A Mean-Value Performance Analysis of a New Multiprocessor Architecture †

Scott T. Leutenegger
Mary K. Vernon

Computer Sciences Technical Report #748

February 1988

† To appear in *Proc. 1988 ACM SIGMETTRICS Conf. on Measurement and Modeling of Computer Systems* Santa Fe, New Mexico, May 24-27, 1988.

A Mean-Value Performance Analysis of a New Multiprocessor Architecture†

Scott T. Leutenegger
Mary K. Vernon

Department of Computer Science
University of Wisconsin - Madison
1210 Dayton Street
Madison, WI 53705

Abstract

This paper presents a preliminary performance analysis of a new large-scale multiprocessor: the *Wisconsin Multicube*. A key characteristic of the machine is that it is based on shared buses and a snooping cache coherence protocol. The organization of the shared buses and shared memory is unique and non-hierarchical. The two-dimensional version of the architecture is envisioned as scaling to 1024 processors.

We develop an approximate mean-value analysis of bus interference for the proposed cache coherence protocol. The model includes FCFS scheduling at the bus queues with deterministic bus access times, and asynchronous memory write-backs and invalidation requests.

We use our model to investigate the feasibility of the multiprocessor, and to study some initial system design issues. Our results indicate that a 1024-processor system can operate at 75 - 95% of its peak processing power, if the mean time between cache misses is larger than 1000 bus cycles (i.e. 50 microseconds for 20 MHz buses; 25 microseconds for 40 MHz buses). This miss rate is not unreasonable for the cache sizes specified in the design, which are comparable to main memory sizes in existing multiprocessors. We also present results which address the issues of optimal cache block size, optimal size of the two-dimensional Multicube, the effect of broadcast invalidations on system performance, and the viability of several hardware techniques for reducing the latency for remote memory requests.

1. Introduction

Multiprocessors are emerging as an important class of computer architectures. These machines include multiple CPUs, each having some amount of local memory, which are connected through a high-speed interconnect to shared memory and I/O devices. The design of a multiprocessor is complex, and prototyping the system is expensive. Thus, this class of systems represents an important opportunity for performance models to contribute to the actual design process.

The *"Wisconsin Multicube"* is a new large-scale multiprocessor architecture [GoWo87]. The two-dimensional version of this architecture is envisioned as scaling to 1024 processors. A key characteristic of the machine is that each processor has a very large cache, comparable in size to the size of main memory on existing machines. Another key characteristic of the machine is that it is based on shared buses and a snooping cache

† This work was supported by the National Science Foundation under grant number DCR-8451405, and by grants from Cray Research, INC. and the AT&T Foundation.

coherence protocol. That is, copies of shared data blocks that reside in caches local to the system processors are kept consistent by hardware that monitors the bus traffic. This greatly simplifies the job of the programmers and/or system compilers, since they need not be concerned with explicitly mapping data to the various local and shared memory modules for efficient execution.

The organization of the shared buses and shared memory in the *Wisconsin Multicube* architecture is unique and non-hierarchical. Issues that must be considered in the preliminary design of the system include the required bandwidth of the system buses, the optimal blocksize for transferring data between shared memory and the caches, and the viability of various hardware techniques for reducing remote memory access latency.

This paper presents a performance model we have used to study the feasibility and several design issues for the *Wisconsin Multicube*. We develop an approximate mean-value analysis to study the effects of bus interference for a machine with up to 1024 (i.e. 32×32) processors. The model assumes first-come first-served service on the buses, and deterministic bus access times. Asynchronous invalidation and memory write-back requests are also modeled. Parameters of the model include system size, mean time between bus requests, estimates of the workload-dependent frequency and type of shared data accesses, bus speed, and block transfer size.

Initial results indicate that a 1024-processor Multicube can operate at 75 - 90% of its peak processing power if the mean time between bus requests is larger than 1000 bus cycles (50 microseconds for 20 MHz buses). In addition, the model results indicate that, strictly from a performance perspective, cache block size should be on the order of 16 - 32, and no larger than 64 times the width of the data bus. Furthermore, our results show that nearly linear speedups are possible for up to 1024 processors, for miss rates below one per 2000-4000 bus cycles. Additional results that address the effects of broadcast invalidations, and viability of hardware techniques for reducing remote memory request latency, are also presented.

The accuracy of the model relative to the real system is unknown. However, the level of detail in the model is compatible with the level of detail known about the real system at this time. Furthermore, the modeling approach has been shown to be remarkably accurate for evaluating cache coherence protocols for single-bus multis [LaVZ88]. The level of detail in the model is comparable to the level of detail in queueing network models used in capacity planning for uniprocessor systems. We believe that this level of detail is also appropriate for initial design and capacity planning for multiprocessor systems.

The organization of the remainder of this paper is as follows. We describe the architecture and cache coherence protocol of the *Wisconsin Multicube* in section 2. In section 3, we develop the approximate mean-value model and discuss the accuracy of the model. Section 4 contains some results of the mean-value analysis that are useful for assessing the feasibility of *Wisconsin Multicube* and for selecting some initial design parameters. Section 5 contains the conclusions of this study.

