Performance Evaluation of the Late Delta Cache Coherence Protocol

Bronis R. de Supinski
Craig Williams
Paul F. Reynolds, Jr.

Computer Science Report No. CS-96-05
March 13, 1996

Performance Evaluation of the Late Delta Cache Coherence Protocol

Abstract

This paper presents the results of a simulation study designed to compare the performance of the late delta cache coherence protocol and a conventional directory based invalidation protocol. Delta cache protocols are a highly concurrent directory based family of coherence protocols which exploit an isotach logical time system to provide support for sequential consistency and atomicity. The late delta protocol is an update based member of this family of protocols. Our results demonstrate the efficacy of the late delta protocol across a wide range of workloads. We show that this protocol provides comparable performance to the conventional invalidation protocol under workloads with no atomicity requirements and little contention, but outperforms the conventional invalidation protocol as atomicity requirements and contention increase.

1. Introduction

Cache memories have proven an effective technique to reduce memory latency in uniprocessors. In shared memory systems, the use of cache memories for shared data creates the potential for multiple copies of the same memory location to exist in the system concurrently. The *cache coherence problem* is the problem of ensuring that the replicas of a memory location provide a logically consistent view of the global address space. The *delta cache protocols* [WRd95] are a novel and highly concurrent family of protocols which solve this problem. This family of protocols exploit an *isotach logical time system* to provide support for sequential consistency and atomicity. We have conducted a performance study comparing one of these protocols, the *late delta cache coherence protocol* to a conventional directory based, invalidation protocol [CeF78, CKA91]. In this study, we expected the late delta protocol to yield excellent performance for workloads with atomicity requirements without sacrificing sequential consistency despite the higher network latencies required to maintain isotach logical time. Our results demonstrate that the late delta protocol offers significant performance benefits.

Our results demonstrate the efficacy of the late delta cache protocol across a wide range of workloads. Significant observations from our study include the following.

- The late delta protocol provides good performance when contention for shared variables is high. The conventional invalidation protocol does not.
- As the number of processors increases, contention for shared variables becomes more severe. The late delta protocol performs well as the number of processors is increased. The conventional invalidation protocol does not.
- Hardware multicast significantly improves the performance of the late delta protocol, but not that of the conventional invalidation protocol.
- The late delta protocol provides good performance under workloads with atomicity constraints. The conventional invalidation protocol provides good performance only if lock contention is low.

In summary, the late delta protocol provides comparable performance to the conventional invalidation protocol under workloads with no atomicity requirements or with little contention, but outperforms the conventional invalidation protocol as atomicity requirements and contention increase.

In the following section, we briefly describe the protocols we have simulated. In Section 3, we present our simulation methodology including a description of the synthetic workload we have developed to evaluate the performance of the delta cache protocols. This workload model encompasses previous synthetic workload models but also provides explicit modeling of data dependence and atomicity requirements. We then present our performance study comparing the late delta protocol to the conventional protocol.

2. Protocols Simulated

We present an evaluation of systems which implement the late delta protocol and a conventional invalidation protocol. All of the systems we have simulated are organized in a standard dance hall topology. Processing elements are connected to memory modules by a multistage interconnection network (MIN). The networks are composed of standard 2X2 crossbar switches, which have been simulated in detail. For the simulations of the late delta protocol, we have used the switch algorithm of network I1 described in a previous performance evaluation of isotach networks [RWW92].

Delta cache protocols use the properties of an isotach logical time system to provide support for sequential consistency [Lam79] and atomicity without the use of locks or other special synchronization constructs. The protocols rely on the interconnection network to maintain isotach logical time. The routing algorithm of the switches of a MIN can be modified slightly to maintain isotach logical time. A previous study [RWW92] has shown that these modifications result in higher raw network latency. To capture this cost, we have modelled the interconnection network explicitly in this study. Members of the delta cache family of protocols support multiple concurrent readers and writers and allow the pipelining of memory accesses. The late delta protocol is an update based member of this protocol family. A full description of the protocol is available in [WiR90].

We compare the performance of an isotach system which employs the late delta protocol to maintain coherence to systems which maintain cache coherence with a conventional directory based invalidation protocol. Existing directory based update protocols either sacrifice sequential consistency [DKC93] or require expensive techniques, such as two phased locking to propagate the updates [WiL92]. We expect the late delta protocol to provide excellent performance without sacrificing sequential consistencyThus, the conventional invalidation protocol provides the most reasonable existing comparison

Our conventional protocol is based on the protocol used in the Alewife machine [CKA91]. We have modified the Alewife protocol slightly to simplify its simulation. In the Alewife protocol, if an operation is received to a block for which there are unacknowledged invalidations, a “busy” response is returned to the requestor. In our conventional protocol, operations which would receive a “busy” response are buffered at the memory module (MM). Once the invalidation phase is complete, the MM executes any buffered operations in the order in which they were received. A similar situation arises when an MM receives a read request to a block for which there is an exclusive copy. In this case, the MM requests that the writer share the block. The writer, which retains a valid read-only copy, responds to this request by providing the value of the block to the MM for distribution to any new readers. While the MM is awaiting the value, it may receive additional operations. The Alewife protocol uses the “busy” response for both read and write operations in this situation as well. In our conventional protocol, if the MM receives an additional read operation, the requestor is added to the directory. Once the MM receives the value from the writer, it multicasts the value to all other processing elements (PE’s) in the directory. Any write operations are buffered exactly as during an invalidation phase. Our conventional protocol is nearly identical to

the Alewife protocol since the Alewife protocol essentially uses the network to do the buffering of our conventional protocol. Our conventional protocol is more effi cient for concurrent reads and guarantees that a writer will not suffer from starvation. Although our conventional protocol would pose implementation diffi culties, it pɒvides a realistic upper bound for performance of conventional, directory based invalidation protocols.

The late delta cache protocol supports atomic execution of multiple operations to shared variables; our conventional protocol does not. We have implemented a lock based mechanism to provide atomicity in the systems which use the conventional protocol. Lock operations are performed in a separate synchronization memory space which is physically collocated with the memory modules. This memory has separate input and output queues which are split from and merged with the streams of the actual memory module at the network interface of the unit. Since locks are a critical resource for our conventional systems in workloads with atomicity requirements, we assume the cycle time of the synchronization memory is comparable to that of the cache memories. Requests for locks which are currently held by another PE are queued at the synchronization memory. When a lock release is received, outstanding lock requests are granted in the order they were received. In order to assure freedom from deadlock, a PE can have no more than one outstanding lock request at any time. Thus at most N - 1 lock requests can be buffered at a synchronization memory. Therefore, this synchronization memory could be realized in actual hardware. More importantly, we are primarily concerned with evaluating the performance of delta cache systems and so are motivated to provide effi cient, if not entirely realistic, support for locking in the conventional systems.

In our workload model, each process executes a sequence of critical sections. Access to critical sections with possibly conflicting operations must be governed by a common lock. PE's must complete lock requests one at a time and in numeric order to prevent deadlock. PE's must obtain locks prior to executing the operations of critical sections governed by the locks. Finally, a PE must determine when no violations of atomicity can occur and thus it is safe to release the locks. These requirements increase programming effort and reduce concurrency. In current multiprocessors, all of the synchronization work required of the processing elements is performed by the processor itself, reducing the cycles available to perform useful computation.

We have designed a novel synchronization coprocessor for conventional cache coherent systems. This coprocessor allows increased pipelining of atomic actions and it removes the burden of maintaining sequential consistency from the processor. Further, it halves the main processor cycles required to provide atomicity in systems which use locks. For its correct and effi cient operation, we assume that private memory operations aɐ distinguished from shared and that the last operation of an atomic action involving shared data is marked accordingly. We assume that atomicity is provided through locks and that the necessary lock request operations, which can be distinguished from other memory accesses, are issued by the processor. Figure I provides a detailed diagram of our synchronization coprocessor.

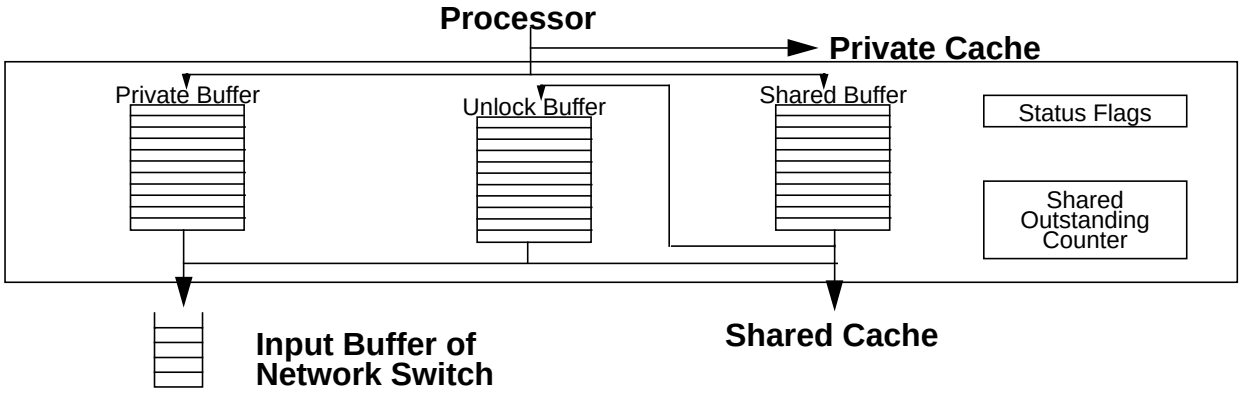


Figure I: Synchronization Coprocessor

The primary function of the synchronization coprocessor is to emit operations into the network. The processor issues data and lock operations to the synchronization coprocessor. Cache misses to private data are buffered separately from all other operations issued to the synchronization coprocessor. Operations to private data bypass the synchronization coprocessor if they are found in the private data cache.

