EVALUATING DESIGN CHOICES FOR SHARED BUS MULTIPROCESSORS IN A THROUGHPUT-ORIENTED ENVIRONMENT

by

M.-C. Chiang
G. S. Sohi

Computer Sciences Technical Report #927

April 1990

Abstract

This paper considers the evaluation of design choices in multiprocessors with a single, shared bus interconnect operating in a *throughput-oriented*, multiprogrammed environment, that is, an environment in which each task is being executed on a single processor and the performance of the multiprocessor is measured by its overall throughput. To evaluate design choices, we develop mean value analysis analytical models and validate our models by comparing their results against the results of a trace-driven simulation analysis for 5,376 multiprocessor configurations. The trace-driven simulation uses actual programs and simulates their execution in a throughput-oriented environment.

Using multiprocessor throughput as a performance metric and the mean value analysis models as tools, we evaluate several design choices. We find that: i) cache block sizes that yield the best performance in a multiprocessor differ from the block sizes that yield the best uniprocessor performance metrics, ii) a larger cache set associativity might be warranted in a multiprocessor even though it might not be warranted in a uniprocessor iii) a split transaction, pipelined bus yields much higher multiprocessor throughput than a circuit switched bus, especially for larger main memory latencies, and iv) increasing the bus width appears to be an effective way of improving multiprocessor throughput.

Keywords

Shared bus multiprocessors, mean value analysis, trace-driven simulation, cache block size, cache set associativity, circuit switched buses, split transaction pipelined buses

1. INTRODUCTION

Low-cost microprocessors have led to the construction of small- to medium-scale shared memory multiprocessors with a shared bus interconnect. Such multiprocessors, which have been referred to a *multis* by Bell [1], are popular for two reasons: i) the shared bus interconnect is easy to implement and ii) the shared bus interconnect allows an easy solution to the cache coherence problem [2]. Currently, many major computer manufacturers have a commercial product or a research project that uses the multi paradigm.

A typical shared bus, shared memory multiprocessor (hereafter called a multi in this paper) is shown in Figure 1. The multi consists of several processors (typically microprocessors) connected together to a memory system. The memory system includes the private caches of each processor, the shared bus interconnect, and the main memory. The overall performance of such a multi is heavily influenced by the design of the memory system. Starting with processors at a particular performance level, to design a multi with a desired performance level, the multi designer must provide an adequate-performance memory system.

To design a memory system with an adequate performance, the designer must have access to sophisticated performance evaluation tools. These tools can vary from analytical models to detailed trace-driven simulation. Analytical models are computationally much cheaper than trace-driven simulation and allow a much larger design space to be explored. However, they are generally considered to be less accurate than trace-driven simulation.

In this paper, we consider a methodology for evaluating design choices in the memory system of multis operating in a *throughput-oriented* multiprogramming environment, and use the methodology to evaluate key design choices in such an environment. By a throughput-oriented multiprogramming environment, we mean an environment in which each task is being executed on a single processor and the

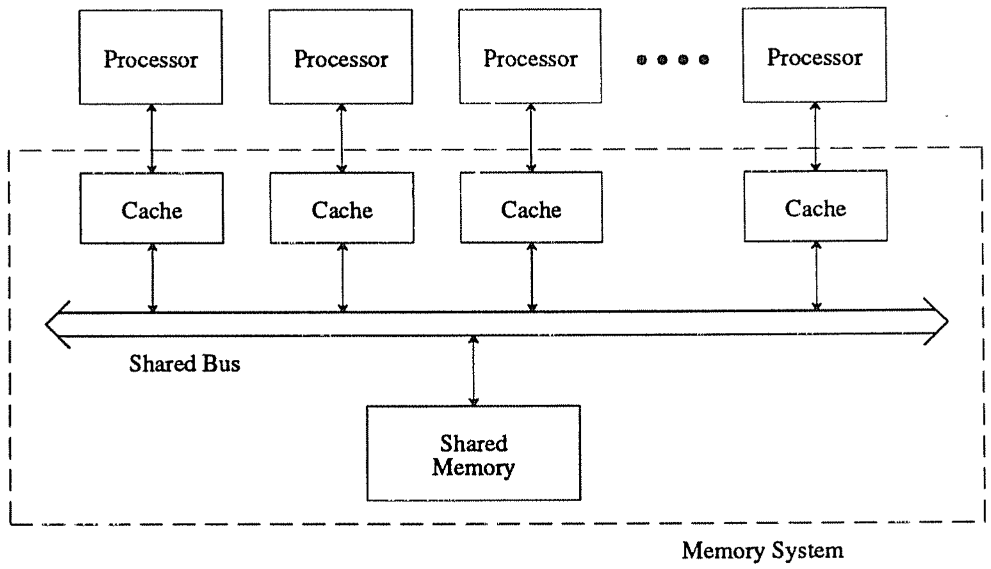


Figure 1: A Shared Bus Multiprocessor (Multi)

performance of the system is measured by the overall throughput of the multiprocessor[1].

At the heart of our methodology are mean value analysis (MVA) analytical models. We develop MVA models of a multi and compare the results of the MVA models to actual trace driven simulation for over 5000 configurations. Having validated the models, we use them to evaluate key design choices in the memory system.

The remainder of this paper is as follows. In section 2, we discuss the memory system model of the multi and present some of the design choices that we consider. In section 3, we discuss the MVA models and the trace-driven simulation setup and summarize the results of a detailed analysis comparing the results of the MVA models with those of trace-driven simulation. In section 4, we use the validated MVA models to evaluate some key design choices and in section 5 we present concluding remarks.

2. MEMORY SYSTEM MODEL AND DESIGN CHOICES

As mentioned earlier, the memory system of a typical multi consists of three main components: i) the private caches of each processor (which may be multi-level), ii) the shared bus interconnect and iii) the shared main memory. In such a memory system, the average memory access time (or latency) seen by a processor, T_m^P, is:

$$T_m^P = T_C^P + M \times T_m^C \qquad (1)$$

where

- T_C^P is the cache access time.
- M is the cache miss ratio.
- T_m^C is the average time taken to service a cache miss.

Equation (1) is applicable in general to any processing system (uniprocessor or multiprocessor) with a cache and a memory. However, the components of the equation are variable and depend upon the parameters of the memory system. For example, T_C^P is a function only of the cache organization. Likewise, M is also a function only of the cache organization and is not dependent on other parameters of the memory system. However, T_m^C can be a function of several parameters of the memory system, such as the characteristics of the shared bus and the main memory.

A major difference between the design of a memory system for a uniprocessor and a multiprocessor operating in a throughput-oriented environment is in the impact of T_m^C. In a uniprocessor, T_m^C can be approximated as $T_m^C = \alpha + \beta \cdot B$, where B is the cache block size and α and β are constants that represent the fixed overhead and the unit transfer cost of transferring a cache block [3].

In a multi, T_m^C cannot be approximated as $T_m^C = \alpha + \beta \cdot B$. This is because T_m^C includes a queuing delay that can have a significant overall contribution to T_m^P. This queuing delay is dependent upon the utilization of the bus which in turn is dependent upon several system-wide characteristics such as: i) the traffic on the bus, which is influenced by the number of processors connected to the bus, and the organizations of their caches, ii) the bus switching strategy and iii) main memory latency. If accurate results are to be obtained for design choices in shared bus multiprocessor memory systems, all system factors and their complex interdependencies must be taken into account.

Before proceeding further, let us consider some design choices in the three main components of the memory system and see how they might influence one another. A comprehensive evaluation of design choices is not possible in this paper (but can be carried out with our methodology) and we focus our

[1]The most popular multis on the market today, those designed by Sequent Computer Systems, are designed for such an environment.

attention on some key design choices for throughput oriented multiprocessors[2].

2.1. Cache Memory

The key component of the memory system is the cache and most of the issues are concerned with how the choice of cache parameters influences the design of the other components and vice-versa. We consider three important cache parameters: i) the cache size, ii) the cache block size and iii) the cache set associativity.

A large cache is able to lower T_m^P directly by lowering M.[3] Furthermore, because of lower bus traffic (assuming a constant number of processors N), the utilization of the bus and the queuing delay for a bus request is lowered and consequently T_m^C is reduced. Alternately, a lower per-processor bus utilization allows more processors to be connected together on the bus, possibly increasing the peak throughput (or processing power) of the multi.

While it is clear that larger caches allow more processors to be connected together in a throughput-oriented environment by reducing per-processor bus bandwidth demands and improving T_m^P, the relationship between the cache size and other memory system parameters (such as the block size, the set associativity, the bus switching strategy, bus width and the main memory latency) needs to be investigated and the improvement quantified.

Cache block size is perhaps the most important parameter in the design of each cache. The block size not only dictates the performance but also the implementation cost of the cache (smaller block sizes result in a larger tag memory than larger block sizes). In a detailed study, Smith mentions several architectural factors that influence the choice of a block size and evaluates them in a uniprocessor environment [3]. Smith's main result that is of interest to us in this paper is that the miss ratio decreases with increasing block size up to a point at which internal cache interference increases the miss ratio. However, larger block sizes also cause more *traffic* on the cache-memory interconnect. Since this additional traffic uses up more bus bandwidth and since the bandwidth of the shared bus is the critical resource in a multi, Goodman has suggested that small block sizes are preferable for cache memories in multis [2].

As Smith points out, minimizing the bus traffic alone is not the correct optimization procedure in multiprocessors and neither is a minimization of the miss ratio, independent of the other parameters of the memory system [3]. This point, which is also apparent from equation (1), is central to the topic of this paper and cannot be overemphasized. If maximizing multiprocessor system throughput is the goal, the choice of the block size should not be decoupled from the parameters of the shared bus and the main memory. Furthermore, any evaluation must consider equation (1) in its complete generality and include not only the main memory latency and the bus transfer time of a request, but also the queuing delays experienced by the memory request.

Cache set associativity is also an important design consideration. It is well known that a larger set associativity reduces M (in most cases) and, if T_C^P and T_m^C are constant, a larger set associativity is preferable, subject to implementation constraints [5]. However, as several researchers have observed, T_C^P is not independent of the cache set associativity since a large set associativity requires a more complex implementation and consequently has a higher T_C^P. If the decrease in T_C^P by going to a lower set associativity is greater than the increase in $M \times T_m^C$, then a lower set associativity results in a lower overall T_m^P, and consequently a higher processor throughput.

[2]For the remainder of this paper, all references to multis shall assume a multi operating in a throughput-oriented environment, unless specifically mentioned otherwise.

[3]When a multi is executing parallel programs, M can be variable even for a fixed size cache. The value of M depends on the number of processors on which the parallel program is executing, as a result of a phenomenon called *reference spreading* [4]. Consequently, a larger cache may not necessarily be able to lower the value of M. However, for a throughput-oriented multi, there is no reference spreading and a larger cache will result in a lower value of M.

In a uniprocessor, T_m^C is a constant for a given block size and memory configuration. If M is sufficiently small (because of a large cache size, for example), the decrease in T_C^P due to a lower set associativity can easily overcome an increase in $M \times T_m^C$ [6-8]. In a multiprocessor, however, T_m^C contains a queuing delay, which can be a large fraction of T_m^C if the bus utilization is high. Increasing cache set associativity not only decreases T_m^P directly by reducing M, it also reduces T_m^P indirectly by reducing the utilization of the bus and consequently the queuing delay component of T_m^C. Therefore, the impact of cache set associativity on multiprocessor memory system design is another important design issue that needs to be investigated.

2.2. Shared Bus

The main design issues in the shared bus are the choice of the bus width and the bus switching strategy (or protocol). Using a wider bus is an effective way to increase the bus bandwidth. Increasing bus bandwidth reduces T_m^C in two ways: directly by reducing the bus transfer time of a block, and indirectly by reducing the bus queuing delay due to the decreased bus tenure of each memory request. However, increasing the bus width affects the design of other modules. For example, the bandwidth of the cache and main memory should match that of the bus. This implies that the block size of the cache and main memory should be made at least as large as the bus width.

Bus switching methods fall into two broad categories: i) *circuit switched* buses and ii) *split transaction, pipelined* buses (hereafter referred to as STP buses in this paper). In a circuit switched bus, the bus is held by the bus master until the entire transaction is complete. The time that the bus is held by the master (or the bus tenure) includes the latency of the slave device. Such a switching strategy is used in most existing bus designs. For example, the block read and block write transactions in the IEEE Futurebus employ a circuit switched protocol [9].

In an STP bus, the bus is not held by the master if the slave device is unable to respond to the request immediately. The bus is released by the master and is made available to other bus requesters. When the slave device is ready to respond to a request, it obtains bus mastership and transfers data to the requesting device. An STP bus is used in the Sequent Balance and Symmetry multiprocessors [10, 11], and is also being considered for the IEEE Futurebus+.

2.3. Main Memory

The final component of the multiprocessor memory system is the main memory and the parameter of importance is the main memory latency. Many studies choose to ignore this parameter (or assume that it is a constant). As we shall see, including main memory latency is crucial since it influences other memory system design parameters such as the cache block size, especially with a circuit switched bus.

3. PERFORMANCE EVALUATION METHODS

For evaluating design choices, the favorite tool of a computer architect is trace-driven simulation using traces generated by the actual execution of sample benchmark programs (we call this an *actual* trace-driven simulation in this paper). Unfortunately, trace-driven simulation is expensive, both in execution time and storage requirements (required to store the traces). The storage expense of actual trace-driven simulation can be reduced by parameterized trace driven simulation. In parametrized simulation, artificial traces are generated on the fly using probability distributions that have the same characteristics as the actual program traces. Parameterized simulation is still computationally expensive and is generally not considered to be as accurate as actual trace-driven simulation. Finally, one can develop an analytical model. Iterative solutions of analytical models generally are much cheaper computationally than trace-driven simulation and consequently allow the designer to explore a much larger design space.