2. The Wisconsin Multicube Architecture

The *Wisconsin Multicube* is based on a grid of system buses, as depicted for the two-dimensional case in Figure 2.1. A processor is located at each grid point. The processor has a large DRAM cache with a snooping controller connected to the row bus and column bus that define the grid point. Note that the processor has another level of cache memory between it and the snooping cache. This level of cache is provided to support high-speed processors, and is not otherwise a focal point for the model developed in this paper. Global shared memory is distributed across the column buses. Each memory block address has a *home column*, which is the column bus connected to the

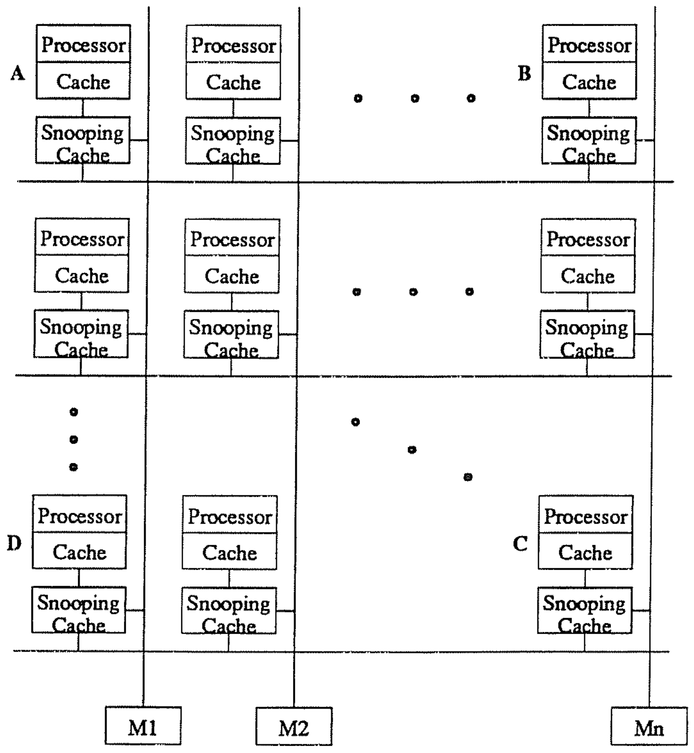

Figure 2.1: The *Wisconsin Multicube* Memory and Bus Organization

global memory containing the block.

2.1. The Cache Coherence Protocol

An efficient snooping cache protocol is currently being designed for the grid of buses [GoWo87]. The goal of the protocol is to maintain copies of shared memory blocks residing in the snooping caches consistent. The protocol is continually being revised as trade-offs between protocol complexity and system performance are evaluated. Below we describe the protocol as defined in our model of the bus request traffic.

There are three cache block states: *modified*, *shared*, and *invalid*. Blocks in state *modified* have been written by the processor, and have not been read or written by any other processor since the write operation occurred. If a block is in state *modified*, no other cache has a copy. Each snooping cache controller maintains a tag cache that contains the addresses of all data blocks that are in state *modified* in any cache on its column.

We will use the term "unmodified" to denote the *global* state of a cache block that does not exist in any cache in state *modified*. The global state, "modified", denotes a block that exists in some cache in state *modified*. Global shared memory contains the current value of all unmodified cache blocks.

We have identified four processor nodes in Figure 2.1 (i.e. processors A, B, C, and D) to aid in the following description of the protocol. Bus requests for write misses are *READ-MOD* (i.e. *read-with-the-intent-to-modify*) operations; bus requests for read misses are *READ* operations.

A processor memory request that misses in the cache is broadcast on the processor's row bus. The bus request is treated differently, depending on whether the block is in global state modified or not. For the first of these two cases, assume that processor A in Figure 2.1 makes the request that misses in its cache, and processor C contains the modified data block. In this case, the request on processor A's row bus is intercepted by processor B's cache controller, which has the address of the block in its tag cache. Cache B signals a modified match on the row bus, and then transmits the request on its column bus. All nodes on cache B's column respond by removing the address from their tag cache. Cache C also transmits the requested data block on its *row* bus, together with a destination address for processor A's cache. Finally, cache D forwards the data from the row bus to its column bus.

If the bus request is a *READ* operation, then the state of the block changes to *shared* in cache C, and global memory must be updated. In this case, the home column cache controller reads the data when it is transmitted on the row bus, and performs the required write-back to global memory. If, on the other hand, the bus request is a

READ-MOD operation, then cache C invalidates its copy of the block, and the caches in the processor A's column add the address of the block to their tag caches when the data is transmitted on the column bus. Note that the row request is not needed, and the row data transfer is not needed for a *READ-MOD* request, if the modified data block is in the same column as the requesting processor (e.g., cache D contains the data requested by A). Similarly, the column request is not needed, and the column data transfer can be replaced by a tag-cache update request, if the modified data block resides in a cache on the same row as the requesting processor (e.g. cache B contains the data requested by cache A). We have modeled these optimizations.

Next consider the case where the request that misses in the cache is for an unmodified data block. Assume that processor A makes the memory request, and processor B is attached to the home column for the data block. The cache for processor A broadcasts the memory request on its row. When no cache on the row signals a modified match, (i.e. no column contains the data block in state *modified*), processor B's cache forwards the request to the memory on its column bus. The shared memory module on the column responds by placing the requested data block on the column bus. Cache B then forwards the data along its row back to processor A. Note that a request for an unmodified block with a home column the same as the column of the requesting processor requires the initial row bus request (to determine that the block is unmodified), but does not require the final row bus data transfer.