The synchronization coprocessor chooses from available operations to emit to the network. The status flags of the coprocessor track the availability of operations present in its buffers. Whenever operations in the unlock buffer are available and there is room in the network, the synchronization coprocessor emits the head of the buffer into the network. If operations in the unlock buffer are not available, the synchronization coprocessor randomly chooses an operation to emit into the network from the head of either the private buffer or shared buffer if operations are available in both. If operations are present in the private buffer, they are available since no violations of sequential consistency or atomicity can occur as a result of accesses to private data. The availability of operations in the shared buffer is constrained to maintain sequential consistency and atomicity.

The availability of operations in the shared buffer is determined by the operations which have been emitted from the buffer. When a lock request is emitted into the network, the shared buffer becomes unavailable until the synchronization coprocessor receives the corresponding lock grant. In addition, when the synchronization coprocessor emits a lock request into the network, a lock release operation is enqueued in the unlock buffer. If an operation to shared data is emitted into the network, then the shared outstanding counter is incremented. If the shared data operation is the last operation of an atomic action, the shared buffer becomes unavailable until the shared outstanding counter is zero. The shared outstanding counter is decremented when operations are received from the network at the cache for shared data.

When a shared atomic action is completed, as detected by the zeroing of the shared outstanding counter, operations in the unlock buffer become available for emission into the network. Once the unlock buffer is emptied, it again becomes unavailable until the next shared atomic action is completed. By giving priority to lock release operations, the coprocessor minimizes the blocking of other PE's waiting for the locks it holds.

We expected the synchronization coprocessor to improve the performance of the conventional protocol for workloads with atomicity requirements. Since the processor is not responsible for freeing locks, the number of processor cycles required for locking operations is halved. The processor can pipeline shared atomic actions since the synchronization coprocessor maintains sequential consistency. Our results indicate that the synchronization coprocessor can improve the performance of the conventional protocol. However, since lock contention is not alleviated, the performance gains are small.

3. Simulation Methodology

In this section, we describe the method we have used to generate memory references for our simulations, as well as other important details of our simulation methodology. A variety of approaches have been used to evaluate cache coherence protocols. Several researchers have modeled cache coherence protocols as Markov processes [DuB82] and attempted to derive closed form solutions for these systems. Others have used mean value analysis to compare protocols [VLZ88, BLA89, YBL89, ADT90]. Generally these approaches have relied on system simulation to verify the models proposed. Much research has been conducted using simulations to evaluate cache coherence protocols directly. Researchers have used both synthetic workloads, generally based on the workload model developed by Archibald and Baer [ArB86], and memory traces to generate a reference stream [EgK88, BMR89, MiB92, ASH88]. Still others have simulated an entire computer system in software [CKA91] or used the execution driven approach, where actual programs are run on the host computer and memory references are trapped to a memory hierarchy simulation as necessary [DGH90, CDK94].

We have evaluated the late delta protocol and our synchronization coprocessor through simulation using a synthetic workload. This is the most tractable and accurate approach now available to us. The effi cient support for atomicity provided by delta cache systems eliminates the use of locks. Therefore, we have rejected trace based simulation for our performance study since existing memory traces do not reflect the high level of concurrent access of overlapping atomic actions anticipated in isotach systems. We must delay an execution driven evaluation of delta cache systems due to our current lack of both a compiler for isotach based systems and body of programs which exploit the many novel features of isotach systems. We are developing the needed compiler and will expand this study to include execution driven evaluation of delta cache protocols. Synthetic workload generation remains an important tool for evaluating cache coherent systems since a wide variety of workloads can be generated and evaluated quickly and easily.

3.1. Synthetic Workload Generation

We base the synthetic workload model used by our simulation upon a signifi cant body of research into the performance of cache coherence protocols. Archibald and Baer [ArB86] compared a wide variety of bus based cache coherence protocols using a simulation driven by a synthetic workload. Nearly all synthetic workload simulations of cache coherence [e.g. LeR90, NaB93] have been based on this research. The remainder of this subsection details the synthetic workload model we have used. Our model, which builds

on that used in several previous studies, allows the evaluation of many important features of delta cache protocols.

As in previous studies, we model the processors of cache coherent systems as simple state machines. Processors spend some number of cycles "thinking," that is, performing useful computation. At the end of each think period, the processor instantaneously generates references, which it then issues subject to sequencing constraints. Once all references have been issued and any data dependencies required for the next computation period have been satisfied, the processor then begins a new think period. Our method for determining data dependence constraints is explained in the following paragraph. The sequencing constraints are system dependent. All of the systems which we evaluate provide a sequentially consistent memory, although the manner in which this is achieved varies. With the late delta protocol or our synchronization coprocessor, the processor is not responsible for maintaining sequential consistency, which is guaranteed by other features of the systems. With our conventional protocol, the processor must determine that no violations of sequential consistency can occur prior to issuing a set of references. Think periods are traditionally expressed in processor cycles and drawn from some distribution. We have drawn think period lengths from a truncated exponential distribution with a mean of 2 processor cycles and a maximum of 5, typical of values used in previous studies.

We have modified the traditional synthetic workload to model data dependencies. We define the data distance of a read as the number of think periods before the reference is required. We do not include write references in our data dependence model. Before a think period is begun, the processor checks for any outstanding references which are required for the computation of the upcoming think period. If so, the processor stalls until the references return. Data distances are determined at the time the references are generated and are selected from a truncated exponential distribution.

References can be either to private or shared data. References to private data must be modeled since they comprise the overwhelming majority of all memory references. Since cache misses for private data must cross the interconnection network and delta cache systems require the use of isotach networks which have higher raw latency, private reference modeling is particularly important for an accurate comparison of delta cache systems to conventional cache coherent strategies. Previous studies have assumed that cache performance for private data in multiprocessor systems is consistent with uniprocessor cache performance. Since we model separate data caches for private and shared data [ADT90], this assumption appears particularly valid. We assume a warm start, where a start up period elapses prior to performance measurement and eliminates the effect of compulsory misses. This assumption allows the use of a constant private hit ratio during any simulation run. Another important parameter for private cache behavior is the percentage of blocks which are dirty upon replacement [ArB86]. The private cache hit ratio is 0.95 with a probability of 0.3 that private cache victims must be written back in all of our experiments. These values are consistent with those used by Archibald and Baer [ArB86] and studies of uniprocessor cache performance.

Since our goal is to determine the effect of shared cache activity on performance, shared cache blocks must be modeled explicitly. Early research [DuB82] proposed that temporal locality would be significantly reduced for shared data, and so proposed modeling shared references with the uniform reference model. However, later studies have chosen to model shared references with temporal locality by using a least recently used stack model (LRUSM) [Spi77]. This assumption is supported by shared memory reference pattern studies [AgG88]. This model allows cache blocks which have recently been selected to be significantly more likely to be selected again, which is important if caching is going to provide much reduction in memory latency. We have modified this approach to model contention for shared variables directly. These modifications are detailed in Section 3.1.2.

In traditional synthetic workloads, two important input parameters for reference generation which we have maintained in our workload model are the probability that any given reference is a write and the probability that any given reference is to shared data. We have used a write probability of 0.3 throughout our experiments. Shared reference pattern studies indicate that the vast majority of dynamic references is to private data [DPS86, BaR89]. For this reason, we have parameterized our reference generation method to keep the percentage of references to shared addresses low, typically about 5 or 10 percent of all references.

Previous synthetic workload models are not sufficient for evaluating the performance of delta cache protocols, which provide support for atomicity and allow pipelining of atomic actions. Few model atomicity requirements [LeR90]. Further, previous workloads provided no direct method for manipulating the degree of contention for shared variables. We expect the effect of varying this contention to reveal significant differences between the systems we are evaluating. When contention for shared variables is high, invalidation protocols will perform poorly, while the highly concurrent nature of delta cache protocols should provide good performance regardless of the level of contention. To study the effect of atomicity constraints and contention for shared variables, we have expanded the traditional synthetic workload model. We incorporated atomicity through reference periods which generate batches of references and added a parameter to control the degree of contention. The remainder of this section details these innovations.

3.1.1. Incorporating Atomicity

An important feature of the systems we have proposed is that they provide support for atomicity. The basis of our atomicity model is derived from the critical section model proposed by Min, Baer and Kim [MBK90]. In our model, a processor issues a batch of one or more references after each think period. We have chosen to model the batches as consisting entirely of either private or shared references. We have made this choice since we assume conventional systems benefit by minimizing the size of shared atomic actions, thus reducing the time that locks are held.

The mean sizes of the two types of reference batches are important factors in the behavior of our synthetic workload. The size of each private batch is determined from a truncated exponential distribution, the mean and maximum of which are simulation input

parameters. As previously stated, we do not model private cache blocks explicitly. Instead, we determine from the specified private hit ratio whether a given private reference is present in the local cache. If not, then it is located at any of the system memory modules with uniform probability.

We model the composition of shared atomic actions explicitly. Each shared variable is assigned to one or more atomic actions, with possibly some of the assignments being read only. If the assignment is not read only, i.e. it is writable, then the type of access to the variable is determined each time a reference to the atomic action is generated. The assignment of shared variables to atomic actions is determined from a truncated exponential distribution, subject to a parameterized constraint on shared atomic action size. The probability that these assignments are writable is also an input parameter.

Through these parameters we can specify a variety of reference batch compositions. In particular, we can specify a workload with no atomicity requirements. This workload corresponds to that used in most previous cache coherence studies which used a synthetic workload. We have performed extensive simulations with no atomicity requirements to demonstrate the efficacy of delta cache protocols even under workloads which do not exploit all of their features.

