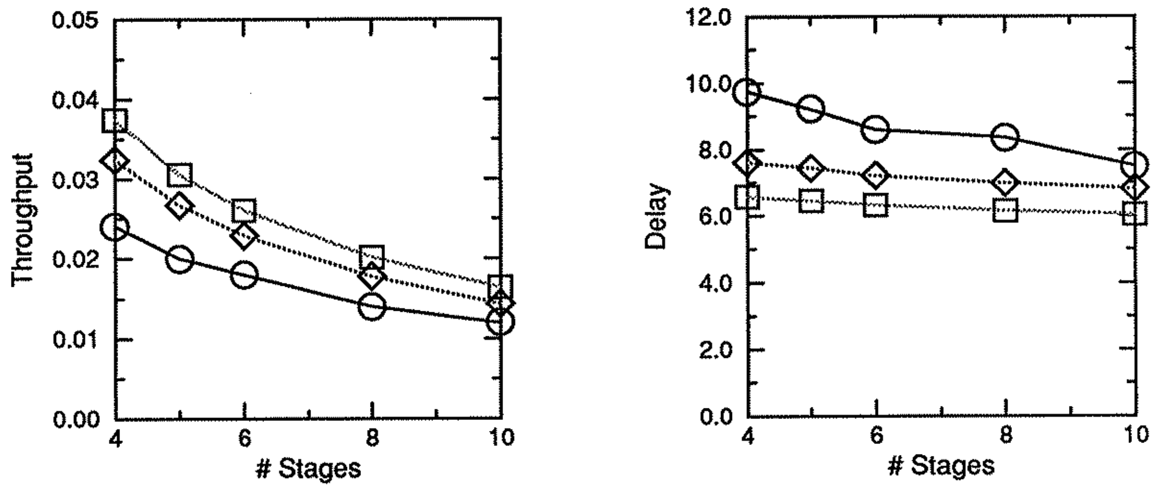
This series shows that the isotach systems perform very well and the conventional systems very poorly when each process's workload consists of a sequence of flat, independent atomic actions. The conventional systems cannot take full advantage of the greater raw power of their networks because the rate at which operations can be issued is severely limited by the locking protocol and the restriction on pipelining. The performance of the isotach systems, by contrast, is limited only by the performance of the raw power of the networks. Though the raw power of the networks is somewhat lower than that of the conventional networks, the synchronization support the isotach networks provide allows the isotach systems to outperform the conventional systems by a wide margin.

4.5 Data Dependency Data

This series compares the performance of conventional and isotach systems for a workload model in which data dependencies exist both within and among atomic actions. The existence of data dependencies among atomic actions means that a process cannot issue an atomic action until the proceeding atomic action is complete. To simulate this type of data dependence, we set aa_cap =1 for all networks.

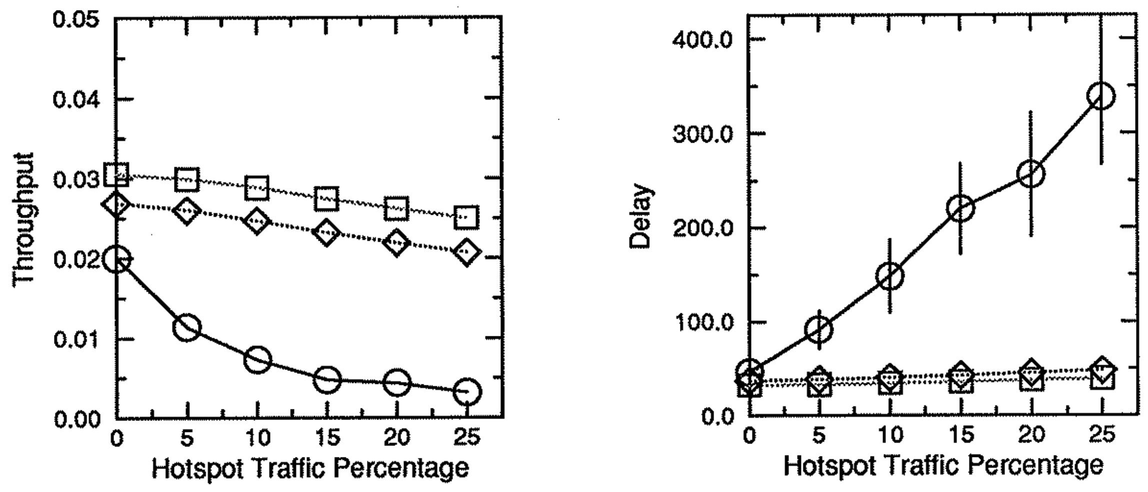
The data dependencies within atomic actions are assumed to be of a simple type: each WRITE depends on all the READ's in the same atomic action. As a consequence, a process cannot issue a WRITE until it has received responses to all the reads in the same atomic action. The introduction of data dependencies within atomic actions should make C1 very slightly less efficient than in series 4, but we assume for simplicity that C1's performance remains the same. The data for C1 in this series is copied from the data from series 4. For an isotach system the impact of data dependencies is much more significant since they prevent the isotach system from taking advantage of its ability to pipeline while maintaining sequential consistency. In the case of I1 and I2, the data dependencies within atomic actions are modelled by using the scheduling isochron technique in section 2 (access sequences and split operations) for every atomic action containing a WRITE.