Multiprocessors with arbitrary interconnection networks have been the subject of several previous studies [12-16]. Studies of bus-based multiprocessor design issues have used trace-driven simulation [17], parameterized simulation [18], as well as analytical modeling [19, 20]. For a system as complex as a multi, ideally a system designer would like to use an *accurate* analytical model to explore the design space with a minimal computational requirement.

We use both analytical modeling as well as actual trace-driven simulation. The analytical models that we use are based on a ''customized'' mean value analysis technique that has been proposed in [20] and applied in [21-23]. Trace-driven simulation is used to study a few thousand cases and, more importantly, build confidence in the analytical models. Once the validity of the analytical models has been established, we use the models to evaluate our design choices.

3.1. Customized Mean Value Analysis (CMVA)

Our CMVA models build on similar models developed to study bus-based multiprocessors [20-23]. The CMVA method is appealing because it is simple and intuitive. To start we simply follow the path of a cache miss request and sum up the waiting times and processing times along the way to form the equations for the cache miss response time. Equations of waiting times are then constructed assuming the relationship between the mean values of these times are stable and consistent.

As mentioned earlier, the operational environment that we consider for the multi is a throughput-oriented, general multiuser environment where each processor is running a different user task. We also assume that the average task characteristics for the tasks executing on each processor are the same, i.e., the environment is homogeneous.

3.1.1. Processor Execution Model

A processor's execution history can be viewed as consisting of two alternating phases, an *execution* phase and a *blocking* phase. During the execution phase the processor executes instructions uninterrupted, with all memory requests satisfied by its local cache. The processor changes to the blocking phase when it makes a blocking bus request. We distinguish between blocking and non-blocking bus requests. A processor cannot proceed unless its blocking request (read miss or invalidation) is satisfied; it can proceed without waiting for its non-blocking request (write back of a dirty block) to finish. The relationship of these events is shown in Figure 2.

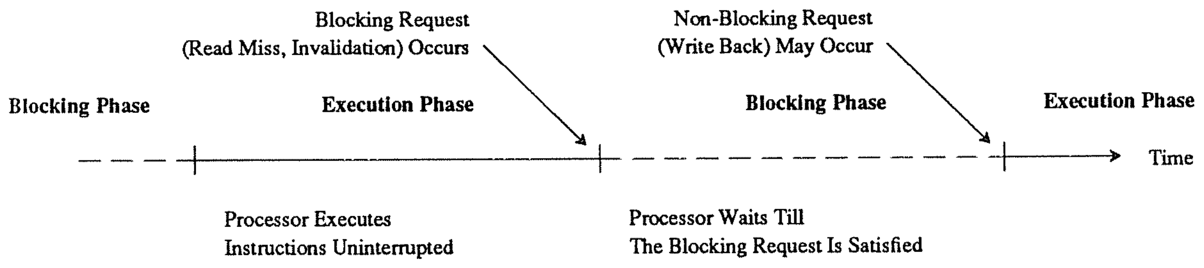


Figure 2: Execution History of A Processor

The throughput of a processor during a time period represented by consecutive execution and blocking phases is the number of instructions executed during the two phases divided by the duration of the two phases. Since the processor is blocked during the blocking phase, the throughput can be calculated as the mean number of instructions executed by the processor during an execution phase, divided by the mean total time of the execution and the blocking phases. The mean number of instructions executed in an execution phase, and the mean length of the phase are derived from the trace-driven simulation of a single processor and its cache since these values are not influenced by other processors in the system. The mean length of a blocking phase, or the mean response time of a blocking bus request, is calculated using a CMVA model of the shared bus and main memory.

3.1.2. Circuit Switched Bus

We use the following notation:

Input Parameters

- T_e is the mean processing time of a processor between two successive blocking bus requests, i.e., the duration of the execution phase. T_e can be expressed as $IT/(M \times M_{ref})$, where IT is the average instruction execution time, assuming all memory references are cache hits. M is the sum of cache miss and invalidation ratios[4], and M_{ref} is the average number of memory references generated per instruction.
- T_a is the bus arbitration time. One cycle is charged for bus arbitration when a request arrives at the bus and the bus is not busy.
- T_{ro} (T_{vo}, T_{wo}) is the time for which the bus is needed to carry out a read (invalidation, write back) operation, excluding the bus arbitration time. The main memory latency is included as a part of T_{ro} for a circuit switched bus.
- P_r is the probability that a blocking bus request is a read operation.
- P_v is the probability that a blocking bus request is an invalidation operation; note that $P_v + P_r = 1$.
- P_w is the probability that a cache miss results in a write back of a dirty cache block.
- N is the total number of caches (or processors) connected to the shared bus.

Output Parameters

- R is the mean time between two successive blocking bus requests from the same cache.
- Rs_r (Rs_v) is the mean response time of a read (invalidation) request, weighted by P_r (P_v).
- W_{rv} is the mean bus waiting time of a read or an invalidation request.
- T_r (T_v, T_w) is the bus access time of a read (invalidation, write back) request, including the bus arbitration time.
- U_r (U_v, U_w) is the partial utilization of the bus by the reads (invalidations, writes back) from one cache.
- U_{rv} is the partial utilization of the bus by the blocking (read and invalidation) requests from one cache.
- U is the partial utilization of the bus by the requests from one cache; NU is the total bus utilization.
- B_r (B_v, B_w) is the probability that the bus is busy servicing a read (invalidation, write back) request from a particular cache, when a new read or invalidation request arrives.

[4]Invalidation ratio is defined in a similar way to cache miss ratio. A write hit to a clean block generates an invalidation, and invalidation ratio is the percentage of memory references that cause invalidation.

- Re^r (Re^v, Re^w) is the residual service time of a read (invalidation, write back) request, when the request is currently being serviced by the bus when a new read or invalidation request arrives.
- W_w is the mean bus waiting time of a write back request.
- $\overline{Q}_r$ ($\overline{Q}_v$, $\overline{Q}_w$) is the mean number of read (invalidation, write back) requests from the same cache in the bus.
- K^r_{rv} (K^v_{rv}, K^w_{rv}) is the mean waiting time of a read or an invalidation request, due to the read (invalidation, write back) requests already in the bus.
- K^r_w (K^v_w, K^w_w) is the mean waiting time of a write back request, due to the read (invalidation, write back) requests already in the bus.

Response Time Equations

The mean time between two successive blocking bus requests (read miss or invalidation) from the same cache, R, is the sum of T_e and the mean time spent in the blocking phase, which is the weighted mean of the delays of the two types of blocking bus requests. Therefore,

$$R = T_e + Rs_r + Rs_v; \text{ where}$$

$$Rs_x = P_x\,(W_{rv} + T_x); \quad x = r, v$$

The time that a request spends on the bus is the time that is needed to service the request once it has obtained mastership of the bus, plus any time that might be spent in arbitration for bus mastership. If the bus is busy servicing a request while the arbitration for mastership for the next request takes place, the arbitration time is overlapped completely and does not contribute to the time spent by a request on the bus. On the other hand, the entire time to carry out arbitration is added to the time spent on the bus by a request if the request arrives when the bus is free. We approximate the arbitration time component of a request's bus tenure by considering it to be proportional to the probability that the bus is busy when a request from a cache arrives.

The probability that the bus is idle is $(1 - NU)$. However, since a cache can have only one outstanding blocking request at a time, a blocking request will never see another blocking request from the same cache using the bus when the request reaches the bus. The fraction of time that the bus is servicing a blocking request from a particular cache is U_{rv}. A new blocking request from the same cache can therefore arrive at the bus only during the remaining fraction of time, i.e., $(1 - U_{rv})$. Of this fraction, $(NU - U_{rv})$ is spent servicing other requests. Therefore, the probability that the bus is busy when a blocking request arrives from a cache is $\frac{NU - U_{rv}}{1 - U_{rv}}$, and the probability that the bus is idle is $(1 - \frac{NU - U_{rv}}{1 - U_{rv}})$.

For a non-blocking request (write back) this probability becomes $(1 - \frac{NU - U}{1 - U})$. U instead of U_{rv} is used because we assume that a write back request can only be issued immediately after a cache block is returned from the main memory (a result of an earlier blocking request on a cache miss), and it should never see any other request, blocking or non-blocking, from the same cache using the bus. Therefore, the total bus access times for blocking and non-blocking requests are:

$$T_x = (1 - \frac{NU - U_{rv}}{1 - U_{rv}}) \times T_a + T_{xo} = \frac{1 - NU}{1 - U_{rv}} \times T_a + T_{xo}; \quad x = r,\ v$$

$$T_w = (1 - \frac{NU - U}{1 - U}) \times T_a + T_{wo} = \frac{1 - NU}{1 - U} \times T_a + T_{wo}$$

Waiting Time Equations

Using the mean value technique for queuing network models [24], we decompose the waiting time of an arriving request into three components based on the types of the requests that delay the service of the new request. For a blocking request:

$$W_{rv} = K_{rv}^{r} + K_{rv}^{v} + K_{rv}^{w}$$

where

$$K_{rv}^{x} = (N - 1)\left[(\bar{Q}_x - B_x) \times T_x + B_x \times \mathrm{Re}^x\right]; \quad x = r,\ v$$

$$K_{rv}^{w} = N\left[(\bar{Q}_w - B_w) \times T_w + B_w \times \mathrm{Re}^w\right]$$

The residual service time for the request that is being serviced when a new read or invalidation request arrives is [24]:

$$\mathrm{Re}^x = \frac{T_x}{2}; \quad x = r,\ v,\ w$$

The probabilities that the bus is busy servicing the request from a particular cache when a new read or invalidation request arrives can be approximated as:

$$B_x = \frac{U_x}{1 - U_{rv}}; \quad x = r,\ v,\ w$$

where

$$U_x = \frac{P_x T_x}{R}; \quad x = r,\ v$$

$$U_w = \frac{P_w P_r T_w}{R}$$

$$U_{rv} = U_r + U_v$$

$$U = U_r + U_v + U_w$$

A scaling factor of $(1 - U_{rv})$ is used because when a blocking request such as a read or an invalidation arrives at the bus it will not see any blocking request from the same cache being serviced by the bus.

The mean number of requests from a particular cache, or the mean partial queue lengths contributed by a particular cache seen by the arriving request can be approximated by:

$$\bar{Q}_x = \frac{Rs_x}{R} = \frac{P_x\,(W_{rv} + T_x)}{R}; \quad x = r,\, v$$

$$\bar{Q}_w = \frac{P_w\,P_r\,(W_w + T_w)}{R}$$

Here the queue lengths include the request that is currently being serviced by bus. K^r_{rv}, K^v_{rv} and K^w_{rv} can now be computed as:

$$K^x_{rv} = (N-1)\left[(\frac{P_x\,(W_{rv} + T_x)}{R} - B_x) \times T_x + B_x \times \mathrm{Re}^x \right]; \quad x = r,\, v$$

$$K^w_{rv} = N\left[(\frac{P_r\,P_w\,(W_w + T_w)}{R} - B_w) \times T_w + B_w \times \mathrm{Re}^w \right]$$

Similarly, we can derive the waiting time equations for a write back request:

$$W_w = K^r_w + K^v_w + K^w_w$$

where

$$K^x_w = (N-1)\,\bar{Q}_x\,T_x = (N-1) \times \frac{P_x\,(W_{rv} + T_x)}{R} \times T_x; \quad x = r,\, v$$

$$K^w_w = (N-1)\,\bar{Q}_w\,T_w = (N-1) \times \frac{P_w\,P_r\,(W_w + T_w)}{R} \times T_w$$

In the above equations for K^x_w and K^w_w, we assume that a write back request immediately follows a cache miss read and is issued after the main memory reply to the miss read arrives at the cache. Therefore when the write back request arrives at the bus, the residual service time of the request is simply a complete service time of the request.

3.1.3. STP Bus

The derivation of the CMVA model for an STP bus is carried out along similar lines as in the case of a circuit switched bus and is summarized in the Appendix.

3.2. Trace Driven Simulation

3.2.1. Simulators

Our trace-driven simulation, whose main purpose is to validate the models of section 3.1, is carried out using a software simulator which simulates program execution on a Sequent Symmetry-like multiprocessor. The simulator consists of three modules: i) a program interpreter or tracer, ii) a cache simulator and iii) a shared bus (and main memory) simulator. Figure 3 shows the basic setup of the simulator.