The difference between data fetches for modified and unmodified blocks lies in whether the data returns along the same path that the request travels, or whether the return path uses a distinct row and column bus.

In addition to the above bus traffic, some memory requests require the propagation of invalidation signals on the row buses. Specifically, a write request for an unmodified block requires the propagation of an invalidation signal on each row bus. For write requests that miss in the cache, the invalidation signal on the cache's own row bus is implicit in the *READ-MOD* request that is broadcast on the row. Invalidation operations on the other row buses are triggered by the *READ-MOD* request that is broadcast on the home column bus for the data block. Each cache on a column, upon seeing a *READ-MOD* request to the main memory, forwards an invalidation signal on its row bus.

A processor write to a data block that is in state *shared* in its cache generates an explicit invalidation request on the processor's row bus. The invalidation row request is intercepted by the controller on the home column for the data block, and forwarded on the home column bus. Each cache on the home column then forwards the invalidation request on its row bus. For the purposes of simplifying the model in this paper, we assume that invalidations are only generated by write requests that miss in the cache. Invalidations generated by write requests that hit in the cache can be approximately represented in our model by increasing the probability of *READ-MOD* requests to include *READ* requests for blocks that will be written before they are purged from the cache.

2.2. Protocol Assumptions

Several details and minor trade-offs in the cache protocol are not considered in the model we develop in Section 3. First, if two requests for the same data block are issued on the buses within a very short period of time, and one of the requests is a *READ-MOD*, one of them may be delayed while the other one is satisfied. We have not considered this delay, which we expect to happen very rarely in the operation of the system.

Second, the protocol described above contains several optimizations that increase the protocol complexity. Those optimizations that don't yield significant performance benefit, (and thus have negligible impact on the results of our model), will not be included in the final protocol design.

Third, interference in the cache and global memories is not accounted for in the model. Experience with single-bus multiprocessor models indicates that the interference in the cache is a secondary effect when compared with bus contention.

Finally, a change in the protocol since the model was developed has resulted in routing responses for *READ* requests for modified data blocks back along the column and row which the requests travel, rather than in the rectangular path described above. We have determined that this change has a negligible effect on the results presented in Section 4.

2.3. System Assumptions

The current design of the *Wisconsin Multicube* specifies a cache size of 16-64 Megabytes. Thus, we assume that most cache misses are for shared data, or for I/O, which can be treated as a special case of shared data. In other words, a bus request is typically the first request a processor makes to a cache block, or a request for a cache block that has recently been modified by another processor. Thus, we will assume very low miss rates are possible for many application workloads.

Cache miss rates, the fraction of misses that are writes, and the fraction of misses that exist in state *modified* in another cache, are workload-dependent values that are specified as parameters in our model.

We assume an $N \times N$ system with uniform memory access probabilities for both modified and unmodified blocks. Thus, a request for a modified block is equally likely to be satisfied by any of the other N^2-1 caches, and a requested unmodified data block is equally likely to be have any column bus as its home column.

We further assume that bus speeds may be different for the row and the column buses. This is primarily due to the physical construction of the machine. Our model will allow for different access times on the row and column buses, respectively. However, the results presented in Section 4 assume that both buses have the same speed.

Finally, we assume that bus access times are deterministic and depend on whether the access is a request (i.e. an address) or a cache block transfer. The bus access time for cache block transfers further depends on the size of the block, the width of the bus, and the bus speed. These three parameters are reduced to a single parameter, the cache block transfer time, in the model.

3. The Mean-Value Model

In this section we present an approximate mean value model for estimating the processing power of the *Wisconsin Multicube* system as a function of various design and workload parameters. The model contains three approximations relative to the exact mean value analysis equations for a product-form queueing network (PFQN) model of the system. The first approximation removes the recursion in the waiting time equations, allowing us to solve the equations for large systems [Schw79]. The other two approximations are extensions to the equations for non-product form features. We use the extensions for the FCFS queueing discipline and deterministic service times developed by Reiser [Reis79], and the extension for the spawning of asynchronous tasks proposed by Heidelberger and Trivedi [HeTr82].

The model is based on computing the mean response time for a processor "cycle", which includes some time productively executing in its local cache, and the mean response time for retrieving the remote data block. The response time for the remote memory request includes mean bus queueing delays, bus access times and the remote memory latency. Note that interference in the cache and global memories is not accounted for in the model, as discussed in Section 2.

Bus queueing delays are based on approximate estimates of the mean queue lengths, by bus request type, seen by an arriving request. These queue length estimates are cyclically dependent on the mean cycle response time. The set of equations are solved iteratively until each of the waiting time estimates agrees with the estimate of the previous iteration, to within some small value, ε.

In Sections 3.1 through 3.4 we develop the approximate mean value model. In Section 3.5 we comment on the accuracy of the model, and on the effects of the three model features that violate product form assumptions.