Base Case - Both the throughput and delay are lower for the isotach systems than in series 4. Delay is lower because the networks are more lightly loaded. As noted above, the data for C1 is copied from series 4. I1 and I2 perform only slightly better than C1 in the base case. Throughput for C1 is lower (.020 for C1 instead of about .030 and .027 for I1 and I2) and delay per stage is higher (9.2 for C1 instead of 6.6 and 7.5 for I1 and I2).



GRAPHS 5-scalability - Series 5 data for variation of network size. O - C1; ❑ - I1; ◊ - I2;

Scalability (Graphs 5-scalability) - The throughput of C1 and I2 diminish gradually and at about the same rate. I1 scales slightly less well than I2 under this series's workload. The throughput of I1 drops faster than I2's and, instead of diminishing, the delay per stage for I1 increases very slightly.



GRAPHS 5-hot_spot - Series 5 data for variation of hotspot load. O - C1; ❑ - I1; ◊ - I2;

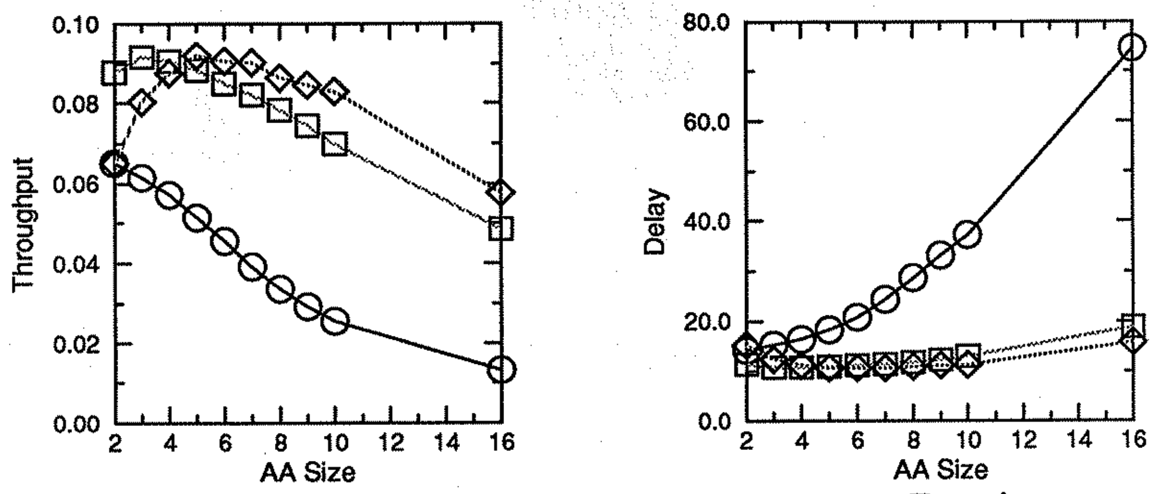
Hot-spot and Warm-spot Traffic (Graphs 5-hot_spot and Table 5) - Although the throughput for the isotach networks is much lower than in series 4, it remains higher than the throughput of C1. Delay for the isotach networks improves significantly over the delay in series 4 and is much lower than the delay for C1. When the probability of accessing the hot-spot is 10%, the throughput for I1 and I2 is about 3.5 times higher than C1's and delay is about 3.5 times lower. When the probability of hot spot access is high (0.25), throughput for I1 and I2 is higher than C1 by a factor of about 7.8 and 5.25 respectively and delay is lower by a factor of about 8.5 and 6. The warm-traffic data show the same pattern.