The benchmark program whose execution is to be simulated is compiled into the Intel 80386 machine language using the Sequent Symmetry C compiler. The tracer program then uses the *ptrace*

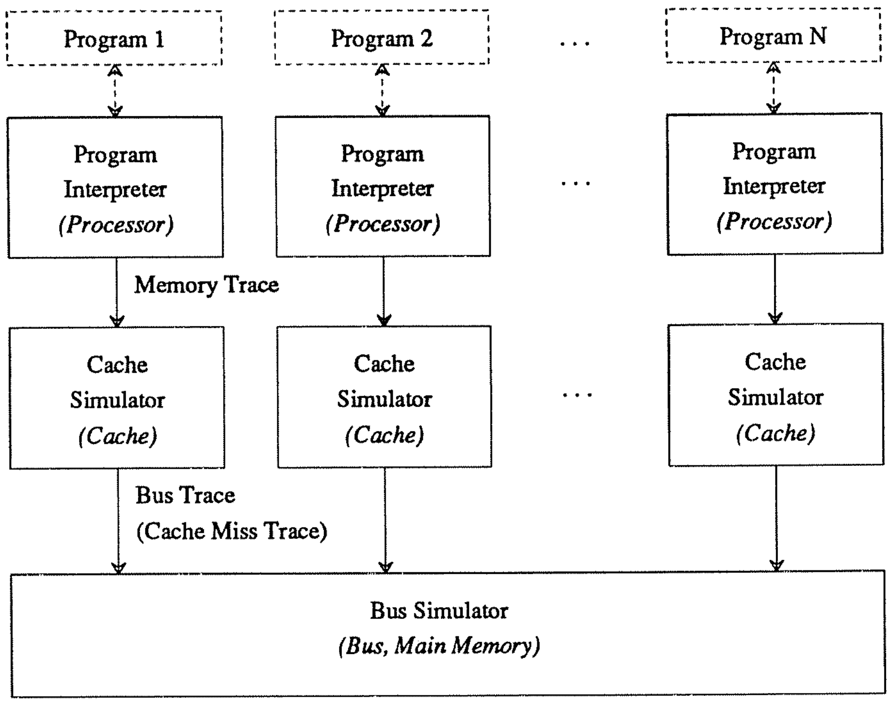

Program 1
Program 2
Program N
Program Interpreter (Processor)
Memory Trace
Cache Simulator (Cache)
Bus Trace (Cache Miss Trace)
Bus Simulator (Bus, Main Memory)

Figure 3: Trace-Driven Simulation Setup

facility of Dynix[5] and interprets the program to obtain a dynamic *memory trace*. It does so by stopping after the execution of each instruction, examining the core image of the instruction, and interpreting the instruction to generate the memory reference trace records. Each memory trace record contains the *virtual memory address* accessed, the *access type* (a read or a write) and the *time* the access is made. The time associated with each memory reference in the trace generated by the tracer program is the dynamic instruction number which generated the memory reference and is an ideal number that would represent the time at which the memory reference would be generated if: i) all memory references generated by an instruction are generated simultaneously, ii) all memory references are serviced in zero time and iii) all instructions take the same amount of time to execute.

Since the time at which memory references are generated during the execution of an instruction and the execution times of each instruction are highly dependent upon the implementation of the processor, we shall assume that all instructions take the same amount of time to execute, and that all memory references from an instruction are generated at the same time (of course they are submitted to the memory one at a time). To obtain realistic times at which the memory references would be generated and serviced in the multiprocessor environment, the memory traces have to be passed through the cache and bus simulators.

The memory trace generated by the tracer program is used to drive a *cache simulator*. By filtering out references that are cache hits (of course depending upon the cache organization), the cache simulator generates a *cache miss, write back*, and *invalidation* trace, i.e., a trace of *bus requests*. Each bus trace record contains the time of generation (still an ideal time) and the type of the bus request. We use the Berkeley Ownership protocol for generating the invalidation requests [25], though any other protocol

[5]Dynix is Unix operating system adapted to run on the Sequent Symmetry.

could be used in a straightforward manner in the cache simulation. Although our throughput-oriented environment precludes any sharing of data between caches, we assume that the coherence protocol is enforced at all times, and an invalidation request is generated when a write occurs to a clean cache block.

Bus traces from several benchmark programs are then used to drive a *bus simulator* which simulates the operation of the shared bus and the main memory. In deciding which request is to be serviced next, the bus simulator uses a FCFS policy. The relevant delays and timing parameters in our simulation model are shown in Figure 4. For each input bus request, the latency seen by the request is the sum of bus queuing delay, the bus transfer time of the request, and for a cache miss read, the reply. As a result of the bus simulation, the realistic time at which each memory reference is serviced is obtained. Note that the decoupling of cache and bus simulations is possible due to the assumption of a throughput-oriented multi-user environment. In such an environment the actual memory performance will not affect the occurrences and the partial ordering of the events that happen on each processor and cache (this may not be true if the multiprocessor is executing a parallel program). The bus simulation simply calculates a total ordering with a correct time scale for all the events in the system.

In our simulation, we assume that a processor stalls until its blocking request (cache miss read or invalidation) is serviced, i.e., it can have only one outstanding memory request[6]. We also assume that there is no task migration. The latter assumption is made to keep the simulator manageable and does not affect our purpose of the simulation, which is to validate our CMVA models.

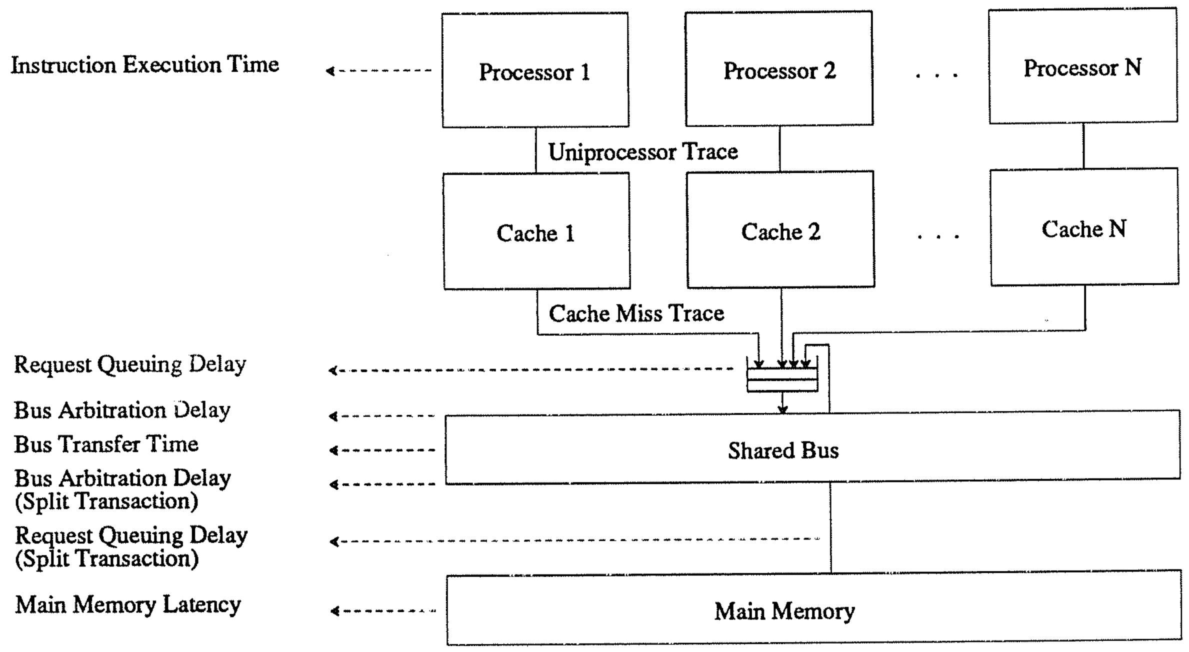


Figure 4: Timing Delays in the Trace-Driven Simulation Model

[6]This is true in shared bus multiprocessors that use microprocessors as their CPUs. Most microprocessors allow only a single outstanding memory request.

3.3. Model Validation

The benchmark programs that we use to validate the models are: i) *as*, which is the assembler for the Intel 80386 processor, ii) *cache*, the cache simulator itself, iii) *csh*, the command interpreter, iv) *nroff*, the nroff text processing program. These benchmarks are used for validation since they are commonly used in the Unix environment and also so that we can simulate their execution and obtain complete and accurate knowledge about which memory references are associated with each instruction. This information is necessary to associate an ideal time of generation for each memory request. We would like to mention that cache and bus simulation (mentioned later) to validate the models could be carried out using other traces. However, most such traces are typically a sequence of memory references, with no explicit notion of the time of generation of each reference, and associating an ideal time of generation with each reference is not always possible.

Using the tracer program a memory trace is collected for each benchmark for 1 million instructions executed. The traces are then passed through the cache simulator. For each cache configuration and for each memory trace we collect a set of statistics and generate a bus request trace file. The statistics (M, M_{ref}, P_r, P_v, and P_w) are used as inputs to the CMVA models, and the bus request trace file is used to drive the bus simulator.

Bus simulation is then carried out using the bus request trace files for each configuration. More details on the bus simulation, and how statistics are gathered during bus simulation, can be found in [26].

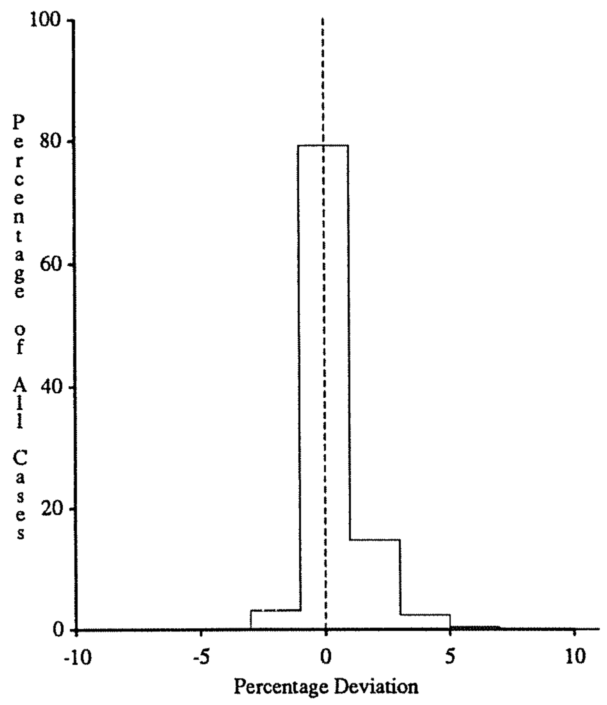
To validate the model, we consider several multiprocessor configurations with varying number of processors, memory system parameters and instruction execution speeds. More specifically, we consider: i) average zero memory-wait-state instruction execution times of 2, 3 or 4 bus cycles[7], ii) 8 or 16 processors, iii) cache sizes of 4K, 8K, 16K, 32K, 64K, 128K or 256K bytes, iv) cache block sizes of 4, 8, 16, 32, 64, 128, 256 or 512 bytes, v) direct mapped or 2-way set-associative caches, vi) main memory latencies of 3, 5, 7 or 9 cycles and vii) circuit switched or STP buses, each with a bus width of 32 bits, and multiplexed address and data lines. The cross product of the parameters therefore allows us to evaluate and compare system performance using our trace-driven simulation and our CMVA models for 5,376 system configurations.

Using inputs to the CMVA models obtained from the cache simulations mentioned above, and the other parameters of the memory system configuration, we solve the models iteratively and obtain the average multiprocessor throughput, which is the sum of the throughputs of the individual processors. Having obtained the results using both techniques, we compute the percentage difference between the values obtained. Since our emphasis in this paper is evaluating design choices, and since we shall use processor throughput as the performance metric (see section 4), we only consider the validity of the models in determining processor throughput here. A comprehensive comparison between the results of the models and simulation for other metrics, such as read latency and bus utilization, is carried out in [26].

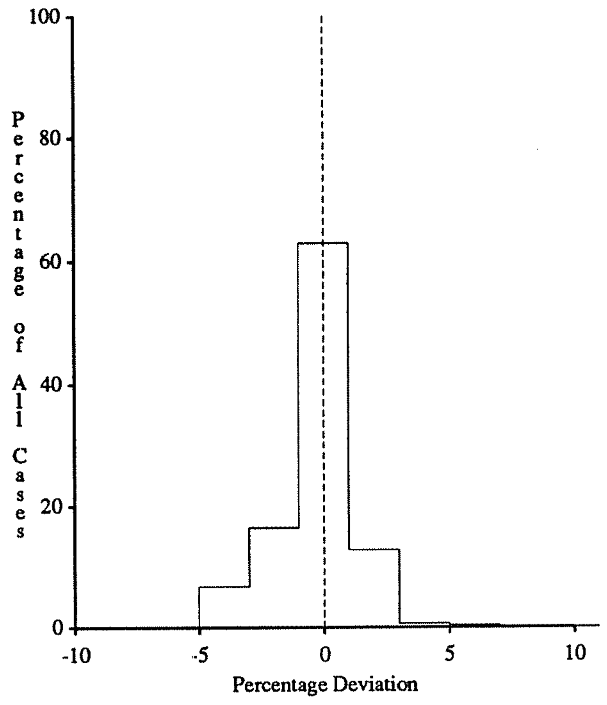
Figure 5 histograms the percentage difference between the values of the average multiprocessor throughput obtained from the trace-driven simulation and the CMVA analysis. Figure 5(a) and 5(b) each represent 2,688 system configurations using circuit switched and STP buses, respectively. In the histograms, a negative difference indicates that the value obtained by the CMVA models is less than the value obtained by simulation. Each step in the histogram represents a 2% difference.

Figure 5(a) and 5(b) show that in about 80% of the 2,688 cases of a circuit switched bus and 60% of the 2,688 cases of an STP bus, the magnitude of the difference in multiprocessor throughput obtained from the two techniques is less than 1%. We can also see that in more than 92% of the cases the magnitude is less than 3%, in less than 0.5% of the cases the magnitude of the difference is greater than 5%, and

[7]All times in our model are in terms of bus cycles where a bus cycle is the time taken for a single transfer on the bus.



(a) Circuit Switched Bus



(b) STP Bus

Figure 5: Difference in Processor Throughput Between the Results of the CMVA Models and Trace-Driven Simı

all the differences are within 8%. This is quite encouraging since it establishes the accuracy of our CMVA models and allows the evaluation of the design choices to be carried out using the models.