The probability that a reference is to shared data and probability that a reference is a write are input parameters to our workload model. Since our model can generate several references at a time and some of the shared variable assignments may be read-only, we cannot use these parameters directly. Instead, we use these parameters to calculate the probability that a batch of references is to a shared atomic action and the probability that a writable shared reference is a write. We have confirmed the resulting workload has the desired proportion of writes and shared accesses.

Our conventional protocol, even when using the synchronization coprocessor, requires locks to ensure that shared atomic actions are executed without conflicting, interleaved accesses by other processors, i.e. atomically. Thus we must provide a reasonable method for assigning locks to shared atomic actions. Given the composition of the shared atomic actions, we can define a critical section graph [MBK90]. Each shared atomic action is a node in this graph. The edges of the graph are determined by the possible conflicting accesses of these atomic actions. Two atomic actions have possible conflicting accesses if the set of writable variables of one intersects with the set of all variables of the other, whether the variables are read only or writable. Using this critical section graph, atomicity can be guaranteed using a variety of lock assignments. We have implemented two. The super-critical section method finds the connected components of the critical section graph and assigns a single lock to each. The other assigns a lock to each edge of the critical section graph. If the connected components are not well connected, then the edge locking approach provides significantly greater opportunity for concurrency. However, our initial results indicate that the cost of acquiring additional locks dominates any performance gain resulting from this additional concurrency. Therefore, we report results only for the super-critical section method.

The stochastically generated sets of shared atomic actions can be saved and reused in later runs. We have run extensive simulations using three different sets of shared atomic actions. Each set has 8192 shared variables. CS Set I has a highly connected critical section graph of 7634 atomic actions. Of these atomic actions, 7328 are in a single connected component. The remaining 306 atomic actions are found in small components of 1 to 3 nodes. The mean size of an atomic action is 2.8 shared variables with no atomic action accessing more than 6 shared variables. CS Set II has 2971 atomic actions which are primarily found in many small connected components. There is one large connected component with 954 nodes. However, no other component has more than 32 nodes and 1781 (60%) of the atomic actions are in components of 10 nodes or less. The mean size of a connected component is 2.4 nodes. The mean size of an atomic action is 3.5 shared variables with no atomic action accessing more than 6 shared variables. CS Set III has 7626 atomic actions, 7304 of which are in 32 approximately equally sized components. The remaining 322 atomic actions are found in components of 1 to 3 nodes. The mean size of an atomic action is 2.9 shared variables with none accessing more than 6 shared variables. For all three sets, approximately 60% of the shared variable assignments are writable.

Atomic actions can be classified as either flat or structured. Structured atomic actions have internal data dependencies while flat ones do not. Our simulations can generate workloads with structured shared atomic actions through two parameters. One parameter allows a write to a shared variable to have an internal data dependence on the previous value of the variable. Thus, the variable must also be read as part of the atomic action and the read must return before the atomic action can complete. Additionally, given that a shared atomic action contains both reads and writes, the writes may be dependent upon the values returned by some of the reads. Thus, for mixed atomic actions, each read may be required for an internal data dependence with some input probability. For any reads which are not part of an internal data dependence, the data distance is determined as described in the previous section. When these input parameters are both zero, then the workload will consist entirely of flat atomic actions.

3.1.2. Modeling Contention for Shared Variables

Most previous synthetic workloads used to evaluate cache coherence protocols have not directly modeled contention for shared variables. We have chosen to include a hot spot model in our workload [PfN85]. Given that a processor is referencing a shared atomic action, with some probability, the atomic action is the hot atomic action. The hot atomic action changes during a simulation run. Periodically, a single atomic action for all processors is chosen uniformly from the set of all shared atomic actions. To modify this to a warm spot model, multiple atomic actions can be hot simultaneously. Given that a hot atomic action is referenced, we choose from the set of hot atomic actions uniformly. This method models contention for shared variables through the use of a single hot spot set for all processors. Since our preliminary results indicate that increasing the size of the hot spot set is analogous to decreasing the hot spot probability, we present results only for a single hot atomic action.

If a reference to a shared atomic action is generated and a hot spot atomic action is not chosen, we use a LRUSM of all shared atomic actions to determine which atomic action to reference. Given that a hot spot atomic action is chosen, it is possible to modify the LRUSM stack to reflect this access. If the LRUSM stack were modified, then the probability of selecting a hot atomic action would vary greatly with the recent history of shared atomic action accesses. Further, since hot atomic actions would be likely to be near the top of LRUSM stack, they would exhibit a cool down period after their hot spot period has elapsed. The cleaner semantics have therefore motivated us not to modify the LRUSM stack.

3.2. Statistical Methodology

Our primary performance metric is processing power [ArB86], which is the sum of the utilization of all processors in the system. When designing a simulation for systems as complex as cache coherent multiprocessors, an important consideration is how confidence intervals will be gathered for the statistics of interest. In order to minimize the number of runs required to gather these statistics since each run requires significant processor time, we have adopted a fixed sample size procedure [LaK92]. This procedure allows a single long simulation run to provide an accurate confidence interval for the processing power metric.

We are interested primarily in the steady state performance of the systems, so the simulation is run for a significant period of time prior to gathering statistics. This allows the shared memory caches to be seeded and eliminates the initial transient behavior. Thus our results apply to a warm start for these systems. A study of the cold start performance of these systems would require that private references be modeled explicitly so that the transient behavior of the entire system would be modeled.

The fixed sample size procedure employed in our simulation is the standardized time series method. After the initial transient period, processing power estimates are gathered periodically. The observations are grouped into batches. Using these batches we are able to compute a confidence interval for the processing power of the system. The point estimate of the system processing power is the grand sample mean of the observations.

4. Performance Evaluation

In this section, we present our initial performance study. All of our results are for systems connected in a dance hall topology by an equidistant multistage interconnection network. We use a shared memory space of 8192 shared variables and a fixed shared memory cache size of 1024 shared variables per processing element. All runs are for systems with 32 nodes, except where noted. Our simulation has considerable memory requirements and obtaining reasonable confidence intervals requires long simulation runs. For these reasons, we model a system with a relatively small shared memory space. Sensitivity studies indicate that increasing the size of the shared memory space has little effect on system performance. For simplicity, we assume each cache blocks holds exactly one variable. Although this cache block size is unrealistically small, we expect increasing

the block size to favor the late delta protocol since increasing the block size favors update protocols [WLT93]. We first compare the sensitivity of the systems to contention for shared variables. We then investigate the scalability of the protocols and the effect of varying the probability of referencing shared data.

4.1. Varying Hot Spot Probability

Initially, we compare the behavior of the protocol variations when subjected to different hot spot probabilities. The hot spot probability in this study is conditioned on a shared atomic action being referenced. The probability that any given reference batch is to a hot atomic action is the product of this probability and the probability that any given reference batch is to shared data. In all of our runs, this total probability, combined with the low bandwidth requirements of the workload, is not large enough to create tree saturation [PfN85]. Thus performance of the protocols is limited by the performance of the protocol under the amount of contention implied by the hot spot accesses.

We have performed all of our experiments which vary the hot spot probability with a probability of any given reference being to shared data of both 0.05 and 0.1. Our results demonstrate that contention for shared variables is dominated by the probability of accessing the hot spot. In almost all cases, we found that halving the shared probability was essentially equivalent to halving the hot spot probability. Therefore, we generally omit our results for a probability of any given reference being to shared data of 0.1.

In the next section we present results for all systems under a workload with no atomicity requirements. These results demonstrate that the late delta protocol combined with networks which provide hardware multicast results in a system which is insensitive to the level of contention for shared variables. We then extend this result to workloads which include atomicity requirements and demonstrate that the late delta cache protocol exhibits little sensitivity to the atomicity requirements of the workload. Also, our experiments demonstrate significant sensitivity to both contention for shared variables and atomicity requirements in systems which use the conventional invalidation protocol.

4.1.1. Software Multicast Versus Hardware Multicast

The cache coherence protocols that we are evaluating employ multicast messages. These multicasts can be implemented whether or not the underlying network provides multicast support. If the underlying network does not provide multicast support, then the multicasts must be implemented as a series of sequential messages, which we refer to as software multicast. We first compare system performance under a workload with no atomicity requirements using networks with and without hardware multicast support. The results of this experiment are shown in Figure II. All systems with software multicast suffer significant performance degradation as the hot spot probability is increased. Our experiments with a hardware multicast facility confirm that the facility improves performance for all of the systems. However, the systems which employ the conventional protocol still exhibit significant sensitivity to contention for shared variables, whereas the late delta protocol suffers almost no performance degradation as the hot spot probability