Atomic Action Throughput	Low Sharing	Medium Sharing	Heavy Sharing
C1	0.0175 (.0175 - .0176)	0.0115 (.0115 - .0116)	0.0032 (.0031 - .0033)
I1	0.0301 (.0297 - .0304)	0.0288 (.0287 - .0289)	0.0207 (.0202 - .0212)
I2	0.0260 (.0258 - .0262)	0.0245 (.0243 - .0247)	0.0163 (.0159 - .1067)
Atomic Action Delay	**Low Sharing**	**Medium Sharing**	**Heavy Sharing**
C1	54.086 (53.907 - 54.264)	83.755 (83.185 - 84.325)	308.015 (296.555 - 319.475)
I1	32.869 (32.259 - 33.479)	34.356 (33.844 - 34.868)	47.973 (47.331 - 48.615)
I2	38.246 (38.011 - 38.482)	40.642 (40.260 - 41.024)	61.089 (59.694 - 62.48

TABLE 5. Series 5 warm-spot data. 99% confidence intervals are shown in parentheses)

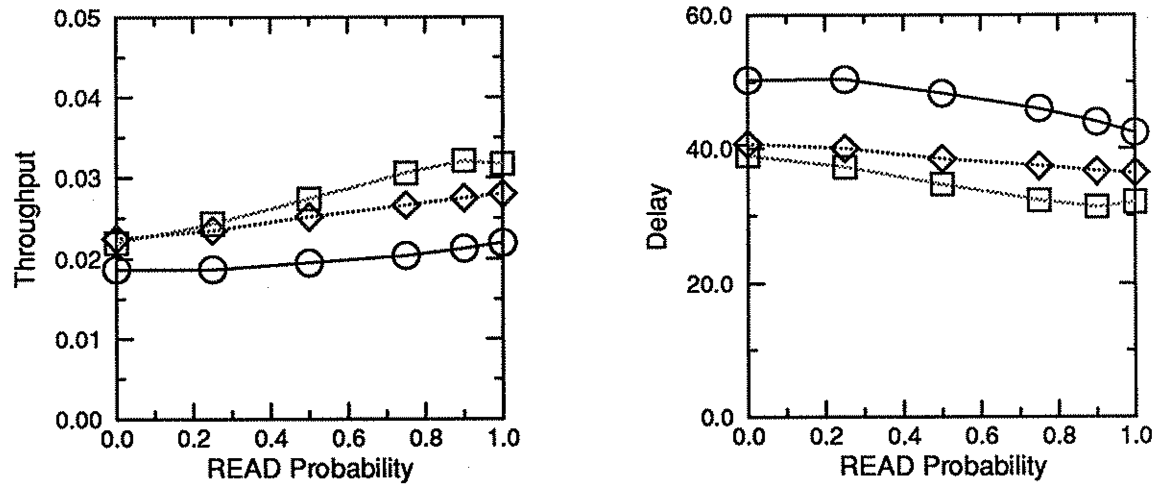
Atomic Action Size (Graphs 5-aa_size) - As in series 3 and 4, throughput and delay for this part are normalized for the atomic action size. As the atomic action mean size increases, the data show that throughput for the isotach networks initially increases, and then decreases, whereas normalized delay initially decreases and then increases. Throughput increases initially because increasing the atomic action size increases the number of operations presented to a network that is underutilized because of the cap of 1 on the number of active atomic actions. The subsequent decrease in throughput can be attributed to several factors:

1. the network becomes less efficient as the size of the atomic actions increases;
2. the percentage of atomic actions that include a WRITE increases with the result that a greater percentage of atomic actions must be executed in 2 stages; and
3. more operations are accessing the same number of variables with the result that the probability a READ waits on an unsubstantiated SCHED increases.



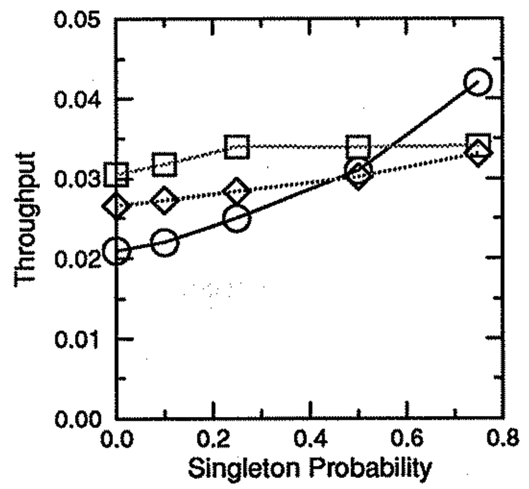
GRAPHS 5-AA_size - Series 5 data for variation of atomic action size. O - C1; ❑ - I1; ◊ - I2;