4. EVALUATION OF DESIGN CHOICES

Since our models are quite accurate over a wide range of multiprocessor configurations as illustrated in the previous section and in [26], and since their solution is 4-5 orders of magnitude faster than trace-driven simulation, we use the models for our evaluation of design choices. To provide inputs to the models, we need to obtain values of M, M_{ref}, P_r, P_w and P_v from traces that are representative of the workload for which we want to evaluate design choices. While the miss ratio characteristics, i.e., M, of various cache organizations are easily available from the literature for a wide variety of workloads, the values of M_{ref}, P_r, P_w and P_v are typically not available.

We could use the traces of section 3.3 which were used to validate our models. However, while those traces were adequate for model validation, we feel that they are not sufficiently representative of workloads for which we would like to evaluate design tradeoffs, especially since they contain no operating system activity. Moreover, to put our results in perspective, we would like to use workloads that have been used previously for uniprocessor cache studies. Therefore, we use traces generated using the Address Tracing Using Microcode (ATUM) technique [27].

In the ATUM technique, patches are made to the microcode of the machine to generate addresses for all the memory references made by the processor. These references include references made by the user programs as well as references made by the operating system. The ATUM traces that we use are gathered via microcode patches on a VAX 8200 by Agarwal and Sites. These traces are distributed by

DEC, are considered to be the best public-domain traces for a multiprogrammed, multi-user environment, and they have been widely used in recent cache studies [6-8,28,29]. By passing the ATUM traces through a uniprocessor cache simulator we obtain the values of M, M_{ref}, P_r, P_w and P_v.

Keeping in mind that our goal is to evaluate the impact of a particular design choice in the memory system on the peak multiprocessor throughput that can be supported by the memory system, we compute the *maximum multi throughput* for the memory system configuration (cache, shared bus and main memory). This is done using the following procedure. For each memory system configuration, we compute the total multi throughput (which is the sum of the throughputs of each processor in the multi) for an increasing number of processors. The maximum multi throughput is the throughput at the point beyond which the addition of more processors contributes less than 1% to the total throughput of the multi, i.e., the throughput when the bus is saturated. The exact number of processors in the multi at the point at which the maximum multi throughput is achieved varies with the parameters of the memory system.

Unless mentioned otherwise, for all the system configurations that we evaluate in the coming sections, we assume that the bus is 32 bits wide with multiplexed address/data lines and has a cycle time of 50ns (or is a 20 MHz bus), the processor CPUs have a peak performance of 5 VAX MIPS and all caches are write back. We would like to mention that the results we present are not tied specifically to the processor and bus speeds. We have obtained similar results for other CPU and bus speeds but we do not present them in this paper due to space considerations. In all our experiments, throughput is measured in VAX MIPS since the traces that we use are relevant only to VAXen.

4.1. Cache Performance Metrics and Uniprocessor Performance

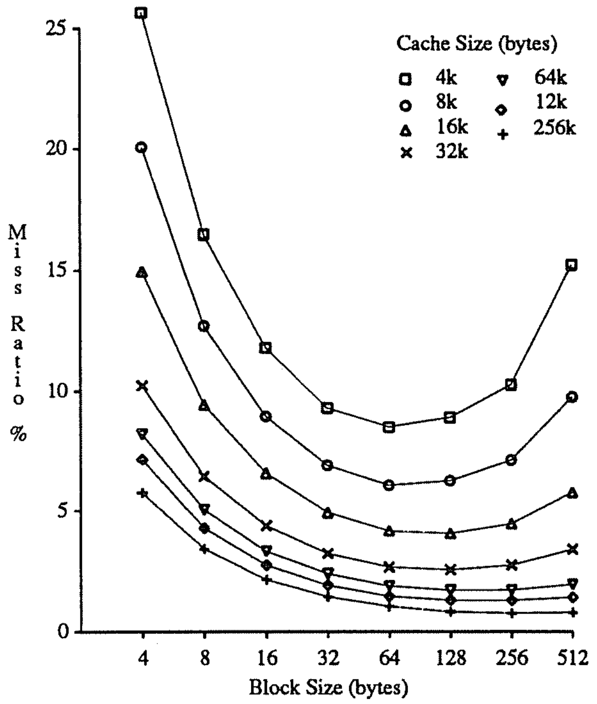
Before we evaluate our design tradeoffs, we consider the performance of several uniprocessor cache organizations using the ATUM traces and traditional uniprocessor cache performance metrics. This allows the design choices for the multiprocessor memory system to be compared with equivalent choices for a uniprocessor memory system.

The miss ratio (in percentage) is presented in Figure 6(a) for various cache sizes and block sizes (all caches are direct mapped and write back). For bus traffic[8], we distinguish between *data only* traffic (Figure 6(b)) and *data and address* traffic (Figure 6(c)). The data traffic includes only the actual data transfer cycles whereas the address and data traffic also includes the addressing overhead (the bus is 32 bits with multiplexed address and data lines). The data traffic ratio (in percentage) is the ratio of the traffic that appears on the bus in the presence of a cache to the traffic that appears without the cache. Thus, if the data traffic ratio is 400%, it means that the traffic on the bus with the cache is 4 times as much as the traffic without the cache. We will use the data of Figure 6 shortly.

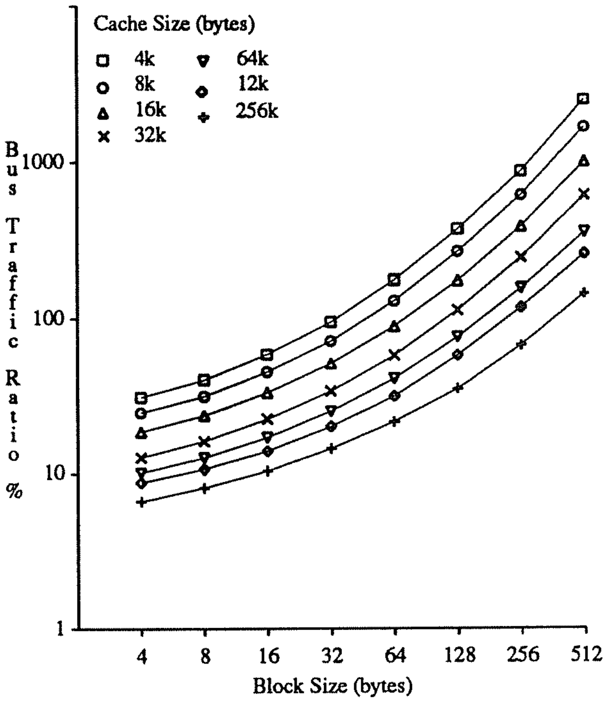
As mentioned earlier, the impact of the memory system on processor performance is directly governed by equation (1). In a uniprocessor, if we assume that T_C^P and T_m^C are independent of the cache organization, the best cache organization is one that minimizes the overall miss ratio. However, as mentioned earlier, T_C^P and T_m^C are not independent of cache organization, and to evaluate the impact of the entire memory system on processor throughput, the impact of T_C^P and T_m^C must be considered. This is illustrated in Figures 7 and 8.

In Figure 7, we plot the throughput of a uniprocessor (in VAX MIPS) as a function of the main memory latency for several cache sizes and main memory latencies. For all cases, the cache is direct mapped. The trends to be observed from Figure 7 are somewhat obvious: i) as the main memory latency increases, T_m^C increases and consequently the throughput of the uniprocessor decreases and ii) the impact of the main memory latency on processor throughput is sensitive to the cache block size. As we shall see, in multiprocessors neither trend needs to be as pronounced as in the case of uniprocessors, especially with

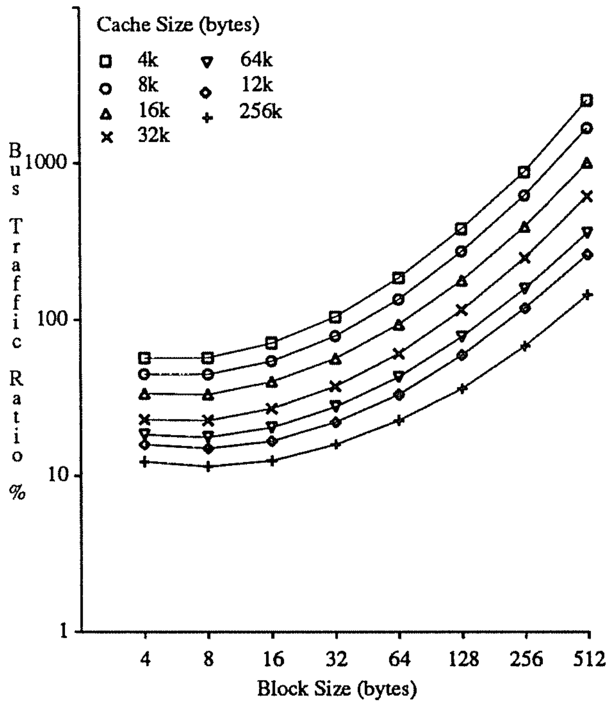
[8]Bus traffic includes traffic generated to service miss requests, as well as write back and invalidation requests.



(a) Miss Ratio

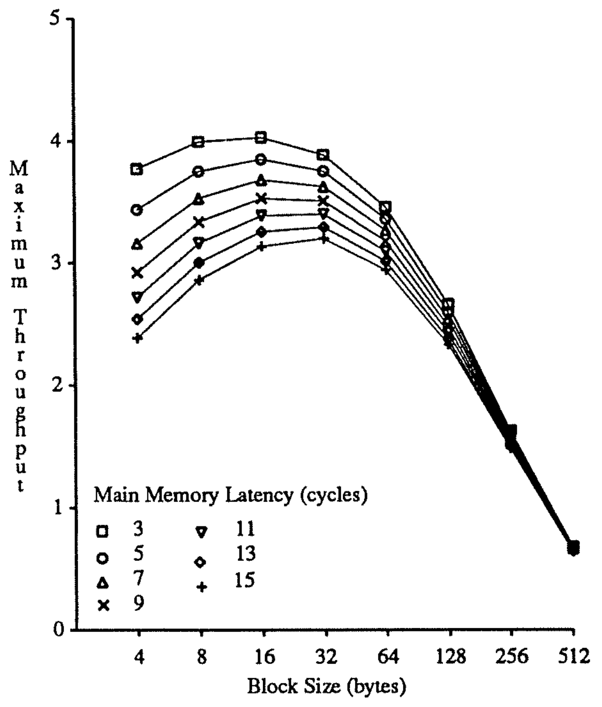


(b) Bus Traffic Ratio (Data Only)

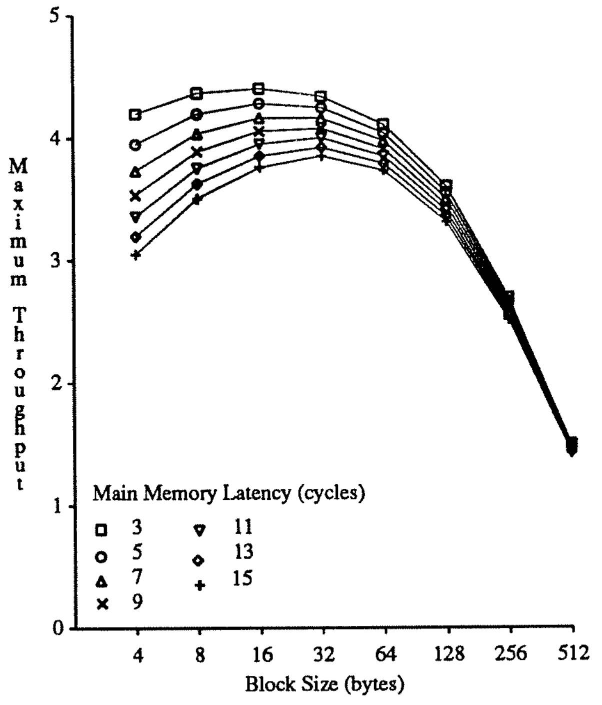


(c) Bus Traffic Ratio (Data and Address)

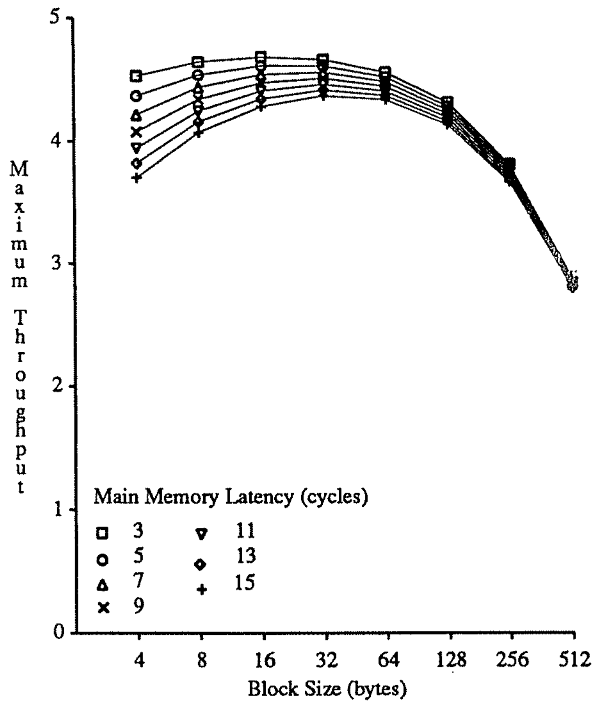
Figure 6: Cache Performance Metrics for the ATUM Traces



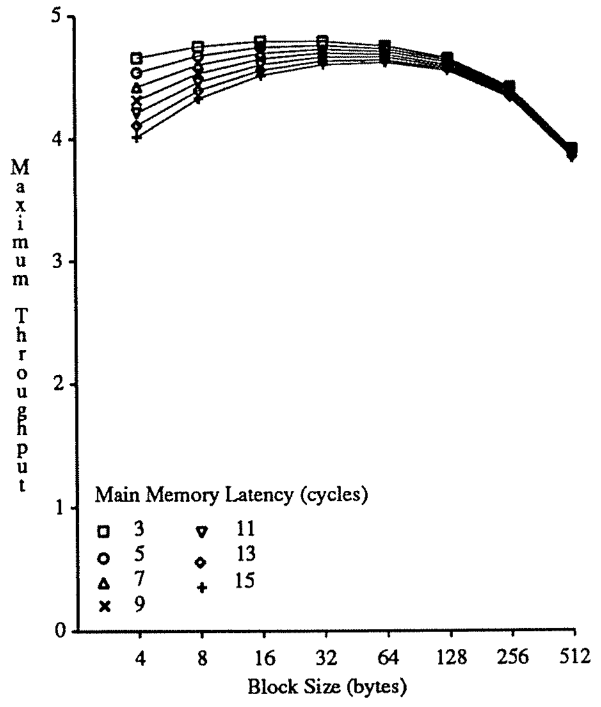
(a) Cache Size = 4K bytes



(b) Cache Size = 16K bytes



(c) Cache Size = 64K bytes



(d) Cache Size = 256K bytes

Figure 7: Maximum Uni-Processor Throughput (in VAX MIPS) with Varying Main Memory Latency

an STP bus. More on this in section 4.2.