The following notation is used in the remainder of this section:

- *resource type* is denoted by subscript i, which can take on any value in the set $\{r, c\}$, representing row and column buses, respectively.
- *customer class* is denoted by subscript j, and can take on any value in the set $\{o, f\}$, representing customers that are using their "own" bus or a "foreign" bus, respectively.
- *bus request type* is denoted by subscript k, and can take on any value in the set $\{A, D, AD, I, WB\}$, representing "address", cache block transfer, a combined address and cache block transfer, invalidation, and memory write-back, respectively.
- R denotes the mean response time for one processor "cycle", in which a processor executes locally for some amount of time until a cache miss occurs, and then the processor waits for the requested cache block to be retrieved from a remote cache or global memory.
- $R_{i,j,k}$ is the expected value of the time during each cycle that a processor is queued for a type i bus as a class j customer requiring a type k access, including the service time.
- $D_{i,j,k}$ is the expected value of the time during each cycle that a processor uses a type i bus as a class j customer for a type k access.
- $U_{i,j,k}$ denotes the utilization of a type i bus by requests of class j and type k.
- $\bar{Q}_{i,k|j}$ denotes the mean number of type k requests in the queue for a type i bus when a class j request arrives.
- $\beta_{i,k|j}$ denotes the probability that a type i bus is busy serving a type k access when a class j request arrives.
- $W_{i,j}$ is the mean waiting time at resource i for a customer of class j, excluding time in service.

In addition, we use the following symbols to denote input parameters for the mean-value model:

- t_p is the mean processing time between cache misses.
- p_s and p_x are the probabilities that a cache miss is for an unmodified or modified block, respectively. (Note that $p_s = 1 - p_x$).
- $p_{read-mod}$ is the probability that the cache miss is a write request.
- $t_{k,i}$ is the bus access time for requests of type k on a bus of type i. For example, $t_{A,c}$ denotes the time to transmit an address on a column bus. Note that $t_{AD,i} = t_{A,i} + t_{D,i}$.
- d_{mem} is the global memory latency.
- d_{cache} is the cache memory latency.

Note that the model allows for $t_{k,r} \neq t_{k,c}$. This allows for different bus speeds and/or bus widths for the row and column buses.

3.1. Response Time Equations

Using the above notation, the basic equation for response time is as follows:

$$R = t_p + R_x + R_s,$$

where R_x (R_s) denotes the mean response time for remote request for a modified (unmodified) data block, weighted by the probability that the cache miss requests a modified (unmodified) block. The calculation of R_s and R_x is discussed next.

R_s is equal to the probability the block is unmodified (p_s) times the mean time to retrieve the block from global memory. The time to retrieve the block depends on whether or not the block is in the memory module on the column where the request originates. In either case the address is first placed on the processor's row bus to make sure that the data block is not modified on another column. The time for this operation is $W_{r,o} + t_{A,r}$. The probability that the data block is in the memory on the same column is $\frac{1}{N}$. With this probability, the mean time for the remainder of the remote request is $(W_{c,o} + t_{AD,c} + d_{mem})$. The terms in this expression are the mean wait for the column bus, the access time for an address and data transfer on the column bus, and the memory latency. Similarly, with probability $\frac{(N-1)}{N}$, the time for the remainder of the remote request is the delay for retrieving the block from another column, or $(W_{c,f} + t_{AD,c} + d_{mem} + W_{r,o} + t_{D,r})$. In both cases, the column bus accesses involve two transfers, one for the address and one for the data. However these two transfers are treated as one operation, since only one bus waiting time is required. The equation for R_s is thus as follows:

$$R_s = p_s \left[W_{r,o} + t_{A,r} + (\frac{1}{N})(W_{c,o} + t_{AD,c} + d_{mem}) + (\frac{N-1}{N})(W_{c,f} + t_{AD,c} + d_{mem} + W_{r,o} + t_{D,r}) \right].$$

The response time for R_x is derived similarly. The expression is (probability data block is modified) × { (probability data on same column) × (mean time to get data from processor on column) + (probability data not on column) × [(mean time for row request) + (probability on same row) × (mean time to return data from row) + (probability not on same row) × (mean time to finish the column request, shared memory access, and column and row data transfers)] }. The equation for R_x is thus [1] :

$$R_x = p_x \left\{ \left(\frac{N-1}{N^2-1}\right)(W_{c,o} + t_{A,c} + d_{cache} + W_{c,o} + W_{D,c}) + \left(\frac{N^2-N}{N^2-1}\right) \left[W_{r,o} + t_{A,r} + \left(\frac{1}{N}\right)(d_{cache} + W_{r,o} + t_{D,r}) \right. \right.$$
$$\left. \left. + \left(\frac{N-1}{N}\right)(W_{c,f} + t_{A,c} + d_{cache} + W_{r,f} + t_{D,r} + W_{c,o} + t_{D,o}) \right] \right\}.$$

3.2. Bus Waiting Times

Given the response time equations in the previous section, only the expected values of the row and column bus waiting times remain to be computed. Below we derive mean bus waiting times, ignoring bus invalidation and memory write-back requests, assuming a FCFS queueing discipline and deterministic bus access times. Extensions to include the asynchronously spawned invalidation requests are discussed in section 3.3. Derivation of the waiting time equations for the PS discipline follows a similar approach, yielding the equations summarized in Appendix B.