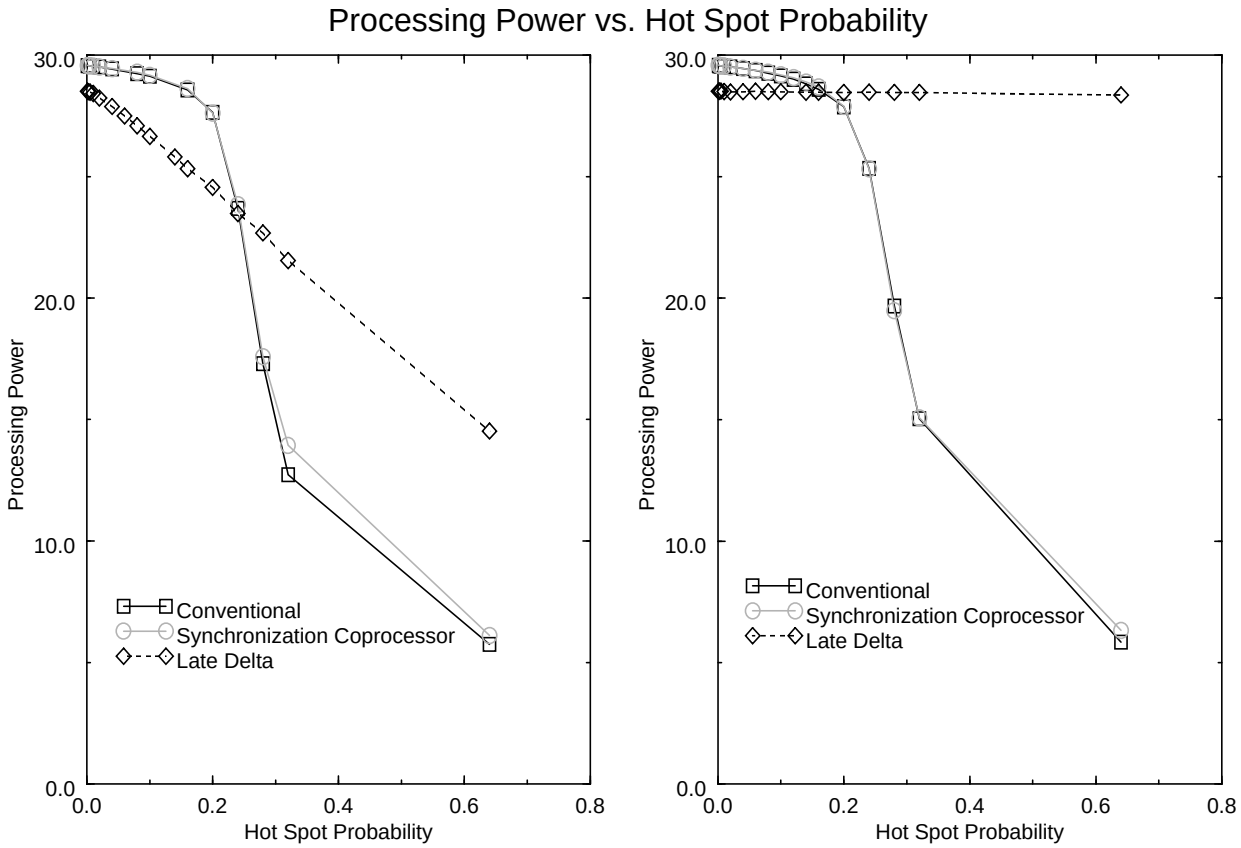


Figure II: Processing Power with No Atomicity Requirements

is increased. We observe that, in isotach systems, software multicast signifi cantly decreases the overall rate of logical time to real time. With software multicast, each multicast results in a stream of messages which effectively stagnate the fl ow of logical time. Although a similar effect occurs with the conventional protocol, the impact on logical time results in a system-wide performance loss, compared to the local effect observed with the conventional protocol. The dramatic improvement of the late delta cache protocol with hardware multicast results from the removal of the system-wide effect of the software multicast scheme.

To determine the cause of the performance degradation with software multicast seen in all protocols, we examined the access latencies. We defi ne the access latencies as the time elapsed from when the processor issues the access until it is completed at the PE. We measure access latencies for both shared and private accesses. We found that the comparative performance of the protocols refl ected the access latency of shared operations, with the increase for the late delta protocol being signifi cantly less than that with the conventional protocol. However, there was virtually no change in the private access latency with the conventional protocol, while a signifi cant increase was observed for the late delta protocol. The increase in private access latency for the late delta protocol indicates a global effect of increasing the hot spot probability.

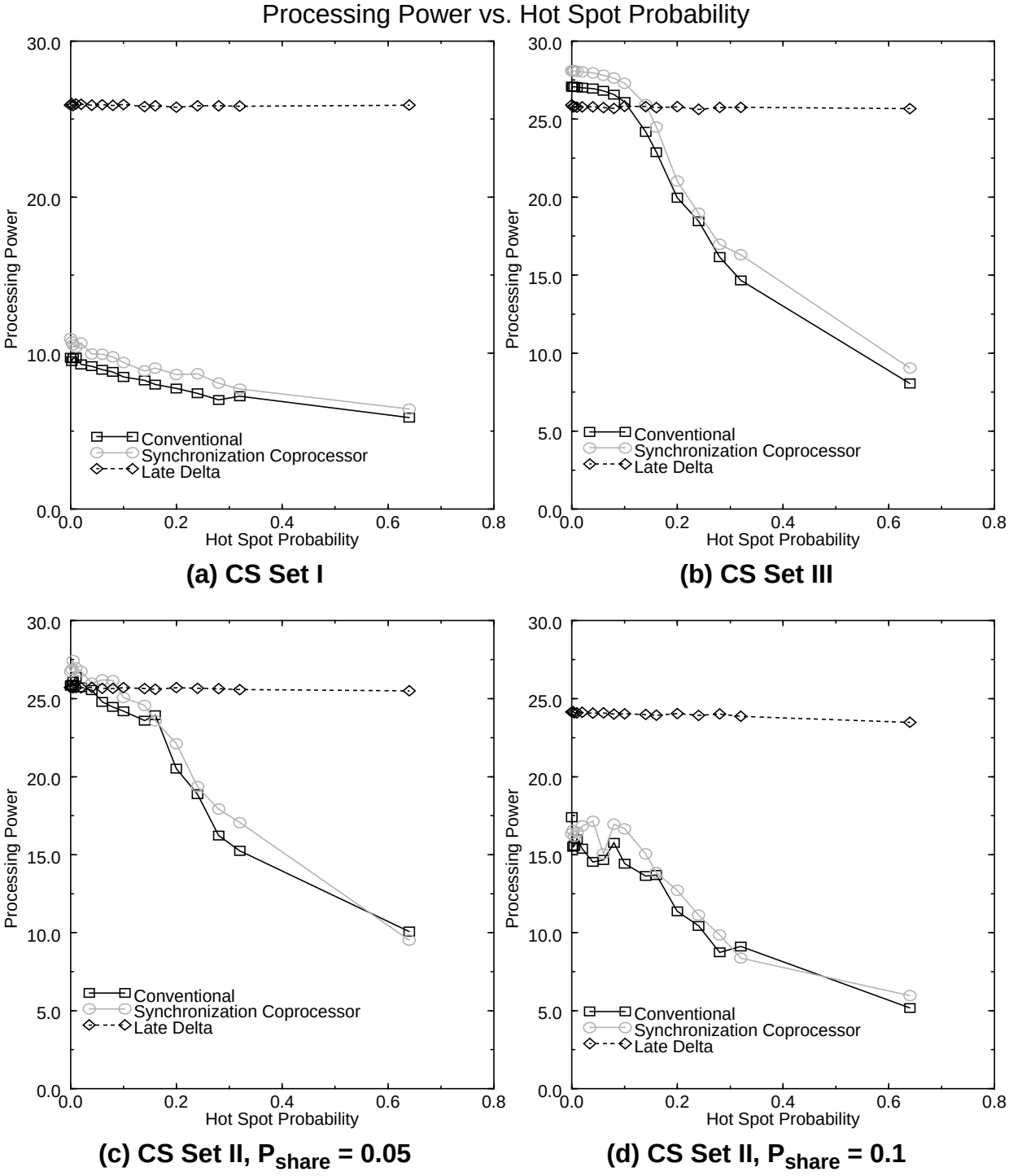
We consider multistage interconnection networks to support hardware multicast if they allow a single message to be sent to multiple destinations. In such a network, additional messages are created only when the multicast message is directed to destinations located on different outputs of the switch. The multicast facility is thus envisioned as following a fan out model. It should be clear that the various coherence protocols could all benefit from a multicast facility being supported by the network. The conventional protocol has two types of messages which can utilize a multicast facility. These are the invalidation messages themselves and the message which grants read access to all processors which have outstanding read requests to a block that had been held in exclusive mode. The late delta protocol is an update protocol and the update messages are typically directed to multiple destinations. Ultimately, nearly all messages which exercise the coherence protocols could utilize the multicast facility.

For the late delta protocol, software multicast causes a system-wide slowdown of logical time which implies that hot spots result in significant non-local performance degradation. Hardware multicast eliminates this non-local effect. Performance gains from the use of hardware multicast are limited for the conventional protocol since the effect of a software multicast is primarily local. The primary cause of reduced performance for the conventional protocol as the hot spot probability is increased is that writes require the invalidation of all other cache copies. Thus when contention for shared variables is high, as it is when the hot spot probability is large, most concurrent accesses to the same block must be performed sequentially. As the hot spot probability increases, the expected number of buffered operations to the block increases. Therefore, the number of network round trips which precede the completion of operations increases significantly. Hardware multicast does not address the cause of this problem and, therefore, does not reduce the expected number of round trips. The late delta protocol combined with hardware multicast provides consistent performance across workloads with a wide range of hot spot access characteristics. The rest of our results assume hardware multicast support for both systems.

4.1.2. Workloads with Flat Atomicity Requirements

To explore the benefits of the support for atomicity provided by the late delta protocol, we compare performance under workloads consisting of flat atomic actions. The late delta protocol provides support for flat atomic actions without the use of locks or other special synchronization constructs. For the conventional protocol, we used the super critical section locking scheme described previously to provide atomicity guarantees for operations to multiple shared variables. This scheme requires exactly one lock to be acquired for atomic access to any atomic action for which conflicting accesses may occur. Our results demonstrate that the late delta protocol provides excellent performance for workloads consisting of flat atomic actions while lock contention can severely limit the performance of the conventional protocol.