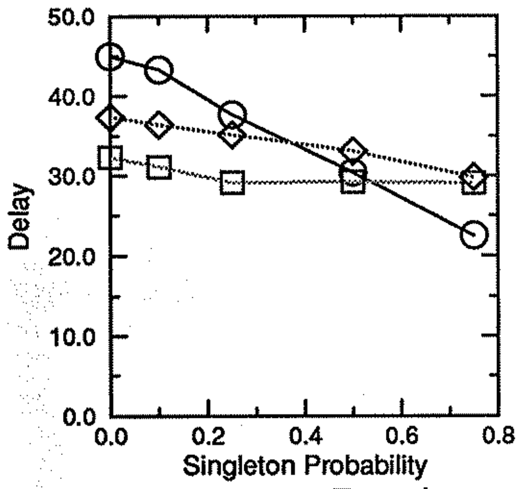
Delay increases for the reasons throughput decreases. The initial very small decrease in delay occurs because operations in the same atomic action are issued and executed concurrently so delay per operation tends to decrease as the number of operations per atomic action increases. Although the efficiency of I1 and I2 is much lower than in series 3 and 4, the isotach networks still outperform the conventional networks by a margin that increases as the atomic action size increases. When the atomic action mean size is 16, throughput is about 3.5 to 4.5 times higher for I1 and I2 than for C1 and delay about 4 to 5 times lower.



GRAPHS 5-read_prob - Series 5 data for variation of READ probability. O - C1; ❑ - I1; ◊ - I2;

Read Probability (Graph 5-read_prob) - The efficiency of each of the systems increases gradually as the probability of a read access increases. None of the systems is very sensitive to the percentage of reads because the probability that operations concurrently access the same variable is low, given the cap of 1 on the number of active atomic actions per PE and the uniform distribution of accesses.





GRAPHS 5-force_sing - Series 5 data for variation of singleton probability. O - C1; ❑ - I1; ◊ - I2;

Forced Singletons. (Graphs 5-force_sing) - This data shows that isotach networks continue to outperform conventional networks even under the performance limiting effects of this series until the forced-singleton probability reaches about .5 (meaning the percentage of singleton accesses is about 60%).

Series 5 shows that isotach systems can outperform conventional systems even for workloads in which the existence of data dependencies negates one of the principal advantages of isotach systems, the ability to pipeline operations while maintaining sequential consistency. The series indicates that the scheduling isochron technique is more efficient than 2PL for structured atomic actions of the type simulated, especially for large atomic actions or nonuniform access patterns.

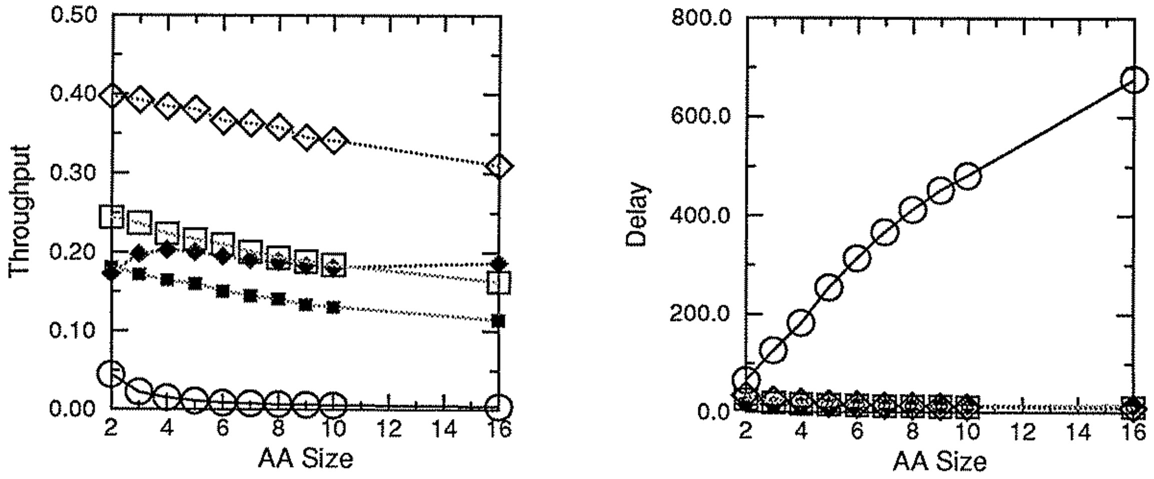
4.6 Warm Traffic Data

Series 6 compares the performance of the networks when accesses are not uniformly distributed. This series repeats parts of earlier series using an 80/20 rule warm traffic model in place of the uniform traffic model. We include this series to explore the effect of contention for shared variables in conventional and isotach systems.