For the FCFS queueing discipline, the mean bus waiting time is a function of the mean number of requests, per request type, found in the bus queue when a request arrives, and the mean time to service each request type. Note that mean queue lengths seen by class o and class f arrivals at a type i bus will differ, since there are fewer class o processors generating requests for the bus, and since class o processors may generate different types of requests for the type i bus than class f processors. On the other hand, mean queue lengths seen by arrivals will be the same for all requests within either of these classes, since we are assuming uniform memory access probabilities. Thus, for the FCFS scheduling discipline, we need to compute the mean arrival queue length, per request type, conditioned only on the class of the arriving request.

We begin by re-writing the equations for R_s and R_x such that the terms for the mean time each request type spends in each of the bus queues are more readily identified. To this end, we use the following notation:

- $p_{k,i,j|S}$ is the probability that the processor requires a type k access on a bus of type i as a customer of class j, given that the cache miss is for an unmodified block. $k \in \{A, AD, D\}, i \in \{r, c\}, j \in \{o, f\}$
- $p_{k,i,j|X}$ is the probability that the processor requires a type k access on a bus of type i as a customer of class j, given that the cache miss is for a modified block.

These conditional bus transaction probabilities are readily computed by examining the equations for R_s and R_x above. For example, $p_{AD,c,o|S} = \frac{1}{N}$, and $p_{AD,c,f|S} = \frac{(N-1)}{N}$. Computing the conditional bus transaction probabili-

[1] Note that the model is missing a term for transmitting the data on the processor's row when a READ request is supplied by a cache in the processor's column, and a term for transmitting the tag-cache update request on the processor's column when a READ-MOD request is satisfied by a cache in the processor's row. At this point in time, it is not specified whether these bus operations are performed before supplying the data to the processor. Since they occur relatively infrequently, and since the tag-cache update request is just one bus cycle, omitting this traffic from the model should have a negligible effect.

ties for requests to modified blocks is equally straightforward, but somewhat more complicated, since it is sometimes necessary to sum the probabilities for more than one term in equation for R_x. The bus transaction probabilities are summarized in Appendix A. Note that 14 of the 24 probabilities evaluate to 0.

Using the conditional bus transaction probabilities, the equation for R_s can be rewritten as follows:

$$R_s = p_s \left\{ p_{A,r,o|s}(W_{r,o} + t_{A,r}) + p_{AD,c,o|s}(W_{c,o} + t_{AD,c} + d_{mem}) + p_{AD,c,f|s}(W_{c,f} + t_{AD,c} + d_{mem}) + p_{D,r,o|s}(W_{r,o} + t_{D,r}) \right\},$$

and the equation for R_x can be rewritten as:

$$R_x = p_x \left\{ p_{A,c,o|X}(W_{c,o} + t_{A,c}) + p_{D,c,o|X}(W_{c,o} + t_{D,c}) + p_{A,r,o|X}(W_{r,o} + t_{A,r}) + p_{D,r,o|X}(W_{r,o} + t_{D,r}) \right.$$
$$\left. + p_{A,c,f|X}(W_{c,f} + t_{A,c}) + p_{D,r,f|X}(W_{r,f} + t_{D,f}) + d_{cache} \right\}.$$

From these equations, we can rewrite the overall response time equation as follows:

$$R = t_p + p_s\, d_{mem} + p_x\, d_{cache} + \sum_{\substack{i \in \{r,c\} \\ j \in \{o,f\} \\ k \in \{A,D,AD\}}} R_{i,j,k}, \tag{1}$$

where:

$$R_{i,j,k} = (p_x\, p_{k,i,j|X} + p_s\, p_{k,i,j|S})\,(W_{i,j} + t_{k,i}). \tag{2}$$

Equation (2) can be interpreted as follows. The mean time per cycle that a processor is queued as a class j request at a type i bus for a type k access, is the probability that a request of this class and type is generated by the cache miss request, times the sum of the mean waiting time and the access time for the request.

The steady state fraction of time a processor is queued for a type i bus as a request of class j requiring a type k access, is given by: $\frac{R_{i,j,k}}{R}$. The additional information needed to compute mean bus waiting times is the fraction of time a processor of class j uses a bus of type i for an access of type k. This is readily computed by considering the mean time per cycle that the type class j processor uses the bus of type i for an access of type k:

$$D_{i,j,k} = (p_x\, p_{k,i,j|X} + p_s\, p_{k,i,j|S})\, t_{k,i}. \tag{3}$$

Equation (3) is analogous to equation (2). It is the product of the probability that a class j cache miss generates a request of type k for a type i bus, and the access time for the type k request on the type i bus.

From the $D_{i,j,k}$'s, we can compute mean bus utilizations by request class and type, as follows:

$$U_{i,j,k} = N \times \frac{D_{i,j,k}}{R}. \tag{4}$$

To see this, first consider the values $U_{i,o,k}$. Note that N processors may generate class o requests for a particular type i resource. Thus, the total utilization by class o requests is N times the fraction of time each processor uses its own type i bus for type k requests. For requests of class f, there are N $(N-1)$ processors that may generate requests of type k for a type i bus. However the probability that a processor selects a particular foreign bus of type i is $\frac{1}{(N-1)}$. Thus, the total utilization of a particular type i bus by class f requests is also N times the fraction of time each processor uses a type i foreign bus.