Figure 8(a)-(d) plots the processor throughput for cache sizes of 4K, 16K, 64K and 256K bytes, respectively, each with varying set associativity and block size. The main memory latency is kept fixed at 250ns (5 cycles) in all cases. To account for the impact of set associativity on processor throughput, T_C^P of caches with set associativities of 2, 4 and 8 is 10% greater than T_C^P for a direct mapped cache [6]. Two trends are obvious from Figure 8. The first trend is that as cache size increases, the block size that results in the best uniprocessor throughput increases. Furthermore, the throughput tends to "flatten" out, indicating that several block sizes may give roughly the same performance. The second trend to note is that as cache size increases, the need for set associativity decreases. For larger caches, when the cycle time advantages of direct mapped caches are taken into account, direct mapped caches can actually provide better throughput than set associative caches even though the set associative caches may have a better miss ratio. Both trends apparent in Figure 8 are well known and have been described in detail in the literature on uniprocessor caches [3,7]. Our purpose of presenting them here is again to show that neither trend may occur for multiprocessor caches that we discuss in the upcoming sections.

4.2. Cache Block Size Choice

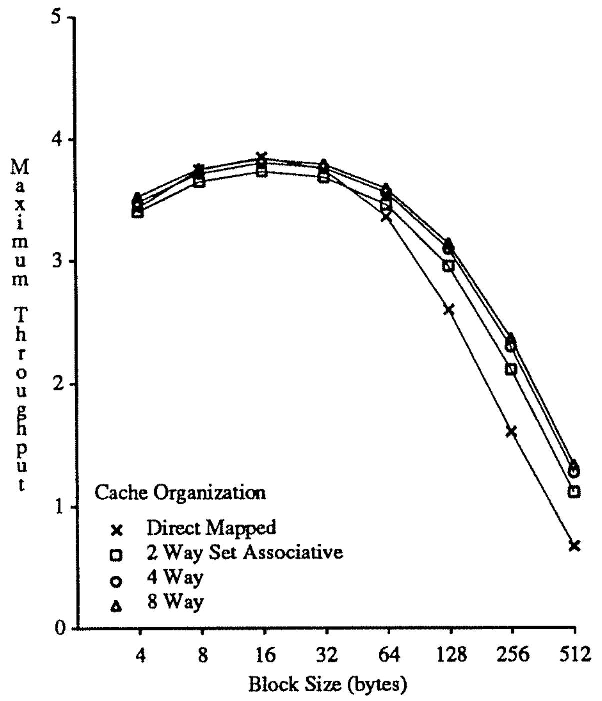
To evaluate the choice of a block size, we consider only direct mapped caches (we consider other set associativities in section 4.3). Using our CMVA models, we calculate the maximum multi throughput as the block size is varied, for different cache sizes and main memory latencies. Figures 9(a)-(d) present the maximum multi throughput (in VAX MIPS) with various cache sizes and main memory latencies for a circuit switched bus and Figures 10(a)-(d) present the same for an STP bus.

From Figure 9, we can make several observations about memory system design choices with a circuit switched bus. First, larger block sizes tend to be favored as the cache size is increased. However, the trend towards larger block sizes is not as strong as in the case of a uniprocessor (compare Figure 9 with Figure 7). While the trend towards larger block sizes may seem obvious, we point out that this conclusion can not be derived from a simple consideration of the bus traffic and/or the miss ratio cache metrics. From Figure 6 we see that the miss ratio metric favors larger block sizes as cache size is increased but the bus traffic metrics still favor smaller block sizes. In a circuit switched bus, consideration of the bus traffic alone is clearly not sufficient since the bus is held by the master until the entire transaction is complete. A read transaction includes the main memory latency and therefore, the data traffic performance metric (which is influenced only by the cache organization and not by other parameters of the memory system) is not a good indicator of the bus utilization. To accurately evaluate design choices, all factors that can influence performance must be taken into account.

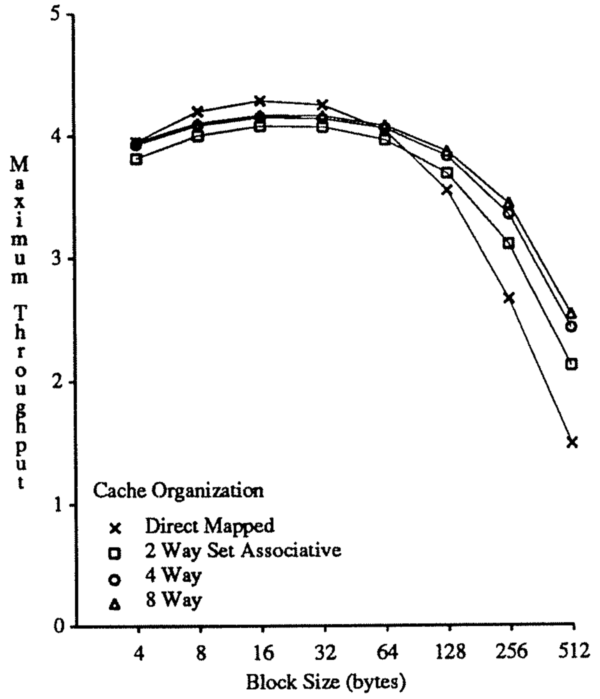
Second, the choice of the block size is also sensitive to the main memory latency in a circuit switched bus. When main memory latency is high, larger block sizes tend to be favored (Figure 9), just as in the case of a uniprocessor (Figure 7).

Third, the main memory latency has a significant impact on the maximum performance that can be achieved. For example, in going from a main memory latency of 150ns to 450ns with a 256K byte cache, the maximum multi throughput, with the best block size, decreases from about 100 MIPS to about 67 MIPS. This is because the communication protocol of a circuit switched bus is such that the bus is not available for use until the entire transaction is complete, and a large main memory latency contributes significantly to the bus utilization.

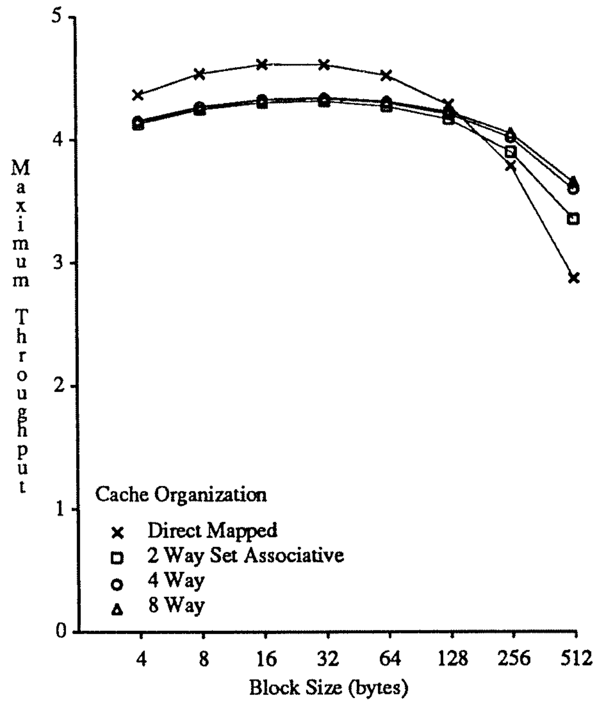
For an STP bus (Figure 10), the results are somewhat different. First, larger block sizes seem to be favored as cache size increases (up to a point), just as in the case of a circuit switched bus. In an STP bus, the bus traffic (address plus data) is an accurate indicator of the utilization of the bus. Therefore, why might larger block sizes be favored with STP buses even though smaller block sizes result in a lower bus utilization? To understand this, we need to look at equation (1) as well as the shapes of the miss ratio and the traffic ratio in Figure 6.



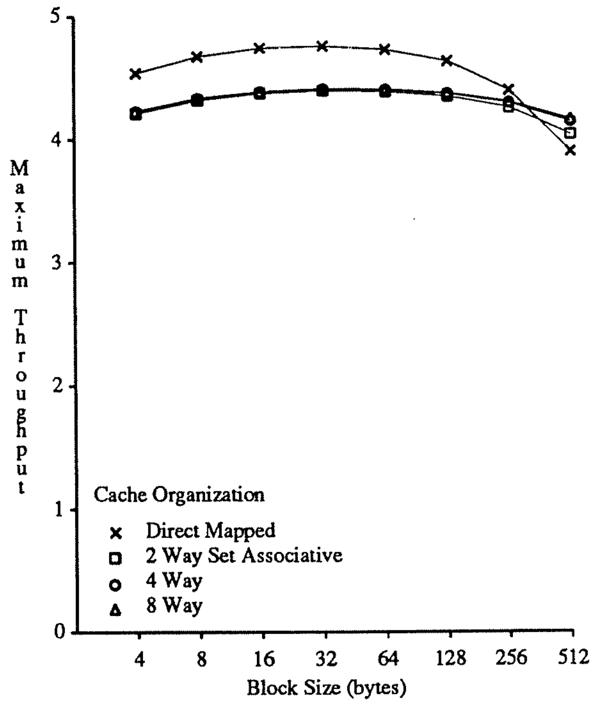
(a) Cache Size = 4K bytes



(b) Cache Size = 16K bytes

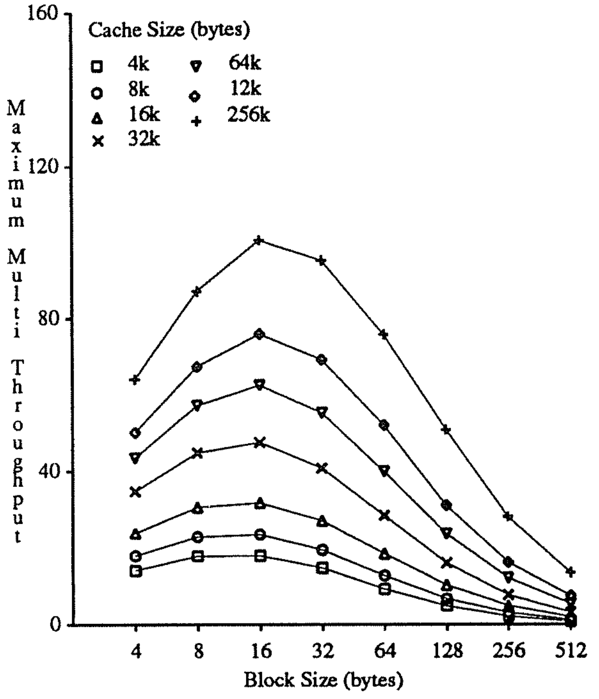


(c) Cache Size = 64K bytes

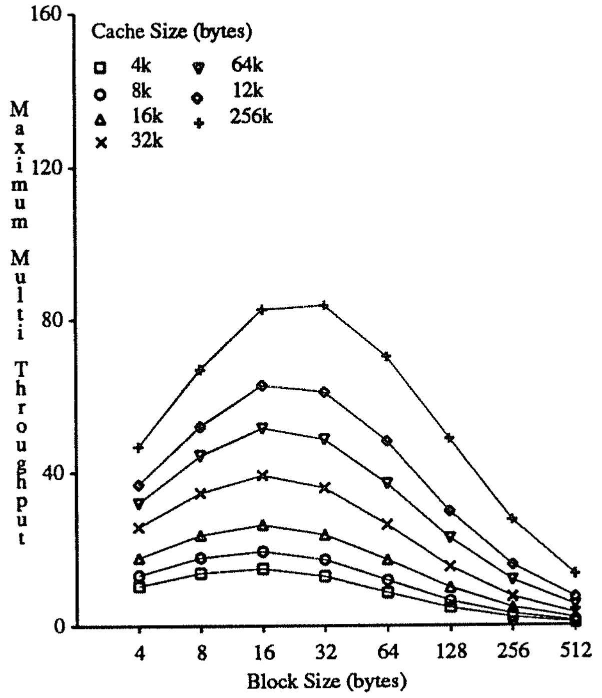


(d) Cache Size = 256K bytes

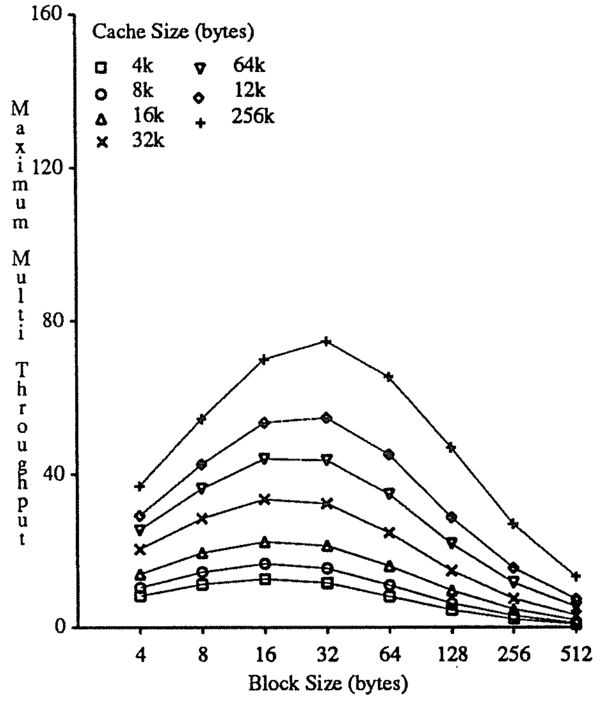
Figure 8: Maximum Uni-Processor Throughput (in VAX MIPS) with Varying Cache set Associativity