This series also differs from earlier series in that it uses a switch buffer size of 2 for the isotach networks. C1 is slightly more efficient with a switch buffer size set at 1 instead of 2 and the isotach networks are slightly more efficient at 2 instead of 1. In all previous series, the switch buffer sizes are all set at 1. In this series we compare the conventional and isotach systems when each has its optimal switch buffer size: 1 for C1 and 2 for I1 and I2.

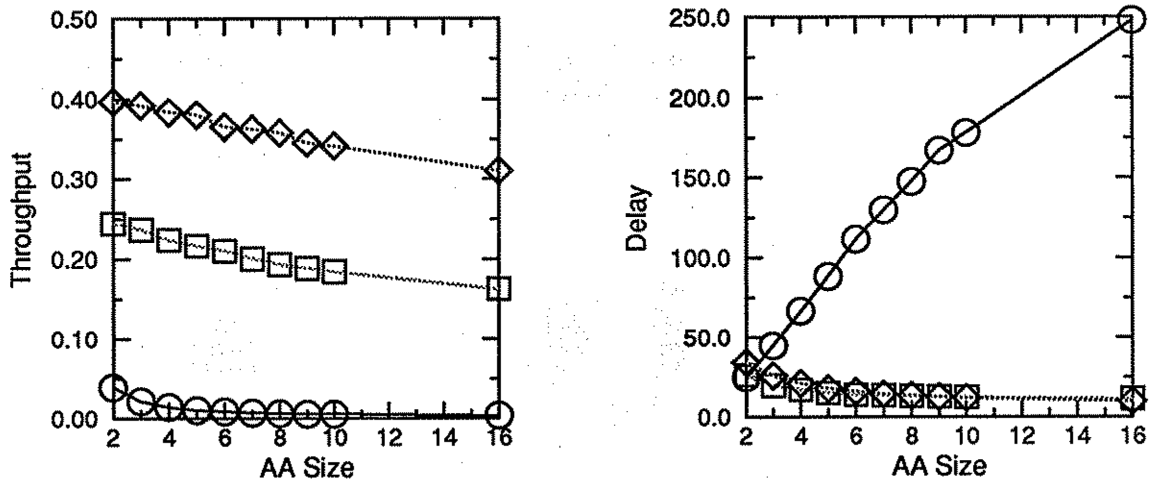
In the first part of series 6 (series 6.1), the workload model is the *atomicity only* model from series 3. In the second part (series 6.2), the workload model is the *flat atomic action* model from series 4.

Base Case - In the base case for series 6.1 (atomicity only), the throughput of the capped version of I2 is higher than C1's by a factor of about 9 and delay is about 8 times lower. In the base case of series 6.2 (flat atomic actions) throughput of I2 is 18 times higher than C1's and delay about half that of C1. When accesses are not uniformly distributed the margin by which isotach networks outperform conventional networks increases. The performance of the isotach networks in series 6.1 and 6.2 is similar to their performance in series 3 and 4. The throughput is essentially the same and the delay only slightly higher. The increase in the buffer size helps offset the decrease in throughput caused by the non-uniform distribution of accesses. The performance of C1, by contrast, is considerably worse in series 6. When the probability of conflicting operations is low, locks have less impact on performance than when it is high. The use of locks to enforce atomicity means C1's performance suffers when accesses are not uniformly distributed.



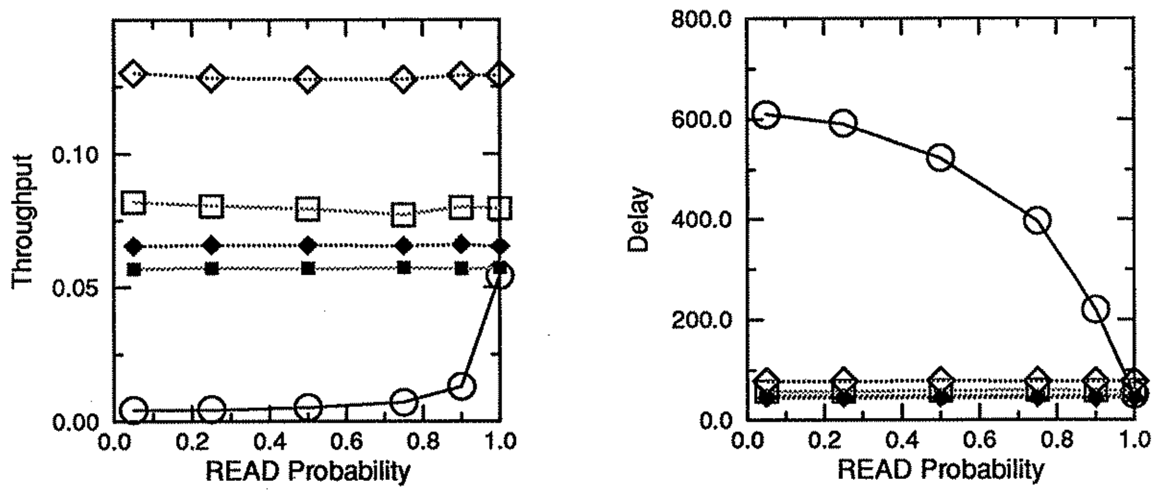
GRAPHS 6.1-AA-size - Series 6.1 data for variation of the atomic action size.
O - C1; ❑ - I1; ◊ - I2; ■ - I1 (AA Cap = 3); ◆ - I2 (AA Cap = 3);