As stated previously, our performance metric is processing power. An important consideration is what processor cycles to count in computing this metric. A question is whether cycles used to issue lock operations should count. Technically, the processor is active



(a) CS Set I **(b) CS Set III**

(c) CS Set II, P_{share} = 0.05 **(d) CS Set II, P_{share} = 0.1**

Figure III: Processing Power with Flat Atomicity Requirements

during these cycles. However, the late delta protocol does not require these cycles to provide atomicity guarantees. With the conventional protocol, processors are not accomplishing useful work during these cycles. Instead, the cycles provide functionality which the late delta protocol demonstrates can be achieved in other ways. Additionally, the synchronization coprocessor relieves the processor of the burden of issuing lock release

operations. Since counting these cycles would hide important differences between the systems we are comparing, we have chosen not to do so.

We again compare the performance of the coherence protocols as a function of the probability of accessing a hot atomic action. Figure III shows the results obtained under workloads with flat atomicity requirements for the three sets of atomic actions described previously with a probability that any given reference is to shared data of 0.05. Accesses to lock variables are not included in this probability. The late delta protocol offers good performance across the full range of hot spot probabilities for all the sets of atomic actions. For CS Set I, we see that the connectedness of the critical section graph results in significant lock contention and therefore poor conventional protocol performance for all values of the hot spot probability. The results for CS Set II demonstrate the sensitivity of the conventional protocol to the probability of a shared access and so we also provide the results for this set with a probability of a shared access of 0.1. With a value of 0.05 for this probability and low hot spot probabilities, contention for the lock governing access to atomic actions in the large component of the critical section graph is low. Thus processors can complete their accesses to the atomic actions governed by this lock without degrading system performance. However, with the larger probability of a shared access, contention for this lock results in poor performance for all hot spot probabilities. CS Set III has 32 approximately equally sized components. Thus, for small values of the hot spot probability, lock contention with 32 processors is low and thus system performance is good. However, as the hot spot probability rises, most processors attempt to access the hot atomic action concurrently. Since lock acquisition sequentializes these accesses, performance again plummets. The late delta protocol does not suffer from these drawbacks. Since isotach logical time enforces sequential consistency and atomicity while allowing concurrent accesses including to the hot spot, we find the late delta protocol to be insensitive to the hot spot probability.

Conventional systems generally provide guarantees of sequential consistency by limiting the number of outstanding atomic actions to one per processor. However, the systems could assume that locks are released only when it is guaranteed that operations cannot be observed out of order. Making this assumption allows conventional systems to pipeline atomic actions in the network. We implemented conventional systems which allow pipelining to determine the benefits of this assumption. We observed that allowing pipelining increased lock contention, thus negating the benefits of the increased concurrency. Therefore, we do not present results from those experiments.

4.1.3. Workloads with Structured Atomicity Requirements

We expect a significant portion of the atomic actions of isotach programs will have internal data dependencies. To explore the effect of such dependencies, we modified our workload to create structured atomic actions as described previously. We have performed extensive simulations comparing the performance of the protocols under workloads with structured atomicity requirements. Our results demonstrate that allowing internal data dependencies in atomic actions does not significantly alter the performance of the protocols.

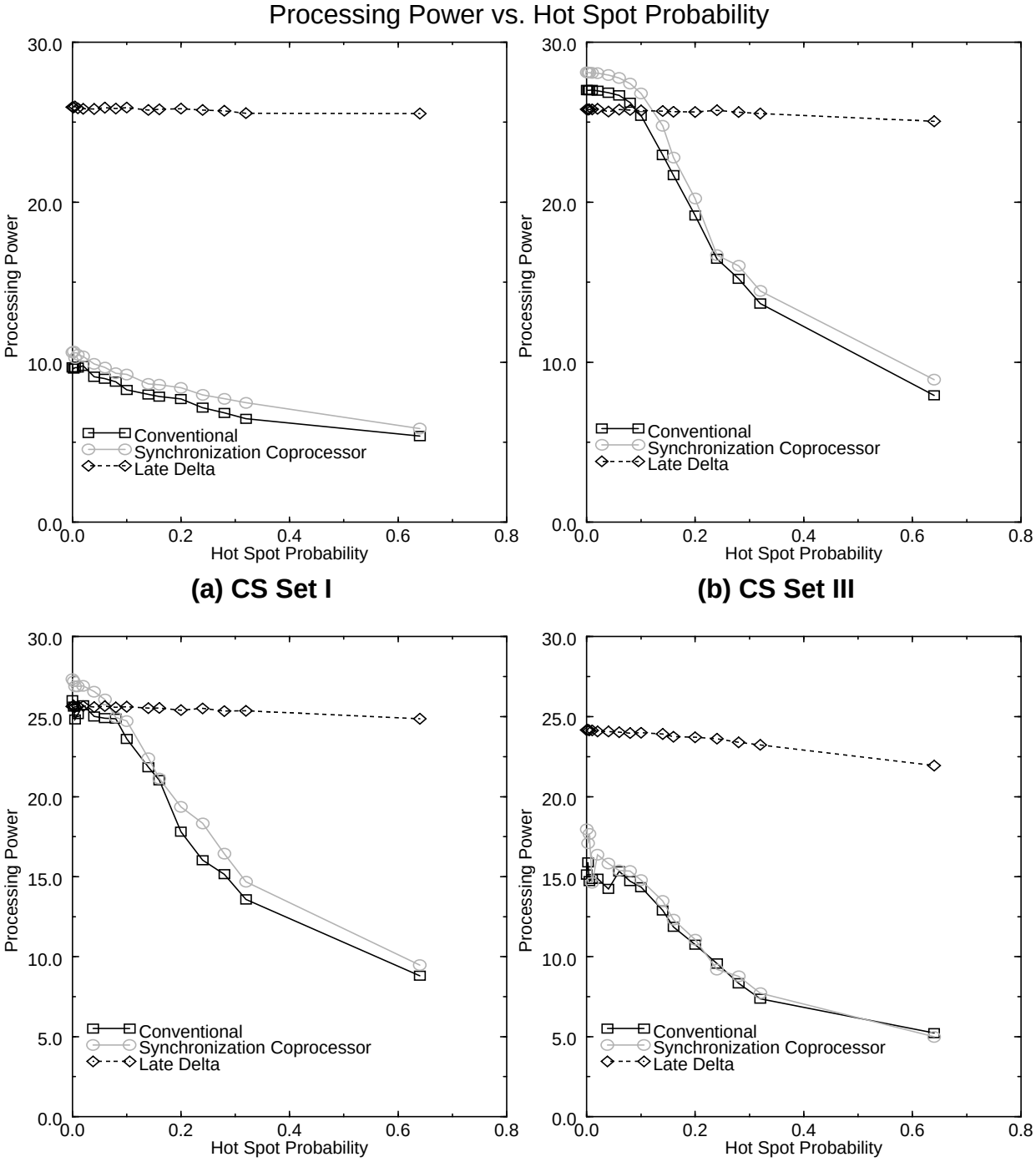


Figure IV: Processing Power with Structured Atomicity Requirements

For our structured atomicity experiments, we used a probability of 0.1 that a shared variable being written is also read as part of the same atomic action. We assume that there is always an internal data dependence on these reads. Additionally, given a read to a shared variable as part of an atomic action which contains at least one write to some other shared variable, all of our experiments use a probability of 0.5 that there is an inter-

nal data dependence on this read. If there is an internal data dependence on any read in an atomic action, we assume that all writes of that atomic action are dependent on it. These parameters resulted in approximately 40-50% of all shared atomic actions exhibiting internal data dependencies.

With the conventional protocol, executing structured atomic actions is nearly identical to executing flat atomic actions. With structured atomic actions, the processor must wait until reads for which there are internal data dependencies return before it issues the write operations. The conventional protocol could be modified to issue operations acquiring write access to the variables which will be written when the reads are issued. We have implemented this optimization although we have not yet tested it.

Allowing structured atomic actions under the late delta protocol required us to implement access sequences. Access sequences allow write operations to be split into two portions. One portion, the SCHED, schedules the write in logical time, while the other, the ASSIGn, actually provides the value of the write. We implement only structured atomic actions which employ the time slice technique. This technique schedules all accesses of the atomic action so that they all have the same effective execution pulse. Once all of the reads on which internal dependencies return with substantiated values, then the values for write operations can be computed and assigned. We have assumed a very basic implementation for issuing SCHED's and ASSIGN's. We assume that the processor must use two cycles to issue each SCHED/ASSIGN pair and that split operations are employed only for internal data dependencies. We do not count cycles used to issue SCHED operations in computing processor utilization for reasons similar to those which led us to discard locking cycles.