We also need the utilizations defined in the following equations:

$$U'_{i,j} = U_{i,j,A} + U_{i,j,D} + U_{i,j,AD}, \tag{4a}$$

$$U''_{i,k} = U_{i,o,k} + U_{i\,f,k}, \tag{4b}$$

$$U_i = U'_{i,o} + U'_{i\,f}. \tag{4c}$$

We now have all of the values needed to compute mean bus waiting times. We first consider class o requests. The mean number of type k requests at a type i bus, as seen by an arriving request of class o, is the mean number generated by all class o processors excepting the processor that generated the arrival, plus the mean number generated by all foreign processors:

$$\bar{Q}_{i,k|o} = (N-1)\,\frac{R_{i,o,k}}{R} + N\,\frac{R_{i\,f,k}}{R}. \tag{5a}$$

Note again that the number of foreign processors generating requests for a type i bus is N $(N-1)$, and each sends a fraction of $\frac{1}{N-1}$ of its type k foreign requests to a particular bus of type i.

The probability that the bus of type i is busy serving a type k request when a class o request arrives is approximated by the following equation, which is the ratio of the steady state fraction of time the bus is in use by type k requests not generated by the processor that generated the arriving request, to the fraction of time available for serving requests not generated by the arriving request's processor:

$$\beta_{i,k|o} = \frac{U''_{i,k} - \frac{1}{N} U_{i,o,k}}{1 - \frac{1}{N} U'_{i,o}}. \qquad \textbf{(5b)}$$

Similar reasoning leads to the following equations for the class f requests:

$$\bar{Q}_{i,k|f} = N \frac{R_{i,o,k}}{R} + \left[N (N-1) - 1\right] \left[\frac{1}{N-1}\right] \frac{R_{i,f,k}}{R}, \qquad (6a)$$

and

$$\beta_{i,k|f} = \frac{U''_{i,k} - \frac{1}{N (N-1)} U_{i,f,k}}{1 - \frac{1}{N (N-1)} U'_{i,f}}. \qquad (6b)$$

From these values, the mean waiting time for a class j request at a resource of type i is finally computed as follows [Reis79]:

$$W_{i,j} = \sum_{k \in A,D,AD} \left\{ (\bar{Q}_{i,k|j} - \beta_{i,k|j})\, t_{k,i} + \beta_{i,k|j} \frac{t_{k,i}}{2} \right\}. \qquad (7)$$

3.3. Extensions to the Waiting Time Equations for Asynchronous Invalidations and Write-Backs

Simple extensions to the mean value model are required for including the asynchronous invalidation traffic generated by *READ-MOD* requests for shared blocks, and the asynchronous memory write-backs generated by *READ* requests for modified blocks. The extension for invalidations (write-backs) is an additional term for $k = I$ ($k = WB$) in the sum for $W_{r,j}$ ($W_{c,j}$). The additional terms represent the waiting time for invalidation traffic on the row bus, and the waiting time for write-back traffic on the column bus. Applying similar reasoning as in the derivation of equations (5a) and (5b), or (6a) and (6b), gives the following equations for the k=I term in $W_{r,o}$:

$$\bar{Q}_{r,I|o} = N (N-1) \left(\frac{p_{read\text{-}mod} \times p_s \times (W_{r,f} + t_{I,r})}{R}\right), \qquad (8a)$$

and

$$\beta_{r,I|o} = U_{r,I}, \qquad (8b)$$

where:

$$U_{r,I} = N (N-1) \times \frac{p_{read\text{-}mod} \times p_s \times t_{I,r}}{R}.$$

Analogous terms can be derived for k=I in $W_{r,f}$ and for k=WB in $W_{c,o}$ and $W_{c,f}$. Note also that equations (4) and (4a) need to be appropriately modified.

3.4. Calculation of System Efficiency and Processing Power

Various system performance measures can be computed from the mean value equations. The measures we are most interested in are the equivalent measures of system efficiency and system processing power. We define efficiency to be the fraction of time that each processor is productively executing, including the time for data accesses in its caches. This estimate is computed by dividing the mean time between cache misses by the mean cycle response time. Thus, efficiency = $\frac{t_p}{R}$. Processing power is $N^2 \times \frac{t_p}{R}$.

3.5. Accuracy of the Mean Value Model

The purpose of the mean-value model derived in sections 3.2-3.3 is to aid in the design of a system prototype. Thus, we do not have a system that can be measured to validate the accuracy of the equations. However, the general approach has been shown to be as accurate as more detailed models for evaluating the performance of cache coherence protocols for single-bus systems [LaVZ88]. Furthermore, we have compared the results of the approximate mean value model modified to assume PS queueing for the system buses and excluding asynchronous traffic, with the exact solution of an equivalent Product Form queueing network (PFQN). This gives an indication of the error in the approximate model due to the removal of the recursion in the exact equations. Note that the complexity of the PFQN increases considerably as we increase the size of the system, so the two approaches can only be compared for small values of N (say $N \le 5$).

The approximate and exact solutions were compared for a range of input parameters. Appendix C shows the PFQN model for $N=3$, discusses the parameter values, and gives the computed minimum, maximum, and mean relative errors in processing power computed in the experiment. For all workloads investigated, the ratio of percent error to bus utilization was constant. The maximum relative error observed in the experiment was 5%, which occurred for bus utilizations of 65%. We find in Section 4 that reasonable processing power is attained in the *Wisconsin Multicube* only when the bus utilizations are below 65%.