Atomic Action Size - Graphs 6.1-AA_size and 6.2-AA_size show the effect of varying the atomic action size under a warm-traffic model. These graphs correspond to Graphs 3-AA_size and 4-AA_size, respectively. When atomic actions are large and traffic is non-uniform, C1's performance is very poor relative to the isotach networks. For large atomic actions (aa_size = 16), the throughput in series 6.1 of the capped version of I2 is about 43 times higher than C1's and delay about 57 times lower. In series 6.2, I2's throughput for large atomic actions is about 78 times higher than C1's and its delay about 24 times lower.



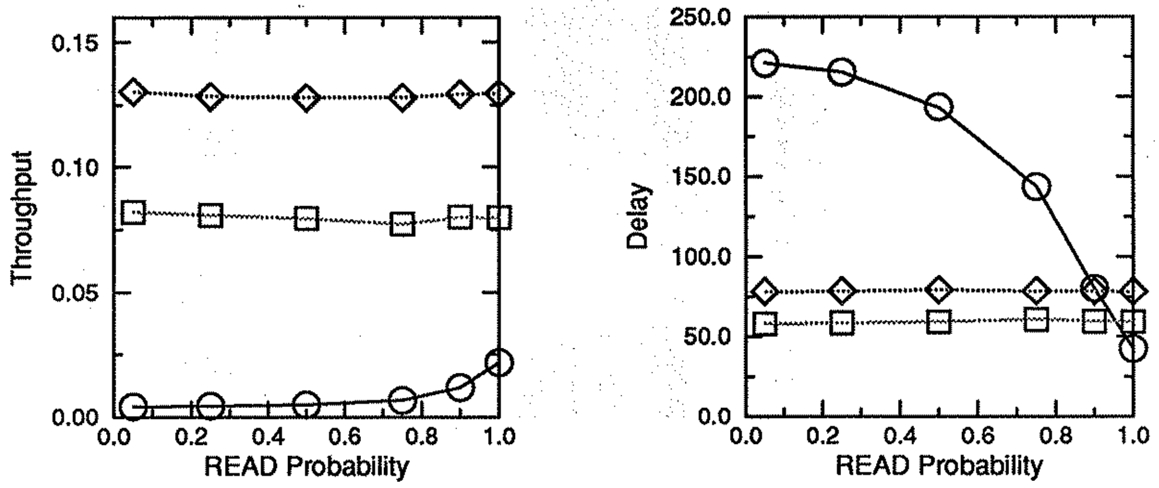
GRAPHS 6.2-AA_size - Series 6.2 data for variation of the atomic action size. O - C1; ❑ - I1; ◊ - I2;

Read Probability - Graphs 6.1-read_prob and 6.2-read_prob show the effect of varying the probability that an access is a READ. The isotach networks are not sensitive to the READ probability, but C1 is, and to a much greater extent than under the uniform traffic model. The effect on C1 is much greater under a warm traffic model because the probability that two operations conflict is significantly higher than when accesses are uniformly distributed. In the base case (read_prob = 75%), the percentage of locks granted immediately declines from 91% in series 3 to 81% in series 6.1 and from 97% in series 3 to 83% in 6.2. When the READ probability is low (25%), the throughput in series 6.1 of the capped version of I2 is higher than that of C1 by a factor of about 15 and its delay is lower by a factor of about 13. The comparable figures for series 6.2 are 29 for throughput and 2.7 for delay.



GRAPHS 6.1-read_prob - Series 6.1 data for variation of the READ probability.
O - C1; ❑ - I1; ◊ - I2; ■ - I1 (AA Cap = 3); ◆ - I2 (AA Cap = 3);

Series 6 indicates that isotach systems outperform conventional systems by a wider margin when accesses are not uniformly distributed.