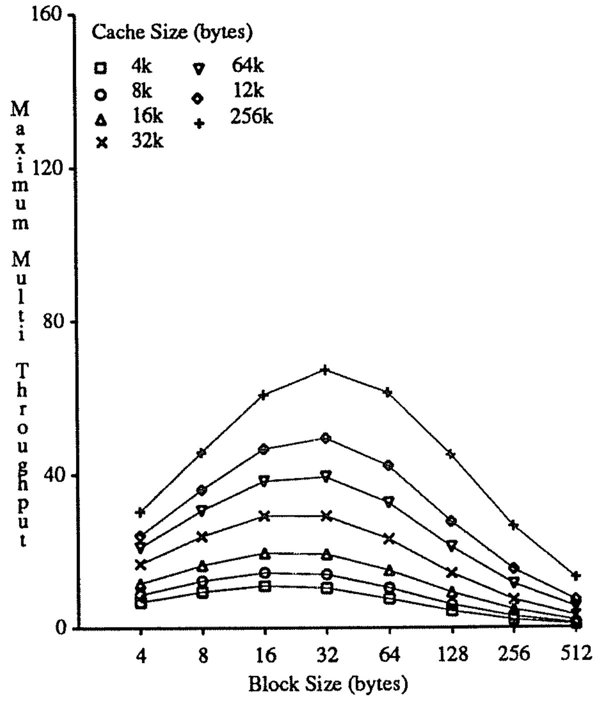
(a) Main Memory Latency = 150 ns



(b) Main Memory Latency = 250 ns

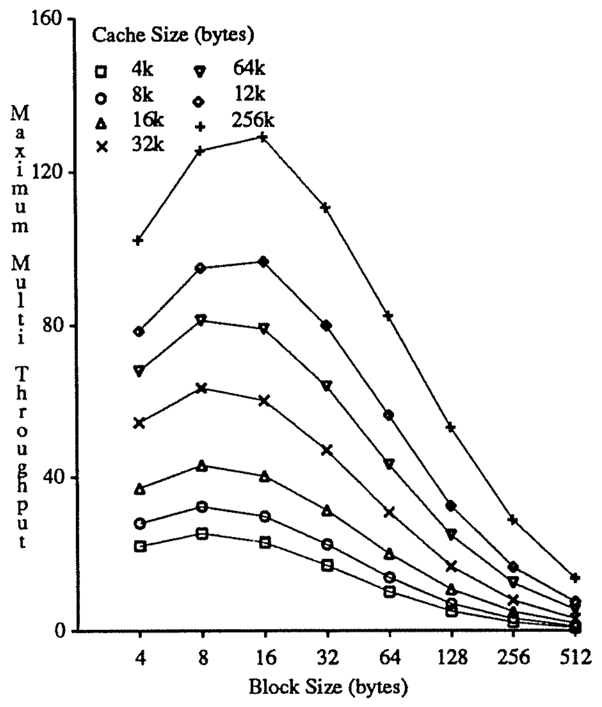


(c) Main Memory Latency = 350 ns

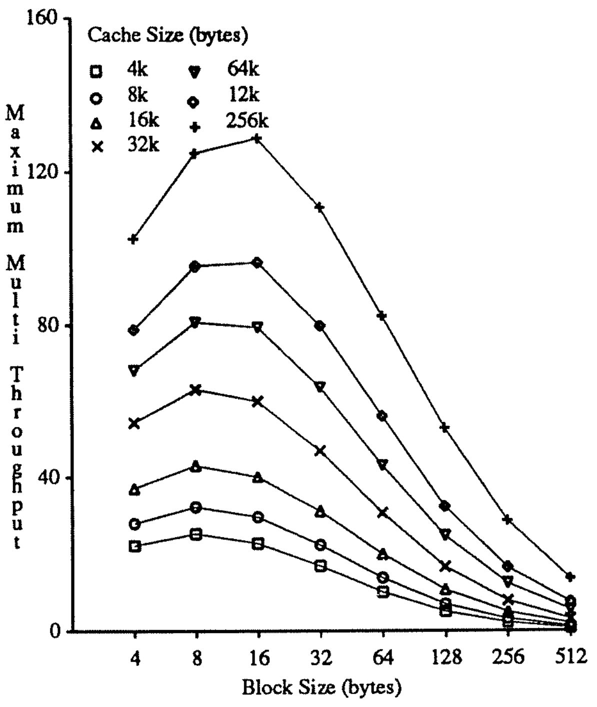


(d) Main Memory Latency = 450 ns

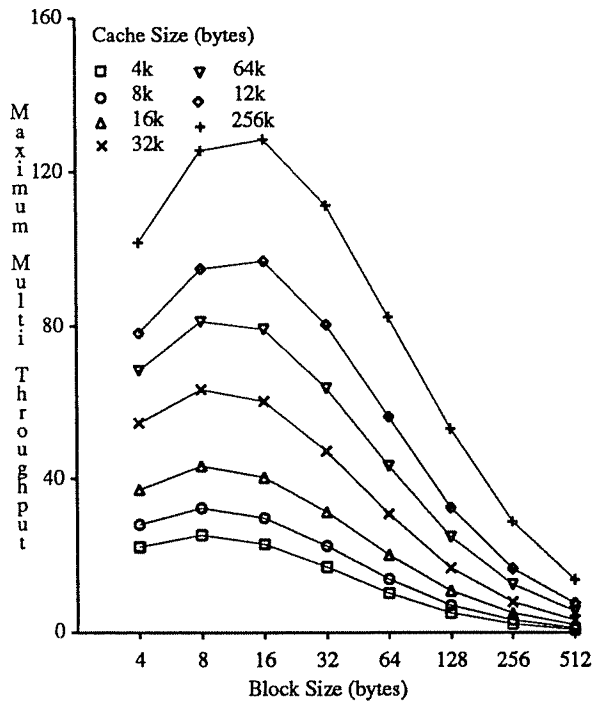
Figure 9: Maximum Multi Throughput (in VAX MIPS) with a Circuit Switched Bus



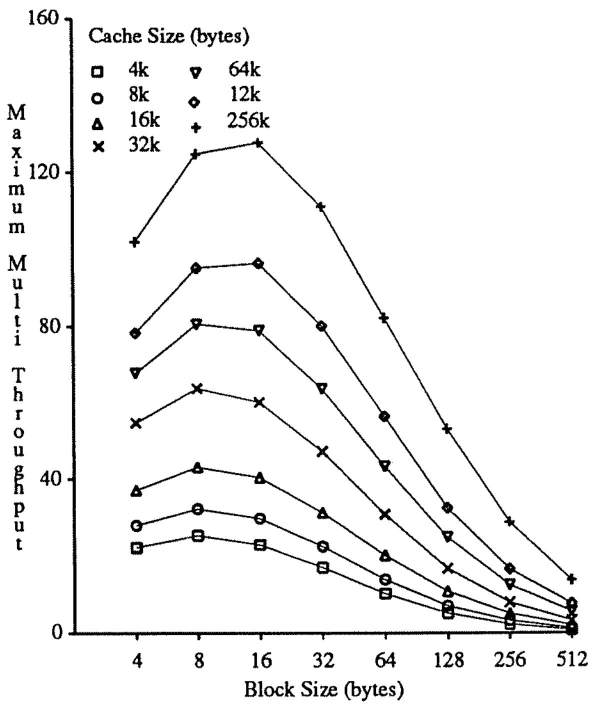
(a) Main Memory Latency = 150 ns



(b) Main Memory Latency = 250 ns



(c) Main Memory Latency = 350 ns



(d) Main Memory Latency = 450 ns

Figure 10: Maximum Multi Throughput (in VAX MIPS) with an STP Bus

The memory latency in equation (1), T_m^P, includes both the probability of making a bus request (the miss ratio M) as well as the queuing delay that the request experiences (a part of T_m^C). While the queuing delay increases as the utilization of the bus increases with a larger block size, M may decrease sufficiently with the larger block size to offset the additional queuing delay. That is, with a larger block size, the processor may be able to achieve a higher throughput by carrying out local (in cache) computation more often than with a smaller block size, even though it experiences a bigger penalty for non-local access. The opposing trends in miss ratio and bus traffic (or bus utilization in case of an STP bus) in Figure 6 lead to a best block size that may not result in either the best miss ratio or the best bus traffic.

Second, the choice of the block size that allows the best maximum multi throughput seems to be insensitive to the main memory latency. In fact, this is just one facet of a more interesting phenomenon that the maximum multi throughput appears quite insensitive to the main memory latency.

These seemingly counter-intuitive observations can be explained as follow. If we view a main memory reply in response to an earlier memory read from a processor as part of the bus access activity of the processor, increasing the main memory latency has the same effect as increasing the idling time between the two accesses (the bus read and the subsequent main memory reply). The resulting smaller bus access rate of each processor (due to the increased idling time) reduces the bus utilization and hence the bus queuing delay. Therefore, the cache miss latency, T_m^C, which includes the main memory latency as well as the queuing delay, does not increase to the same extent as the increase in the main memory latency. Figure 11 shows this effect. In the initial configuration the main memory latency is 3 cycles (or 150ns) and each processor has a 64K byte cache. By connecting a sufficient number of processors to the bus, the system saturates and delivers its maximum throughput. Keeping the same number of processors

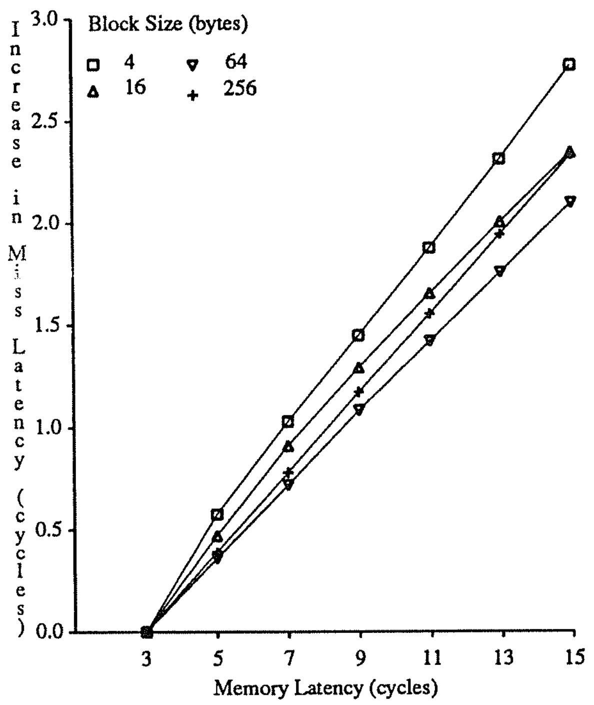


Figure 11: Increase in Cache Miss Latency

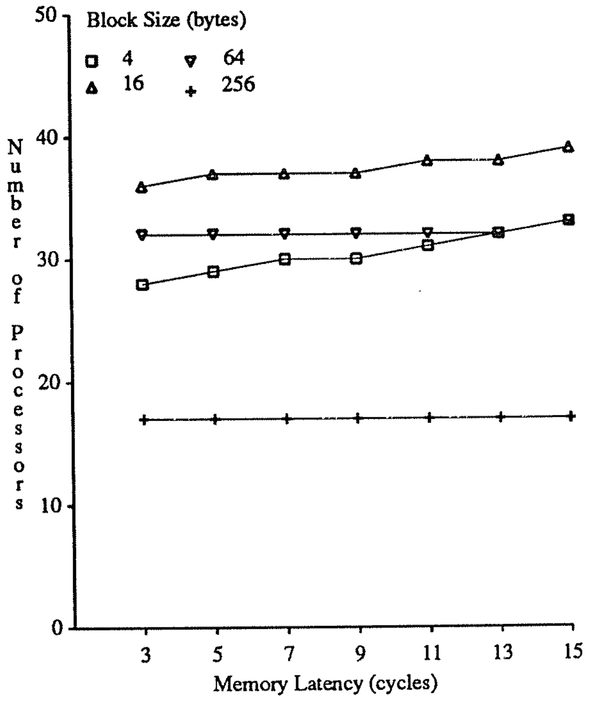


Figure 12: No. of Processors that Give the Maximum Multi Throughput

that saturate the bus with a main memory latency of 3 cycles, the increase of cache miss latency is plotted against larger values of main memory latency. From Figure 11 we can see that, for example, when block size is 64 bytes, changing main memory latency from 3 cycles to 15 cycles (750ns) increases the cache miss latency by only 2 cycles (100ns); the difference represents a decrease in the queuing delay because of the slightly slower bus access rate of each processor, and the consequent lower bus utilization.