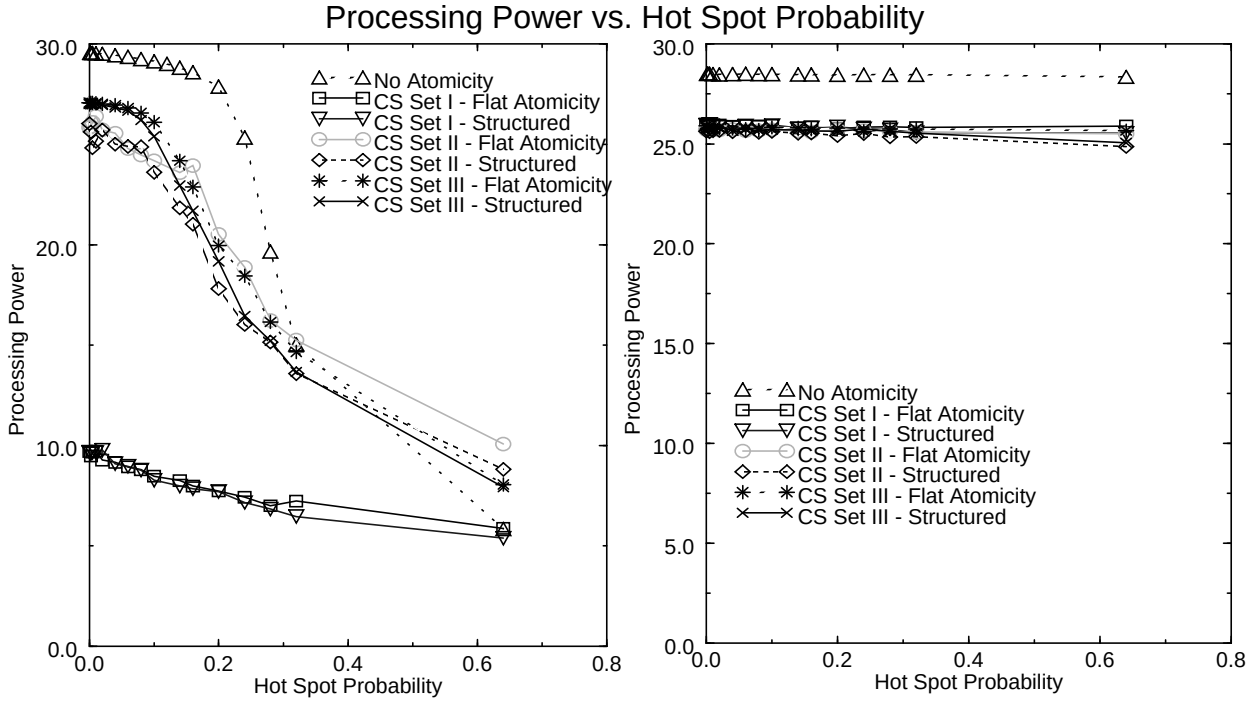
We present the results obtained for workloads with structured atomicity requirements in Figure IV. As expected, the results for the conventional protocol do not differ significantly from those of workloads with flat atomicity requirements. The addition of internal data dependencies does not significantly alter the lock contention exhibited by the workload. In examining the results for the late delta protocol, we again see consistent strong performance for all sets of atomic actions regardless of the hot spot probability. The existence of internal data dependences requires the scheduling of writes without providing the corresponding value. As the hot spot probability is increased, other processors become more likely to require these unsubstantiated values at the beginning of their think periods due to data dependencies between atomic actions, thus limiting the pipelining of atomic actions. There is some drop-off in system performance for large values of the hot spot probability with the late delta protocol for CS Set II with a probability of a shared access of 0.1. However, this effect is limited as the protocol combined with access sequences limits the restriction on pipelining to true data dependencies. We observe that the late delta protocol allows the extraction of a higher degree of parallelism from these workloads than the conventional protocol.

4.1.4. Another Look: Varying the Atomicity Requirements

We have presented our results comparing the performance of the protocols for a given workload. An interesting question is how a given protocol performs while varying the workload. Figure V summarizes these results. Our results demonstrate that the late delta protocol can offer good consistent performance, while the conventional protocol is extremely sensitive to both hot spot probability and atomicity requirements.

We fi nd that the conventional pɒtocol offers consistently poor performance when the hot spot probability is large. Careful coding may limit hot spot accesses in systems based on this protocol. The differences in performance exhibited for the three sets of atomic actions when the hot spot probability is low demonstrate that organizing data accesses to use a large number of equally likely to be accessed locks is also essential.

The results for the late delta protocol are excellent. Not only is performance good under any of the workloads, there is little variation between them. We theorize that performance of the late delta protocol is primarily dependent on the mean and variance of the size of atomic actions. The performance of the late delta protocol under workloads with no atomicity requirements is about ten percent higher than under those with atomicity requirements. The no atomicity workloads can be viewed as workloads of fl at atomic actions of one operation. We expect larger atomic actions to decrease the rate of logical time with respect to real time, thus reducing performance. However, we expect small atomic actions in most workloads and thus good system performance regardless of the hot spot characteristics of the application.



(a) Conventional Protocol **(b) Late Delta Protocol**

Figure V: Affect of Atomicity Requirements within Protocol

4.1.5. The Effect of Increasing System Size

The increased latency of isotach networks compared to conventional networks is an important concern for delta cache systems. The results with 32 processors and no atomicity requirements indicate that the pipelining of memory accesses and the increased concurrency of the late delta protocol adequately compensate for this latency under small systems even with workloads which do not exploit all of the features of the protocol, such as atomicity support. However, the difference in network latencies increases with increases in network size. Therefore, we compared the performance of the cache coherence protocols as a function of hot spot probability in systems with 128 processors under workloads with no atomicity requirements. Increasing the number of processors in the system increases the amount of contention for shared variables created by a given hot spot probability. Thus, the difficulties with contention for shared variables in systems based on the conventional protocol become more important as system size is increased. The late delta protocol begins to exhibit some sensitivity to contention for shared variables for large values of the hot spot probability, but still yields excellent performance.

In Figure VI, we show the results obtained with larger systems for probabilities of a shared access of 0.05 and 0.1. The late delta protocol continues to offer excellent performance with significant degradation only for the highest value of the probability of accessing the hot spot and a probability of a shared access of 0.1. These results demonstrate the resiliency of the late delta protocol and the unlikelihood of saturating the hot spot memory module. Additionally, we find that the hot spot probability for which the

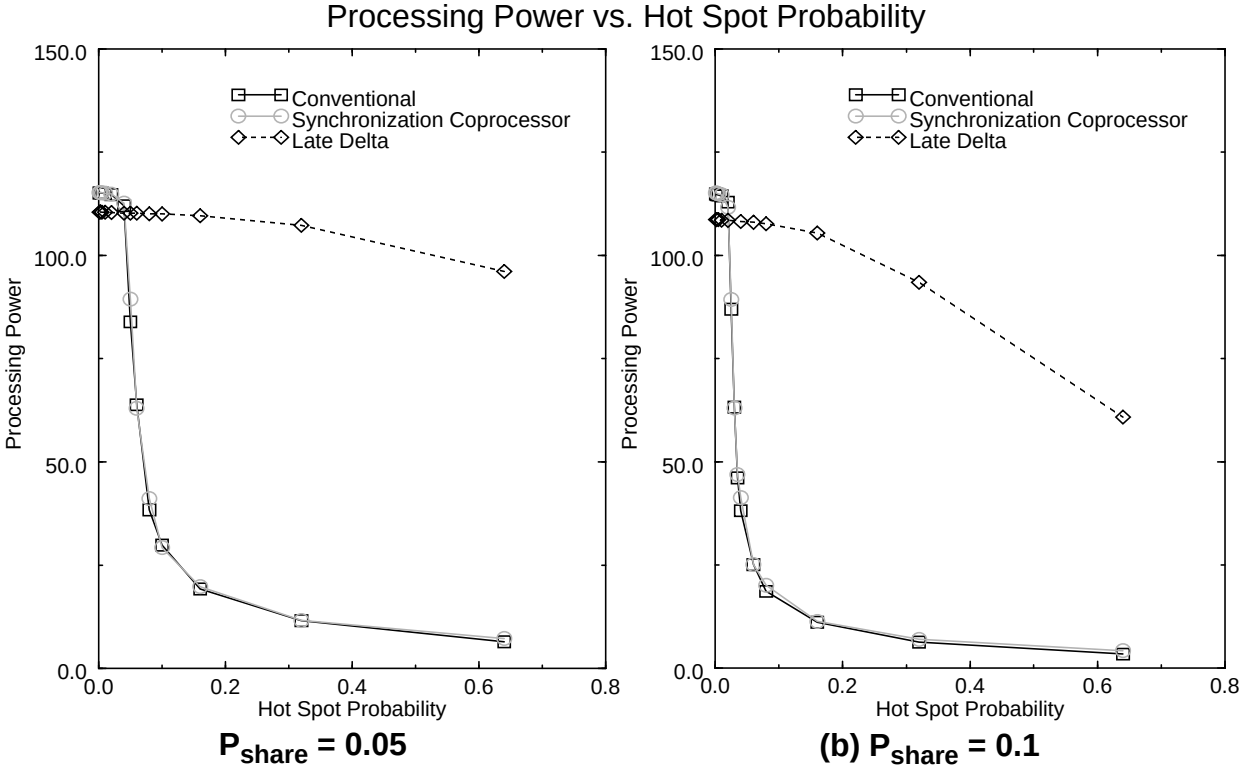


Figure VI: Processing Power with 128 Processors

conventional protocol is saturated is decreased signifi cantly. We fi nd that the conventional protocol cannot provide adequate performance in large scale systems.

We have demonstrated that the late delta protocol offers good performance across a wide range of the hot spot probabilities. For our remaining results, we fi x this probability at 0.1. Other than for CS Set I, the conventional protocol offers reasonable performance at this value. For CS Set I, we found that the data layout was such that the conventional protocol could never offer good performance. The remaining sections will explore the performance of the protocols when two important system characteristics are varied - system size and the probability of a shared access.

4.2. Scalability

In this section, we extend our study of the scalability of the protocols. We compare the performance of the protocols under workloads with no atomicity requirements and with structured atomicity requirements. We expect the conventional protocol to perform best in relation to the late delta protocol under the no atomicity workloads since these workloads do not offer the late delta protocol the opportunity to use its effi cient techniques for enforcing atomicity.

The results of an initial study exploring the scalability of the protocols are presented in Figure VII. For all of these curves, the probability that any given reference is to shared data is 0.05. We include a curve showing the theoretical maximum performance corresponding to a processor utilization of one for all processors in the system. With a small number of processors, we fi nd the performance of these protocols is nearly indistinguishable for each of the workloads. For the workloads with structured atomicity requirements, we see that lock contention increases as the number of processors is increased. The late delta protocol continues to provide good performance as the system size is increased. The performance gain in going from 64 to 128 processors for the conventional systems for CS Set III cannot justify the use of the additional processors, while performance has declined on the same interval for the other sets of atomic actions.