GRAPHS 6.2-read_prob - Series 6.2 data for variation of READ probability. O - C1; ❑ - I1; ◊ - I2;

5.0 Conclusions

We have presented results from a simulation study comparing the performance of isotach and conventional networks. The study shows conventional networks have higher raw power than isotach networks, but that under a workload of operations with atomicity and sequencing constraints, isotach networks outperform conventional networks. Isotach networks perform best in relation to conventional networks under workloads with the following characteristics:

1. execution is required to be sequentially consistent;
2. the distance between data dependent operations within the same process's program is large enough to allow operations to be pipelined;
3. atomic actions are large; and
4. contention for shared variables is high.

When the workload has two or more of these characteristics, an isotach network performs markedly better than a conventional network. In some cases, the improvement in both throughput and delay is more than ten-fold.

The isotach systems outperform the conventional systems in spite of the lower raw power of the isotach networks. A conventional system cannot take advantage of the higher throughput and lower delay of its network when the synchronization techniques it uses work by restricting throughput and imposing delays. In conventional systems, the limiting factor on execution speed is not the network, but restrictions and delays imposed by the synchronization techniques. In isotach systems, the limiting factor is the network itself. Atomicity and sequencing constraints have markedly less effect on execution speed in isotach than in conventional systems.

The simulation results indicate that network throughput is of more importance relative to network latency in its effect on system performance in isotach systems than in conventional systems. Isotach systems are better able to take advantage of high throughput and are better able to use pipelining to mask latency. The high-throughput z-switch isotach network (I2) in most cases outperforms the lower throughput isotach switch (I1) in spite of the assumption that I2's cycle time is twice that of I1. In designing conventional networks, however, the simulation indicates that accepting worse delay for better throughput is a poor decision. The high-throughput z-switch C2 outperforms all other networks in raw power, but consistently performs worse than the simpler conventional switch C1 under a workload with synchronization requirements.

The most important assumptions made by the study are as follows: 1) the cycle time for each isotach network switch is the same as that of the corresponding conventional switch and the cycle time for a z-switch is twice that of a single-cycle switch -- in particular, that the latency of the isotach network which performs best (I2) is twice that of the conventional network that performs best (C1); 2) the workload models simulated are relevant to actual parallel computations; and 3) the conventional system simulated is a good basis for a fair comparison of conventional with isotach systems.

One of our goals for future research is the refinement of these assumptions. This work includes the following tasks:

1. Specifying the low-level design of the isotach network switch. A more detailed design should allow us to determine the relative cycle time of the isotach network switch with more certainty.
2. Extending the study to include more complex atomicity and sequencing constraints such as those implied by pointer-following or data dependencies between operations issued by different processes.
3. Driving the simulations with operation streams derived from actual programs or with a synthetic workload that more closely models typical parallel programs. Parallel programs typically use locks on execution paths to enforce atomicity instead of locks on individual variables. The method of enforcing atomicity assumed in the simulation is closer to that used in databases than in parallel programs.
4. Comparing isotach-based techniques to conventional techniques other than 2PL, such as timestamp or optimistic concurrency control techniques.

Another goal for future research is improving the raw power of the isotach network. We intend to investigate the following ideas for improving performance:

1. Substituting C1 for C2 as the return network (MM->PE) for I2. The higher latency network C2 was used as the return network for I1 although the simulation results indicate that C1 is a good match for I2 in throughput. Substitut-

ing C1 as the reverse network can be expected to decrease the latency of the I2 system without decreasing the maximum throughput. A preliminary investigation using the series 5 base case shows an improvement of about 7.5% in both latency and throughput.

2. Modifying the switch algorithm so that the synchronization delay imposed on an operation not subject to any atomicity or sequencing constraints is decreased or eliminated. The simulation shows isotach networks perform less well in relation to conventional networks when the workload does not require synchronization or when synchronization requirements are low. The reason for the relatively poor performance is that the isotach networks simulated impose a synchronization charge on every operation, whether the operation requires synchronization or not. The modification required to decrease the synchronization charge imposed on operations not subject to synchronization constraints appears to be straight-forward. A tag bit is added to each operation and the tag bit for an operation is set if the operation can be executed correctly in any order in relation to other operations. Each switch routes tagged operations contingent only on the availability of the required output, i.e., the switch would apply the local synchrony algorithm to maintain ordering by route-tag only among the untagged operations. We would expect the modified algorithm to improve the performance of the isotach networks in series 1, 3, 5 (the ASSIGN operations can be tagged), and where the percentage of singleton atomic actions is high and sequential consistency is not required.
3. Modifying I2 to decrease latency. The current switch design of I2 is based on 2 two-step cycles. We believe it may be feasible to design a switch based on 3 single-step cycles with a latency closer to 1.5 times that of the low throughput switch.