The increase in miss latency, however reduced, still decreases the throughput of an individual processor. However, since the bus utilization is also reduced, more processors can be added to compensate for the loss of the performance of the individual processors. This is illustrated in Figure 12 which shows the number of processors used to deliver the maximum multi throughput for different main memory latencies. As we can see, the number of processors that can be connected together to achieve the maximum multi throughput increases with the decrease in throughput of each processor due to the increase in main memory latency. Putting it together, the maximum throughput of a multi with an STP bus seems to be quite insensitive to the main memory latency, as evidenced by the nearly identical graphs for varying memory latencies in Figures 10(a)-10(d). Of course, if the number of processors in the multi were fixed, the throughput of the multi would decrease as the memory latency was increased.

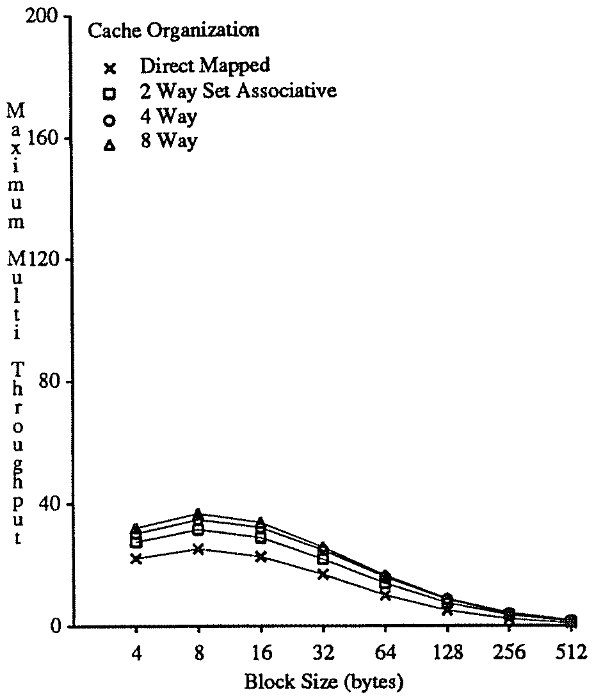
4.3. Cache Set Associativity Choice

We now consider the choice of the set associativity for the cache in a multi. In Figures 13(a)-(d), we present the maximum multi throughput that can be supported by a memory system using cache sizes of 4K, 16K, 64K and 256K bytes, respectively, with varying set associativities. For each cache size, we consider a direct mapped, 2-way, 4-way and 8-way set associative organizations. Again the cycle time of a cache with set associativity of 2, 4 or 8 is assumed to be 10% longer than the cycle time of a direct mapped cache. The bus is an STP bus and the main memory latency is 250 ns (5 cycles).

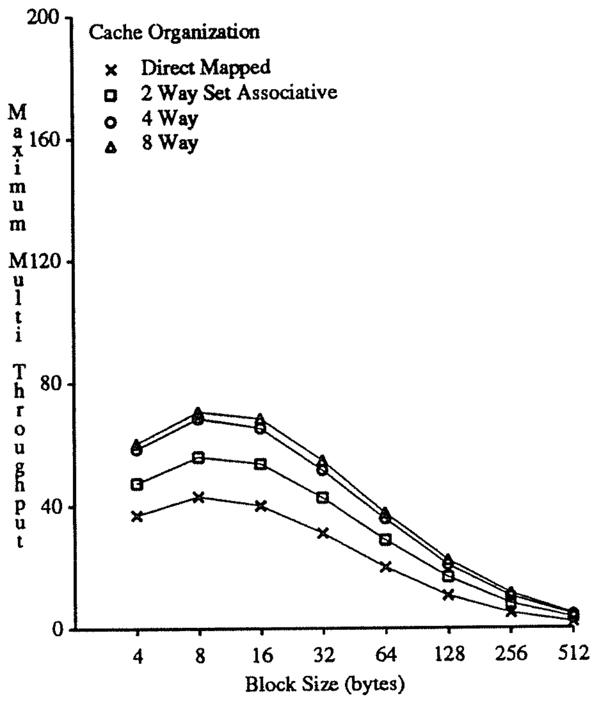
In Figure 13 we can see that for all four cache sizes, the maximum multi throughput increases at least 20% when 2-way set associative instead of direct mapped caches are used, if these caches always choose the block sizes that give the best performance. For example, when cache sizes are 256K bytes and block sizes are 16 bytes, the maximum multi throughput increases from 128 MIPS with direct mapped caches to 156 MIPS with 2-way set associative caches, an improvement of 22%. These data suggest that 2-way or 4-way set associativity may be warranted in a multi even when the cache size is quite large (256K bytes). This is unlike uniprocessor caches where the need for set associativity diminishes significantly as the cache size increases [6-8].

The reason why a larger associativity is favored for the multiprocessor caches is due to the fact that caches with a larger associativity lower the miss ratio as well as the per-processor utilization of the shared bus. The lower bus utilization results in a lower queuing delay and consequently a lower overall T_m^C. Therefore, the product $M \times T_m^C$ might decrease sufficiently to offset the increase in T_C^P, resulting in a lower T_m^P. This is in contrast to a uniprocessor in which T_m^C is independent of the cache set associativity, and a decrease in the value of $M \times T_m^C$ due to a decrease in the value of M alone may not be sufficient to offset the increase in T_C^P due to the increased cache set associativity. Furthermore, the lower per-processor bus utilization with an increased set associativity allows more processors to be connected together, and to improve the multiprocessor throughput, even though the throughput of each processor might suffer. Therefore, keeping in mind that caches in multiprocessors serve to reduce memory latency as well as to increase *system* throughput (by reducing the demand for the shared bus) whereas the main purpose of a cache in a uniprocessor is to reduce memory latency and to improve *uniprocessor* throughput, we see that a larger set associativity may be warranted in a multiprocessor even though it may not be warranted in a uniprocessor with similar memory system parameters.

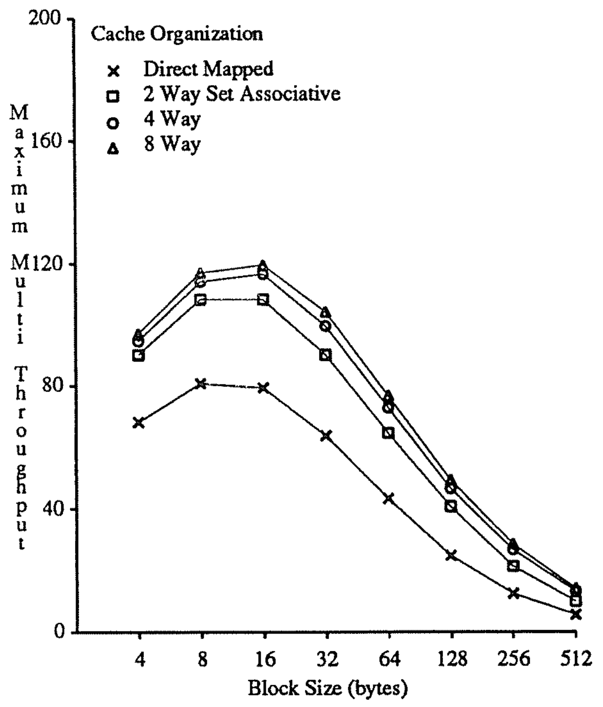
Also observe that set associativity has little effect on the choice of the best block size. This reinforces our results of section 4.2 on block size choice that were derived for direct mapped caches.



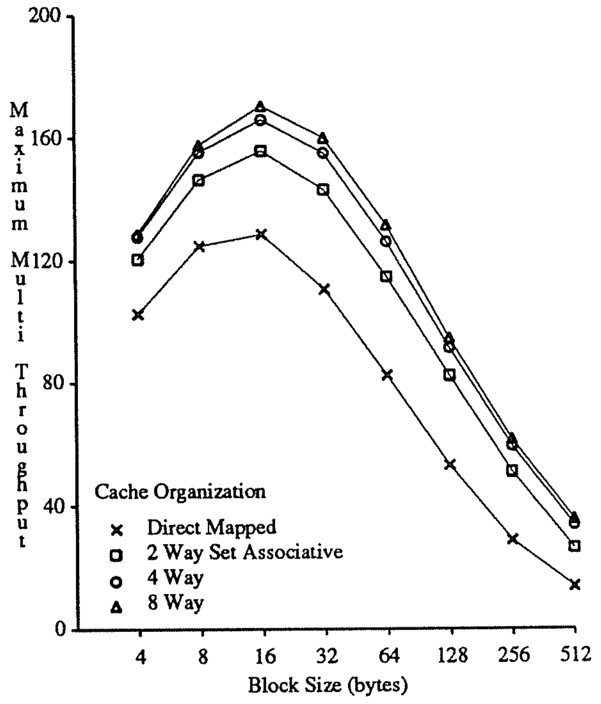
(a) Cache Size = 4K bytes



(b) Cache Size = 16K bytes



(c) Cache Size = 64K bytes



(d) Cache Size = 256K bytes

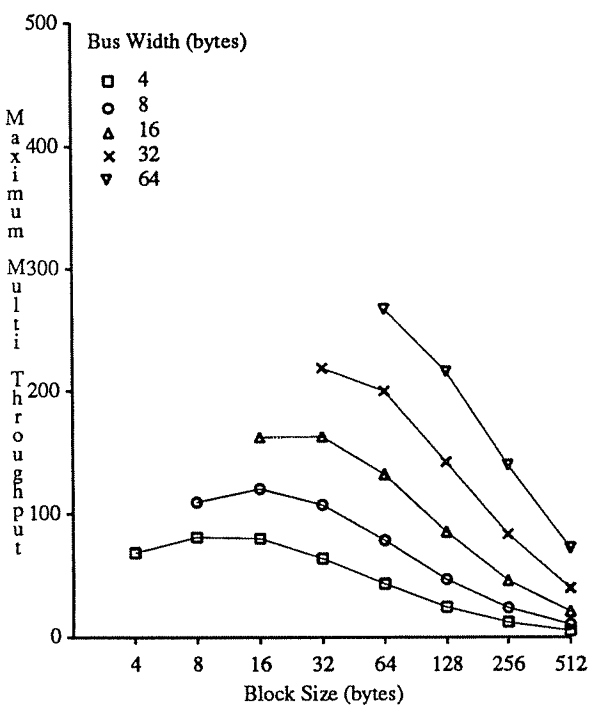
Figure 13: Maximum Multi Throughput (in VAX MIPS) with Varying Cache Set Associativity

4.4. Bus Choice

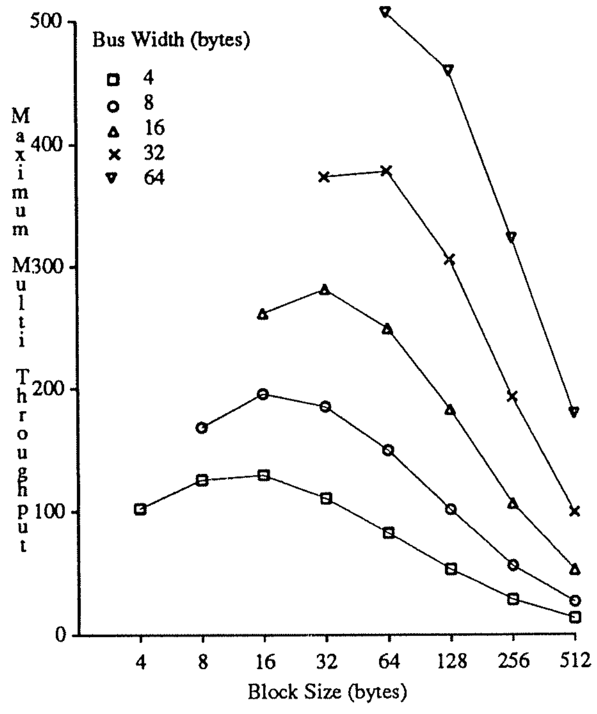
From the results presented in Figures 9 and 10, it is clear that an STP bus can provide much better maximum system performance than a circuit switched bus, especially when the main memory latency is large. Furthermore, an STP bus is able to sustain maximum system performance for a wide range of main memory latencies. However, for low main memory latencies, circuit switched buses can compete in performance with STP buses.

Finally, we consider the bus width choice. Figure 14 shows the performance impact of increasing the bus width. In all cases the main memory latency is 250ns and an STP bus is used. For both 64K and 256K bytes caches, doubling the bus width from 4 bytes to 8 bytes increases the maximum multi throughput by about 50% (at block size of 16 bytes.) Each further doubling of the bus width improves performance less (about 30%). A wider bus decreases the block transfer time and consequently the bus utilization and the queuing delay. The reduced queuing delay and bus transfer time improve the read latency and the throughput of an individual processor; the reduced bus utilization allows more processors to be added to the system. Increasing the bus width appears to be an effective way of improving system performance, but has diminishing returns. It warrants investigation when a system is being designed, just as any other memory system parameter.

While our results on bus choice are not unexpected, we reiterate the need to include all components of the memory system in evaluating any design choices and determining the magnitude of the maximum system processing power. Furthermore, our analytical models allow one to determine quantitatively the magnitude of performance difference between arbitrary design choices in the memory system.