The surprising result appears under the workload with no atomicity requirements. Under this workload, we had expected the conventional protocol to slightly outperform the late delta protocol for all system sizes. However, with a hot spot probability of 0.1 and probability of a shared access of 0.05, we fi nd that for the conventional protocol increasing the system from 64 to 128 processors actually decreases processing power. Thus the late delta protocol, which demonstrates far greater resiliency to hot spot accesses, signifi cantly outperforms the conventional systems under a workload which does not exploit the support for atomicity offered by isotach systems.

We observe that the performance of the conventional protocol is actually better with structured atomicity requirements and the atomic actions of CS Set III than with no atomicity requirements. The primary cause for this apparent anomaly is that the synchronization memory is implemented at cache memory speeds while the directory is

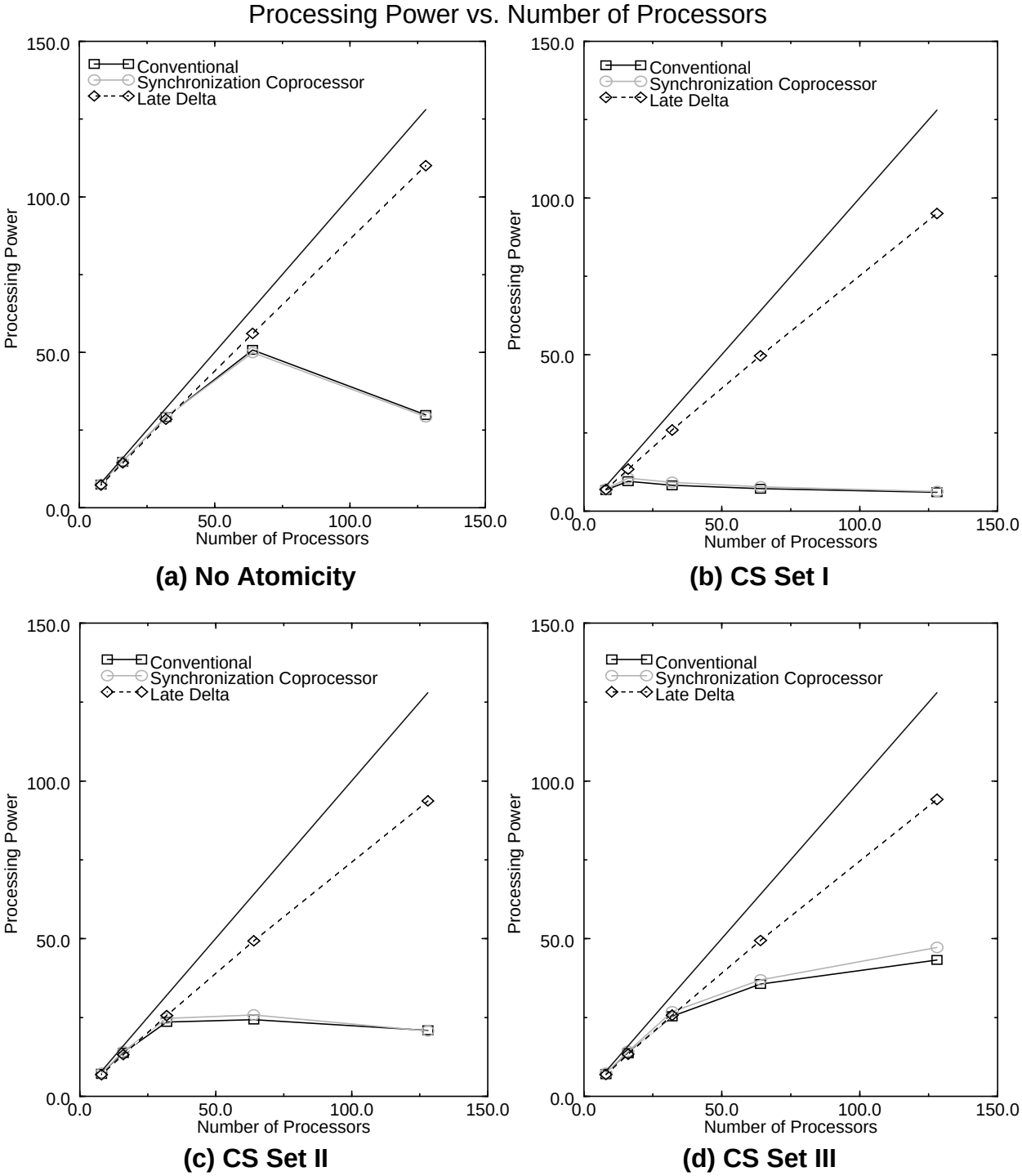


Figure VII: Varying the Number of Processors

implemented at main memory speeds. Further, the lock protocol is optimized for the single type of operation to lock memory, whereas the conventional coherence protocol must support both reads and writes. Since CS Set III has 32 large, approximately equally sized components, lock requests which are not for the hot atomic action do not signifi cantly increase contention. Thus when approximately equal amounts of contention are limiting

performance for both the no atomicity and atomicity workloads, the faster synchronization memory combined with the efficient locking protocol can yield improved performance. However, that this effect is not observed with CS Sets I and II where the existence of one lock governing a significant portion of the shared atomic actions results in significantly higher contention from the same parameters.

The data layout of CS Set III could be viewed as optimizing the application being modelled for a system with 32 processors. It might be possible to recode the application for the larger system, yielding 128 large components. Then we would expect the conventional protocol again to provide good performance. However, this process is likely to be difficult and time consuming. Further, despite there being more parallelism inherent in the problem it is not clear that this system size specific targeting is always possible. Most importantly, this exercise is unnecessary for the late delta protocol.

4.3. Varying the Probability of a Shared Access

Studies of reference patterns in shared memory multiprocessor indicate that the probability of any given reference being to shared data varies greatly among applications [BaR89, DPS86]. Although this probability is low for most applications, it can be as high as 0.25 or even 0.5 for some applications. Therefore, protocols which can provide consistently good performance across a range of values for this probability are desirable.

With our workload model, there are two effects of increasing the probability of a shared access which may reduce system performance. Since the shared hit ratio is typically lower than the private hit ratio, increasing the proportion of shared accesses increases the proportion of accesses that miss in the cache. This is particularly true for the late delta protocol, since all writes to shared data must be sent to the memory module, which distributes the updates required by the protocol. Since we count only operations which are performed on the cache copy as cache hits, the late delta protocol has a maximum shared hit ratio of 0.7 with a shared write probability of 0.3. This first performance degrading effect of additional use of the network includes increased memory module utilization, which we expect to be uniformly distributed across the memory modules. The second effect involves hot spot accesses. When the probability of a shared access is increased, the probability that any given reference is to the hot spot is increased a proportionate amount. Thus, there is hot spot effect as well as a system-wide effect from increasing the probability of a shared access.

We have conducted a set of experiments to investigate the performance of the conventional protocol and the late delta protocol as a function of the probability of a shared access. The results are presented in Figure VIII. In these experiments, we kept the hot spot probability constant at 0.1. We observe that the late delta protocol offers greater resiliency under this parameter. Even under workloads with no atomicity requirements, the conventional systems cannot provide good performance when accesses to shared data are frequent. Recalling the results from the experiments under which the hot spot probability was varied, we hypothesize that the hot spot effect dominates the performance of these systems when the probability of a shared access is high.