We also examined the effects of each of the three non-product form features in the estimates of the approximate mean-value model. Our results are similar to the results for a similar model of single-bus snooping cache protocols reported in [LaVZ88]. For a 32×32 system, results indicate that the assumptions of deterministic bus access times and FCFS scheduling yield system efficiency estimates that are as much as 5% higher than the estimates for the PS discipline. The effect of asynchronous bus invalidations is even more substantial. As the mean time between bus requests decreases, system efficiency degrades by as much as 25% relative to the PS model without invalidations. Note, however, that at system efficiencies above 0.9, the degradation due to invalidations is less than 5%. The effect of invalidation traffic is explored further in section 4.

4. Model Results

In this section we consider the feasibility and the design of the proposed architecture. We perform several parametric studies to evaluate the effects of system size, cache miss rate, cache block size, and workload sharing parameters on system performance.

Cache miss rate and cache block size are two key parameters that affect system performance. These two parameters are related, but their precise relationship is unknown for the system we are investigating. Thus, in these initial experiments we choose to vary the miss rate and cache block size independently, and then discuss the implications of the results in light of known bounds on the relationship between the two quantities. Section 4.2 presents upper bounds on system processing power for a 32×32 Multicube, as a function of cache miss rate and cache block size. Section 4.3 then presents estimates of system processing power derived from the mean value model, for 32×32, 20×20, and 10×10 systems, also as a function of cache miss rate and cache block size. In section 4.4, we plot processing power as a function of system size for several miss rates and two possible cache block sizes.

The experiments in sections 4.2-4.4 hold $p_x = 0.2$, and $p_{read\text{--}mod} = 0.2$ constant. The effect of varying $p_{read\text{--}mod}$, which varies the amount of write-back and broadcast invalidation traffic in the system, is explored in section 4.5. We look at the relative utilizations of the row and column buses in section 4.6, and study the viability of various hardware techniques for reducing the latency in remote memory requests in section 4.7.

4.1. Parameter Values Used in the Experiments

In all of the experiments reported in this section, we assume equal row bus speeds and column bus speeds. Furthermore, we let the bus cycle time define the unit of time in the model. Thus, the unit of time is 50 nanoseconds for 20 MHz buses, and 25 nanoseconds for 40 MHz buses. Bus access times, the mean time between bus requests, and the memory latency parameters are all specified in units of bus cycles. This allows us to present the results for a wide range of bus technologies concisely. Note that the cache block sizes quoted in our experimental results are measured in number of bus cycles. The actual size of the cache block is the quoted size times the width of the data bus. We assume $t_{A,r} = t_{A,c}$ = 2 cycles, and $t_{I,r} = t_{I,c}$ = 1 cycle. Thus, the bus access times for data transfers, $t_{D,r}$

and $t_{D,c}$, are the quoted cache block size plus 2 cycles.

For simplicity, we set the memory latency parameters, d_{cache} and d_{mem}, equal to 15 bus cycles in all experiments. For a 20 MHz bus, this specifies cache and main memory latencies of 750 nanoseconds. This number is a reasonably conservative, but possibly realistic value. Choosing this value allows us to consider bus speeds ranging from 20 MHz to 40 MHz without introducing much error due to memory latency assumptions.

The effect of cache miss rate is reflected in the parameter for mean time between bus requests, t_p. In order to compute t_p from miss rate, we need an estimate of the rate at which the class of memory references over which the miss rate is defined are generated by the processors. We denote the rate at which such references are generated per microsecond, by the MMRPS (millions of memory references per second) rating of the processors. Note that the references counted in the rating may include instructions and data, or only data, or to some other subset of the references generated by the processors. The important point is that there be some way of estimating (or guesstimating) the fraction of such references that miss in the 16 - 64 megabyte snooping cache, which we believe is roughly proportional to the fraction of the memory references that are to shared data.

Realistic values of the MMRPS and cache miss rate parameters in the proposed system are unknown. However, it may be useful to consider the values of these two higher level parameters that are implied by various values of t_p. Table 4.1 presents the miss rates corresponding to the values of t_p used in our experiments, for 20 and 40 MHz bus speeds, assuming MMRPS equal to 10. Miss rates for multiplying the bus speed by k, or the MMRPS value by $\frac{1}{k}$, are k times the values given.

Table 4.1: Miss Rate Conversions for t_p
(MMRPS = 10)

t_p (bus cycles)	100	200	400	1000	2000	4000
Miss Rate (20 MHz.)	2.0%	1.0%	0.5%	0.2%	0.1%	0.05%
Miss Rate (40 MHz.)	4.0%	2.0%	1.0%	0.4%	0.2%	0.1%

4.2. Upper Bounds on System Processing Power

We can derive an upper bound on system performance by solving equations (1) and (2) with all values of $W_{i,j}$ set to zero. This represents processing power that is limited only by the latency to retrieve remote data blocks.

Figure 4.1 shows this upper bound on system efficiency for a 32×32 system with block sizes ranging from 4 to 64 words (i.e. 4 to 64 bus cycles). The upper bound for a block size of 64 shows that even without contention for the buses, miss rates below one per 300 - 400 bus cycles (15 - 20 microseconds for 20 MHz buses) are required system efficiencies above 70%. The next section shows the effect of block size considering contention. In section 4.7 we will consider techniques that can be used to reduce the remote memory access latency.