A third goal for future research is generalizing the study of the isotach network performance to other systems or networks, as follows:

1. Networks that use worm-hole or virtual cut-through instead of store-and-forward routing.
2. Network switches with more than 2 inputs and outputs.
3. Systems with privately cached shared variables. Isotach networks support a family of new cache coherence protocols that are more concurrent than protocols for conventional systems [WiR90]. We expect inclusion of caches to increase the margin by which isotach systems outperform conventional systems. Private caches also replace the elements in access sequences, eliminating the need for access sequences.
4. Recombining networks [KSS88] that can combine concurrently issued operations that access the same variable. Although conventional systems can benefit from combining, we expect the benefit to be greater in isotach systems. In

isotach systems, unlike conventional systems, the network can combine operations from different non-trivial atomic actions and yet maintain atomicity and sequential consistency [WiR91].

5. Isotach networks with other topologies, in particular the mesh and hypercube topologies.

BIBLIOGRAPHY

[Awe85] B. Awerbuch, "Complexity of Network Synchronization", *Journal of the ACM 32*, 4 (October 1985), 804 -823.

[BGS89] Y. Birk, P. B. Gibbons, J. L. C. Sanz, and D. Soroker, "A Simple Mechanism for Efficient Barrier Synchronization in MIMD Machines", *IBM Technical Report RJ 7078 (67141)*, October 1989.

EGL76] K. P. Esarwan, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System", *Communications of the ACM 19*, 11 (November 1976), 624-653.

[Fid91] C. Fidge, "Logical Time in Distributed Computing Systems", *Computer*, August 1991, 28-33.

[KHM87] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input Versus Output Queueing on a Space-Division Packet Switch", *IEEE Transactions on Communications COM-35*, 12 (December 1987), 347-356.

[KRS88] C. P. Kruskal, L. Rudolph, and M. Snir, "Efficient Synchronization on Multiprocessors with Shared Memory", A*CM Transactions on Programming Languages and Systems 10*, 4 (October 1988), 579-601.

[Knu73] D. E. Knuth, *Fundamental Algorithms, Vol. 3, Sorting and Searching,* Addison-Wesley. 1973, p. 397.

[KuJ84] M. Kumar and J. R. Jump, "Performance Enhancement of Buffered Delta Networks Using Crossbar Switches and Multiple Links", *Journal of Parallel and Distributed Computing 1* (1984), 81-103.

[Lam78] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", *Communications of the ACM 21*, 7 (July 1978), 558-565.

[Lam79] L. Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Mulitprocess Programs", *IEEE Transactions on Computers C-28*, 9 (September 1979), 690-691.

[Lom77] D. B. Lomet, "Process Structuring, Synchronization, and Recovery Using Atomic Actions", *Proc. Conf. on Language Design for Reliable Software, SIGPLAN Notices 12*, 3 (March 1977), 128-137.

[Mat88] F. Mattern, "Virtual Time and Global States of Distributed Systems", *Parallel and Distributed Algorithms*, 1988, 215-226.

[OwG76] S. Owicki and D. Gries, "An Axiomatic Proof Technique for Parallel Programs I", *Acta Informatica 6* (1976), 319-340.

[Ran87] A. G. Ranade, “How to Emulate Shared Memory”, in *Proceedings of the Annual Symposium on the Foundations of Computer Science ‘87*, IEEE, 1987, 185-194.

[ReF87] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, MIT Press, Cambridge, Mass., 1987.

[ReW91] P. F. Reynolds, Jr. and R. R. Wagner, Jr., “A Local Synchrony Implementation: Banyan Networks”, *UVA Computer Science Technical Report* 91-38, December 1991.

[RWW89] P. F. Reynolds, Jr., C. Williams and R. R. Wagner, Jr., “Parallel Operations”, *UVA Computer Science Technical Report* 89-16, December 1989.

[Wag87] R. R. Wagner, Jr., *Parallel Operations in Shared Memory*, Master’s Thesis, University of Virginia, 1987.

[WiR89] C. Williams and P. F. Reynolds, Jr., “On Variables as Access Sequences in Parallel Asynchronous Computations”, *UVA Computer Science Technical Report* 89-17, December 1989.

[WiR91] C. Williams and P. F. Reynolds, Jr., “Combining Atomic Actions in a Recombining Network”, *UVA Computer Science Technical Report* 91-33, November, 1991.