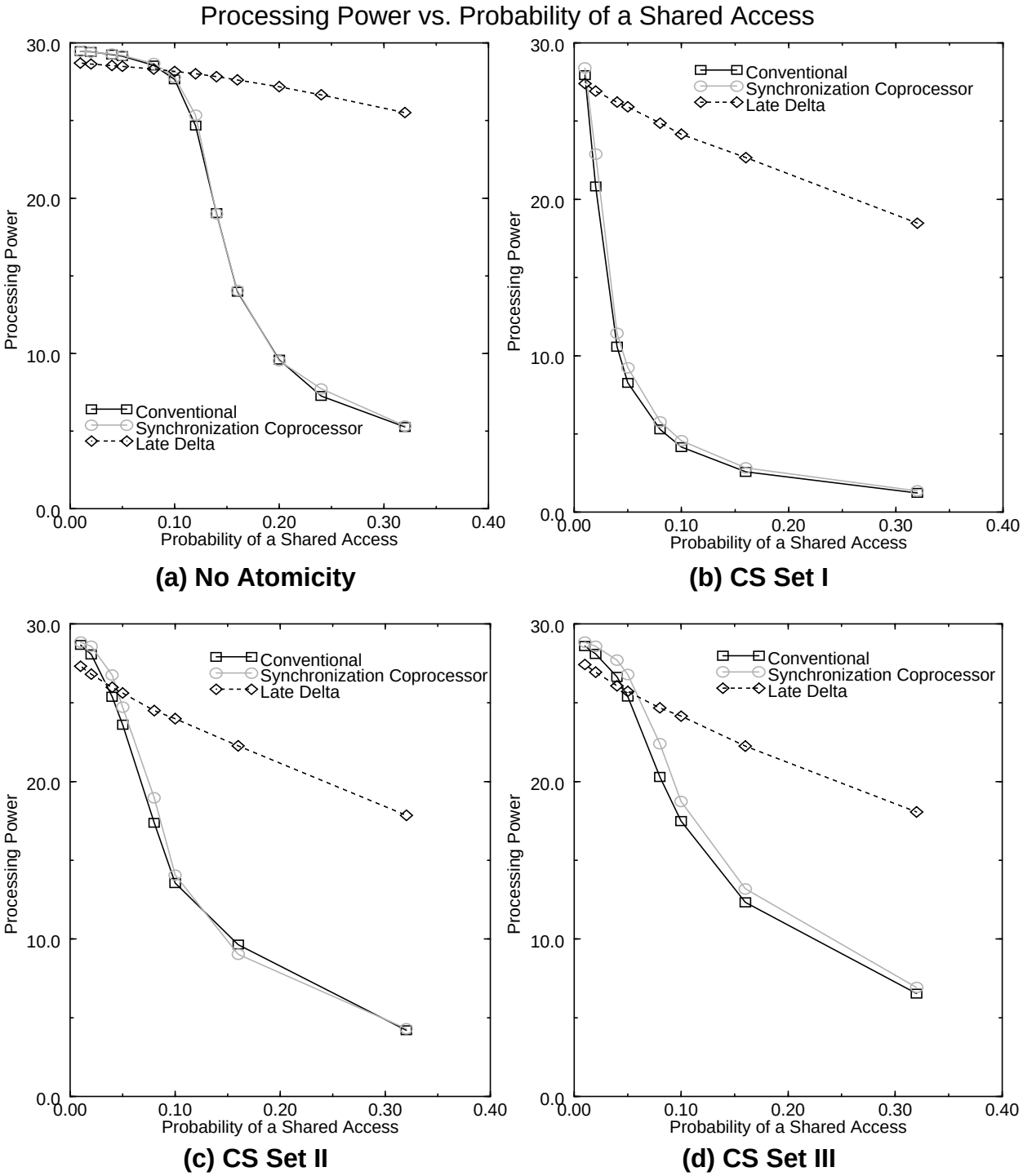


Figure VIII: Varying the Probability of a Shared Access

The performance of the late delta protocol shows far greater sensitivity to the probability of a shared access than to the hot spot probability. Recall that the performance of the late delta protocol was nearly unaffected when the hot spot probability was increased. Therefore, it is clear that the system-wide effect of an increased probability of a shared access signifi cantly afects the performance of the late delta protocol. We are currently consider-

ing system design changes which will alleviate this problem. Nonetheless, the late delta protocol significantly outperforms the conventional protocol under workloads in which the proportion of shared accesses is high.

5. Conclusion

Delta cache coherence protocols are a highly concurrent family of directory based protocols. These protocols allow pipelining of shared accesses and provide support for sequential consistency and atomicity without the use of locks. These features of the protocol family are expected to result in consistently good performance, even for workloads with atomicity requirements.

We have compared the performance of the late delta protocol, an update based member of this protocol family, to that of a conventional invalidation protocol. An important result of our study is that hardware multicast enhances the performance of the late delta protocol. We expect this result to be true for all delta cache coherence protocols. We have demonstrated that the late delta protocol has significantly greater resiliency to contention for shared variables. This resiliency to contention for shared variables has important scalability implications. We have demonstrated that workloads which include atomicity requirements have little effect on the performance of the late delta protocol while the conventional protocol is severely hampered by these requirements.

We have augmented the conventional protocol with a synchronization coprocessor in an effort to provide the benefits of the delta cache family of protocols to the conventional protocol. Our study indicates that this coprocessor can provide small performance benefits to the conventional protocol. However, it does not alter lock contention for workloads with atomicity requirements. Lock contention can be addressed through careful coding and data layout. However, this is a time consuming process which yields performance for a fixed number of processors. The late delta protocol does not suffer from these drawbacks since its atomicity support eliminates the use of locks. Our study demonstrates that the delta cache family of protocols is a promising technology which can significantly improve performance of shared memory systems.

6. References

[AgG88] Agarwal, A. and A. Gupta, "Memory Reference Characteristics of Multiprocessor Applications under MACH," *Proceedings of the 1988 Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 215-225, 1988.

[ASH88] Agarwal, A., R. Simoni, J. Hennessy and M. Horowitz, "An Evaluation of Directory Schemes for Cache Coherence," *Proceedings of the 15th International Symposium on Computer Architecture*, pp. 280-289, 1988.

[ADT90] Algudady, M.S., C.R. Das and M.J. Thazhuthaveetil, "A Write Update Cache Coherence Protocol for MIN-Based Multiprocessors with Accessibility Based Split Caches," *Proceedings of Supercomputing '90*, pp. 544-553, 1990.

[ArB86] Archibald, J. and J.L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Transactions on Computer Systems*, Vol. 4, No. 4, pp. 273-298, 1986.

[BaR89] Baylor, S.J. and B.D. Rathi, "A Study of the Memory Reference Behavior of Engineering/Scientific Applications in Parallel Processors," *Proceedings of the 1989 International Conference on Parallel Processing*, pp. I-78-I-82, 1989.

[BMR89] Baylor, S.J., K.P. McAuliffe and B.D. Rathi, "Cache Coherence Protocols for MIN-Based Multiprocessors,"RC15221, IBM Research Report, 1989.

[BLA89] Bhuyan, L.N., B.C. Liu and I. Ahmed, "Analysis of MIN based Multiprocessors with Private Cache Memories," *Proceedings of the 1989 International Conference on Parallel Processing*, pp. I-51-I-58, 1989.

[CeF78] Censier, L.M. and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Transactions on Computers*, Vol. 27, pp. 1112-1118, 1978.

[CKA91] Chaiken, D., J. Kubiatowicz and A. Agarwal, "LimitLESS Directories: A Scalable Cache Coherence Scheme," *Proceedings of the 4th ASPLOS*, pp. 224-234, 1991.

[CDK94] Cox, A.L., S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony and W. Zwaenepoel, "Software Versus Hardware Shared Memory Implementation: A Case Study," *Proceedings of the 21th International Symposium on Computer Architecture*, pp. 106-117, 1994.

[DPS86] Darema-Rogers, F., G.F. Pfister and K. So, "Memory Access Patterns of Parallel Scientific Programs,"RC12086, IBM Research Report, 1986.

[DGH90] Davis, H., S.R. Goldschmidt and J. Hennessy, "Tango: A Multiprocessor Simulation and Tracing System,"Tech. Rep. CSL-TR90-439 Stanford University, Computer Systems Laboratory, 1990.

[DuB82] Dubois, M. and F.A. Briggs, "Effects of Cache Coherency in Multiprocessors," *IEEE Transactions on Computers*, Vol. 31, pp. 1083-1099, 1982.

[DKC93] Dwarkadas, S., P. Keleher, A.L. Cox and W. Zwaenepoel, "Evaluation of Release Consistent Software Distributed Shared Memory on Emerging Network Technology," *Proceedings of the 20th International Symposium on Computer Architecture*, pp. 106-117, 1993.

[EgK88] Eggers, S.J. and R.H. Katz, "Evaluating the Performance of Four Snooping Cache Coherency Protocols," *Proceedings of the 15th International Symposium on Computer Architecture*, pp. 373-382, 1988.

[Lam79] Lamport, L., "How to Make a Multiprocessor Computer that Correctly Executes Multiprocessor Programs," *IEEE Transactions on Computers*, Vol. 28, pp. 690-691, 1979.

[LaK92] Law, A.M. and W.D. Kelton, Simulation Modeling and Analysis, McGraw-Hill, Inc., 1992.

[LeR90] Lee, J. and U. Ramachandran, "Synchronization with Multiprocessor Caches," *Proceedings of the 17th International Symposium on Computer Architecture*, pp. 27-37, 1990.

[MiB92] Min, S.L. and J.L. Baer, "Design and Analysis of a Scalable Cache Coherence Scheme Based on Clocks and Timestamps," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 1, pp.25-44, 1992.

[MBK90] Min, S.L., J.L. Baer and H.J. Kim, "An Efficient Caching Support for Critical Sections in Large-Scale Shared-Memory Multiprocessors,"RC1531 1, IBM Research Report, 1990.

[NaB93] Nanda, A.K. and Bhuyan, L.N., "Design and Analysis of Cache Coherent Multistage Interconnection Networks," *IEEE Transactions on Computers*, Vol. 42, No. 4, pp. 458-470, 1993.

[PfN85] Pfister, G.F. and V.A. Norton, "'Hot Spot' Contention and Combining in Multistage Interconnection Networks," *IEEE Transactions on Computers*, Vol. 34, No. 10, pp. 943-948, 1985.

[RWW92] Reynolds, Jr., P.F., C. Williams and R.R. Wagner, Jr., "Empirical Analysis of Isotach Networks,"Tech. Rep. 92-19, University of Virginia, Department of Computer Science, 1992.

[Spi77] Spirn, J.R., Program Behavior: Models and Measurements, Elsevier North-Holland, Inc., 1977.

[VLZ88] Vernon, M.K., E.D. Lazowska and J. Zahorjan, "An Accurate and Efficient Performance Analysis Technique for Snooping Cache-Consistency Protocols," *Proceedings of the 15th International SYNP Computer Architecture*, pp. 308-315, 1988.

[Wil93] Williams, C., Concurrency Control in Asynchronous Computations, Ph.D. Thesis, University of Virginia, 1993.

[WiR90] Williams, C. and P.F. Reynolds, Jr., "Delta-Cache Protocols: A New Class of Cache Coherence Protocols,"Tech. Rep. 90-34, University of Virginia, Department of Computer Science, 1990.

[WiL92] Wilson, A.W. and R.P. LaRowe, Jr., " Hiding Shared Memory Reference Latency on the Galactica Net Distributed Shared Memory Architecture," *Journal of Parallel and Distributed Computing*, Vol. 15, No. 4, pp. 351-367, 1992.

[WLT93] Wilson, A.W., R.P. LaRowe, Jr. and M.J. Teller, " Hardware Assist for Distributed Shared Memory," *Proceedings of the 13th International Conference on Distributed Computing Systems*, pp. 246-255, 1993.

[YBL89] Yang, Q., L.N. Bhuyan and B.C. Liu, " Analysis and Comparison of Cache Coherence Protocols for a Packet-Switched Multiprocessor," *IEEE Transactions on Computers*, Vol. 38, No. 8, pp. 1143-1153, 1989.