(a) Cache Size: 64K bytes



(b) Cache Size: 256K bytes

Figure 14: Maximum Multi Throughput (in VAX MIPS) for STP Bus with Varying Bus Width

5. SUMMARY AND CONCLUDING REMARKS

We have considered the evaluation of design choices for shared bus multiprocessors (multis) operating in a multi-user, throughput-oriented environment. We developed ''customized'' mean value analysis (CMVA) models for evaluating multis and compared the values of processor throughput obtained from the models with the values obtained from *actual* trace-driven simulation for 5,376 system configurations. Our results indicate that the CMVA models can predict the processor throughput with an error of less than 3% in about 90% of the cases and with an error of less than 5% in almost all cases (99%). This is done with computational requirements that are typically about five orders of magnitude less than that of trace-driven simulation. Therefore, we believe that the CMVA models are a very useful tool in exploring the design space and in evaluating design choices in bus-based, throughput-oriented multiprocessors.

Using our CMVA models and processor memory reference characteristics derived from the widely-used ATUM traces, we evaluated some design choices in the memory system of a multi. We found that a simple consideration of traditional performance metrics (such as miss ratio and bus traffic), independent of the parameters of the shared bus and the main memory, is likely to result in erroneous conclusions. With a circuit switched bus, it is especially important to consider all components of the memory system, including main memory latency. With a split transaction, pipelined (STP) bus, main memory latency is less crucial to maximum multi throughput, but the best block size is neither the one that results in the lowest cache miss ratio nor the one that results in the lowest bus traffic. Also, an STP bus is preferable to a circuit switched bus if the system performance is to be maximized. The performance of an STP bus can be further improved, with diminishing returns, by increasing the bus width.

We also considered the need for set associativity in the caches. Although the importance of set associativity in uniprocessor caches diminishes when the cache size is as large as 256K bytes, we find that set associativity is desirable in multiprocessors even with such large caches. This is because the additional set associativity reduces the per-processor bus bandwidth demand and allows more processors to be connected together in the multi, thereby increasing the maximum multiprocessor throughput.

Our evaluation of design choices is specific to the traces that we use and while we caution the reader against interpreting our results as true in general, we encourage the reader to use mean value analysis models similar to the ones we have considered, customize them to their particular environment, drive them with program characteristics particular to their environment and use the models to evaluate their design choices.

Acknowledgments

This work was supported in part by NSF Grants CCR-8706722 and CCR-8919635. The authors would like to thank Mary Vernon for introducing them to mean value analysis models and encouraging them to consider their use.

References

[1] C. G. Bell, ''Multis: a New Class of Multiprocessor Computers,'' *Science*, vol. 228, pp. 462-467, April 1985.

[2] J. R. Goodman, ''Using Cache Memory to Reduce Processor-Memory Traffic,'' *Proc. 10th Annual Symposium on Computer Architecture*, pp. 124-131, June 1983.

[3] A. J. Smith, ''Line (Block) Size Choice for CPU Cache Memories,'' *IEEE Transactions on Computers*, vol. C-36, pp. 1063-1075, September 1987.

[4] D. Lilja, D. Marcovitz, and P.-C. Yew, ''Memory Reference Behavior and Cache Performance in a Shared Memory Multiprocessor,'' CSRD Report Number 836, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL 61801-2932, December 1988.

[5] A. J. Smith, ''Cache Memories,'' *ACM Computing Surveys*, vol. 14, pp. 473-530, Sept. 1982.

[6] M. D. Hill, ''Aspects of Cache Memory and Instruction Buffer Performance,'' Technical Report UCB/CSD 87/381, University of California at Berkeley, Berkeley, CA, November 1987.

[7] M. D. Hill, ''A Case for Direct-Mapped Caches,'' *IEEE Computer*, vol. 21, pp. 25-40, December 1988.

[8] S. Przybylski, M. Horowitz, and J. Hennessy, ''Performance Tradeoffs in Cache Design,'' *Proc. 15th Annual Symposium on Computer Architecture*, pp. 290-298, June 1988.

[9] P. Borrill and J. Theus, ''An Advanced Communication Protocol for the Proposed IEEE 896 Futurebus,'' *IEEE Micro*, pp. 42-56, August 1984.

[10] B. Beck, B. Kasten, and S. Thakker, ''VLSI Assist for a Multiprocessor,'' *Proc. ASPLOS II*, pp. 10-20, October 1987.

[11] G. N. Fielland, ''Symmetry: A Second Generation Practical Parallel,'' *Digest of Papers, COMPCON Spring 1988*, pp. 114-115, February 1988.

[12] J. H. Patel, ''Analysis of Multiprocessors With Private Cache Memories,'' *IEEE Transactions on Computers*, vol. C-31, pp. 296-304, April 1982.

[13] M. A. Marsan and M. Gerla, ''Markov Models for Multiple-Bus Multiprocessor Systems,'' *IEEE Transactions on Computers*, vol. C-31, pp. 239-248, December 1982.

[14] M. A. Marsan, G. Balbo, G. Conte, and F. Gregoretti, ''Modeling Bus Contention and Memory Interference in a Multiprocessor System,'' *IEEE Transactions on Computers*, vol. C-32, pp. 60-72, January 1983.

[15] T. Lang, M. Valero, and I. Alegre, ''Bandwidth of Crossbar and Multiple-Bus Connections for Multiprocessors,'' *IEEE Transactions on Computers*, vol. C-31, pp. 1227-1234, December 1982.

[16] T. N. Mudge, J. P. Hayes, G. D. Buzzard, and D. C. Windsor, ''Analysis of Multiple Bus Interconnection Networks,'' *Proc. 1984 International Conference on Parallel Processing*, pp. 228-232, August 1984.

[17] S. J. Eggers and R. H. Katz, ''A Characterization of Sharing in Parallel Programs and its Application to Coherency Protocol Evalaution,'' in *Proc. 15th Annual Symposium on Computer Architecture*, Honolulu, HI, pp. 373-382, June 1988.

[18] J. Archibald and J. -L. Baer, ''Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model,'' *ACM Transactions on Computer Systems*, vol. 4, pp. 273-298, November 1986.

[19] M. K. Vernon and M. Holliday, ''Performance Analysis of Multiprocessor Cache Consistency Protocols Using Generalized Timed Petri Nets,'' *Proc. SIGMETRICS International Symposium on Computer Performance Modeling, Measurement and Evaluation,* pp. 9-17, May 1986.

[20] M. K. Vernon, E. D. Lazowska, and J. Zahorjan, ''An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols,'' in *Proc. 15th Annual Symposium on Computer Architecture*, Honolulu, HI, pp. 308-315, June 1988.

[21] S. Leutenegger and M. K. Vernon, ''A Mean-Value Performance Analysis of a New Multiprocessor Architecture,'' *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1988.

[22] R. Jog, G. S. Sohi, and M. K. Vernon, ''The TREEBus Architecture and Its Analysis,'' Computer Sciences Technical Report #747, University of Wisconsin-Madison, Madison, WI 53706, February 1988.

[23] M. K. Vernon, R. Jog, and G. S. Sohi, ''Performance Analysis of Hierarchical Cache-Consistent Multiprocessors,'' *Performance Evaluation*, vol. 9, pp. 287-302, 1989.

[24] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance, Computer System Analysis Using Queueing Network Models.* Englewood Cliffs, New Jersey: Prentice Hall, 1984.

[25] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, and R. G. Sheldon, ''Implementing a Cache Consistency Protocol,'' *Proc. 12th Annual Symposium on Computer Architecture,* pp. 276-283, June 1985.

[26] M.-C. Chiang and G. S. Sohi, ''Experience with Mean Value Analysis Models for Evaluating Shared Bus, Throughput-Oriented Multiprocessors,'' *Proc. SIGMETRICS International Symposium on Computer Performance Modeling, Measurement and Evaluation*, May 1991.

[27] A. Agarwal, R. L. Sites, and M. Horowitz, ''ATUM: A New Technique for Capturing Address Traces Using Microcode,'' in *Proc. 13th Annual Symposium on Computer Architecture*, Tokyo, Japan, pp. 119-127, June 1986.

[28] A. Agarwal, M. Horowitz, and J. Hennessy, ''An Analytical Cache Model,'' *ACM Transactions on Computer Systems*, vol. 7, pp. 184-215, May 1989.

[29] A. Agarwal, J. Hennessy, and M. Horowitz, ''Cache Performance of Operating Systems and Multiprogramming Workloads,'' *ACM Transactions on Computer Systems*, vol. 6, pp. 393-431, November 1988.

APPENDIX

THE CMVA MODEL FOR AN STP BUS

We use the following additional notation for an STP bus:

Input Parameters

- T_m is the main memory latency.
- T_{qo} is the bus access time of a read request, excluding the bus arbitration time.
- T_{do} is the bus access time of a block transfer either for a cache write back or a main memory reply, excluding the bus arbitration time.

Output Parameters

- W_{qv} is the mean bus waiting time of a read or an invalidation request.
- W_d is the mean bus waiting time of a main memory reply.
- T_q is the bus access time of a read operation, including the bus arbitration time.
- T_d is the bus access time of a write back or a main memory reply, including the bus arbitration time.
- U_q denotes the partial utilization of the bus by the read requests from one cache.
- U_d denotes the partial utilization of the bus by the main memory replies to one cache.
- U_r denotes the partial utilization of the bus by the reads, invalidations from, and the main memory replies to one cache.
- $\bar{Q}_q$ denotes the mean number of read requests from the same cache in the bus.
- $\bar{Q}_d$ denotes the mean number of main memory replies to the same cache in the bus.
- B_q (B_v, B_w) is the probability that the bus is busy servicing a read (invalidation, write back) request from a particular cache, when a new read, invalidation, or main memory reply arrives.
- B_d is the probability that the bus is busy servicing a main memory reply to a particular cache, when a new read, invalidation, or main memory reply arrives.
- Re^q is the residual transfer time of a read request when the request is serviced by the bus and a new read, invalidation, or main memory reply arrives.
- Re^d is the residual transfer time of a main memory reply when the memory reply is serviced by the bus and a new read, invalidation, or main memory reply arrives.
- K_r^q (K_r^v, K_r^d) is the mean bus waiting time of a read request, an invalidation from, or a main memory reply, due to the read (invalidation, main memory reply) requests already in the bus.

- K_{qv}^{w} is the mean bus waiting time of a read request or an invalidation, due to the writes back already in the bus.
- K_{d}^{w} is the mean bus waiting time for a main memory reply, due to the writes back already in the bus.
- K_{w}^{d} is the mean bus waiting time of a write back request, due to the main memory replies already in the bus.

Response Time Equations

The response time equations of the CMVA model for an STP bus can be derived in a way similar to that for a circuit switched bus.

$$R = T_e + Rs_r + Rs_v$$

$$Rs_r = P_r\,(W_{qv} + T_q + T_m + W_d + T_d)$$

$$Rs_v = P_v\,(W_{qv} + T_v)$$

where

$$T_x = (1 - \frac{NU - U_r}{1 - U_r}) \times T_a + T_{xo} = \frac{1 - NU}{1 - U_r} \times T_a + T_{xo}; \quad x = q, v, d$$

$$T_w = (1 - \frac{(N-1)\,U}{1 - U}) \times T_a + T_{do} = \frac{1 - NU}{1 - U} \times T_a + T_{do}$$

Waiting Time Equations

The waiting time equations for an STP bus are more complicated than those for a circuit switched bus because there are four kinds of requests in the system: a cache can generate read, write and invalidation requests, and the main memory can generate replies in response to read requests. An arriving request can see all four kinds of requests in the bus queue, hence its average waiting time consists of four components.

$$W_{qv} = K_r^q + K_r^v + K_r^d + K_{qv}^w$$

$$K_r^x = (N-1)\left[(\bar{Q}_x - B_x) \times T_x + B_x \times \mathrm{Re}^x\right]; \quad x = q, v, d$$

$$K_{qv}^w = N\left[(\bar{Q}_w - B_w) \times T_w + B_w \times \mathrm{Re}^w\right]$$

$$W_d = K_r^q + K_r^v + K_r^d + K_d^w$$

$$K_d^w = (N-1)\left[(\bar{Q}_w - B_w) \times T_w + B_w \times \mathrm{Re}^w\right]$$

$$W_w = K_w^q + K_w^d + K_w^v + K_w^w$$

$$K_w^x = (N-1)\,\bar{Q}_x\,T_x; \quad x = q, d, v, w$$

The multiplication factor for K_{qv}^w is N and for K_d^w is $(N-1)$ because an arriving read or invalidation request may see a write back request from the same cache in the bus, whereas a main memory reply destined for a particular cache will never see a write back from the same cache on the bus. The equations for the residual service time of the request that is currently being serviced, when a new read or invalidation request from some cache, or a reply from main memory arrives, are

$$\mathrm{Re}^x = \frac{T_x}{2}; \quad x = q, v, d, w$$

The remaining equations are

$$B_x = \frac{U_x}{1 - U_r}; \quad x = q, v, d, w$$

$$\overline{Q}_q = \frac{P_r\,(W_{qv} + T_q)}{R}$$

$$\overline{Q}_v = \frac{P_v\,(W_{qv} + T_v)}{R}$$

$$\overline{Q}_d = \frac{P_r\,(W_d + T_d)}{R}$$

$$\overline{Q}_w = \frac{P_r\,P_w\,(W_w + T_w)}{R}$$

$$U_x = \frac{P_r\,T_x}{R}; \quad x = q, d$$

$$U_v = \frac{P_v\,T_v}{R}$$

$$U_w = \frac{P_r\,P_w\,T_w}{R}$$

$$U_r = U_q + U_d + U_v = \frac{P_r\,(T_q + T_d) + P_v\,T_v}{R}$$

$$U = U_q + U_d + U_v + U_w = \frac{P_r\,(T_q + T_d) + P_v\,T_v + P_r\,P_w\,T_w}{R}$$