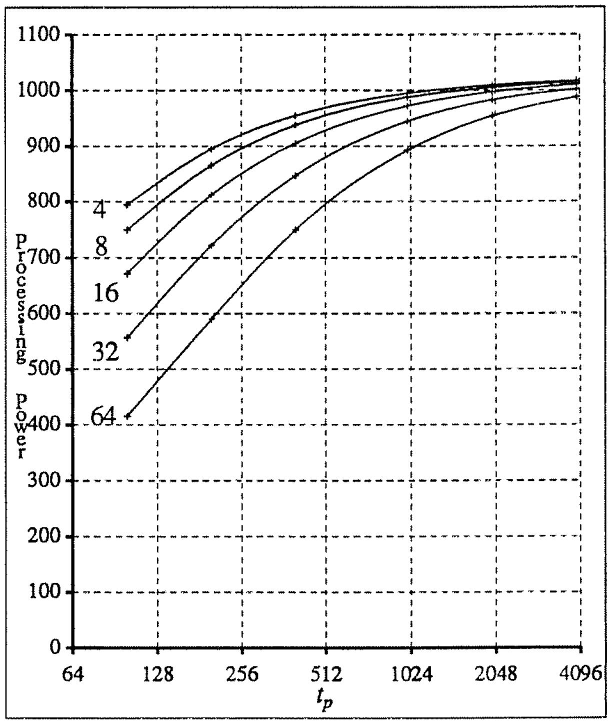

Figure 4.1: Upper Bounds on Processing Power

4.3. Effect of Block Size Choice

Figure 4.2(a) shows processing power estimates derived from our mean value performance model, versus miss rate (t_p), for a 32×32 system with block sizes of 4 through 64. Figures 4.2(b) and (c) show the results for system sizes of 20×20 and 10×10, respectively. In each figure, the five solid curves show the processing power estimates for different block sizes. The dashed and dotted curves represent bounds on the relationship between cache miss rate and cache block size which are discussed below.

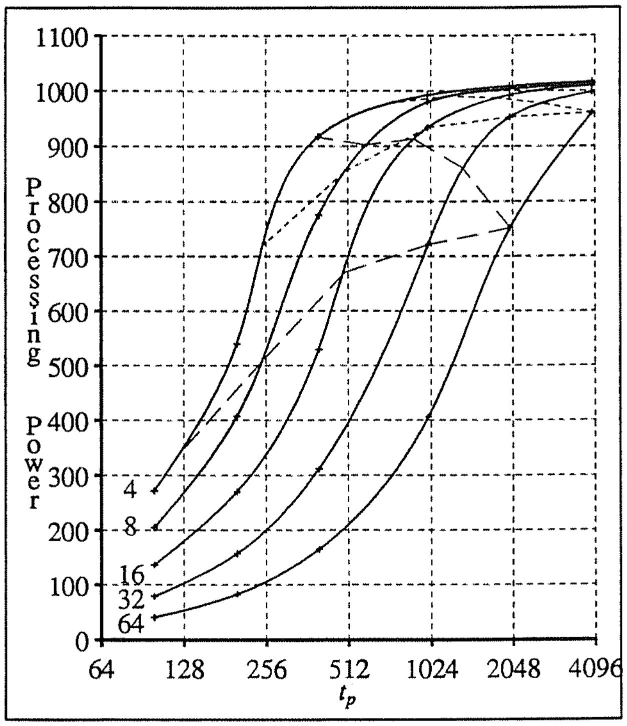

Figure 4.2(a): Effects of Cache Block Size for the 32 by 32 Multicube

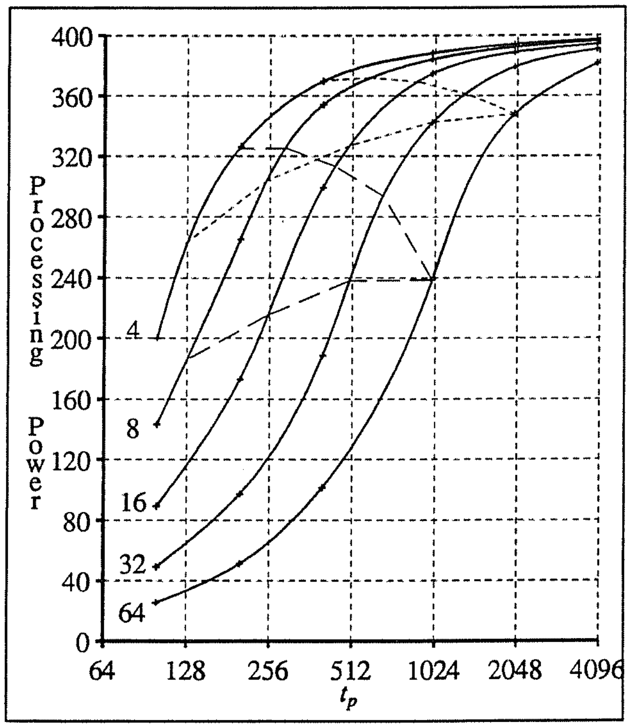

Figure 4.2(b): Effects of Cache Block Size for the 20 by 20 Multicube

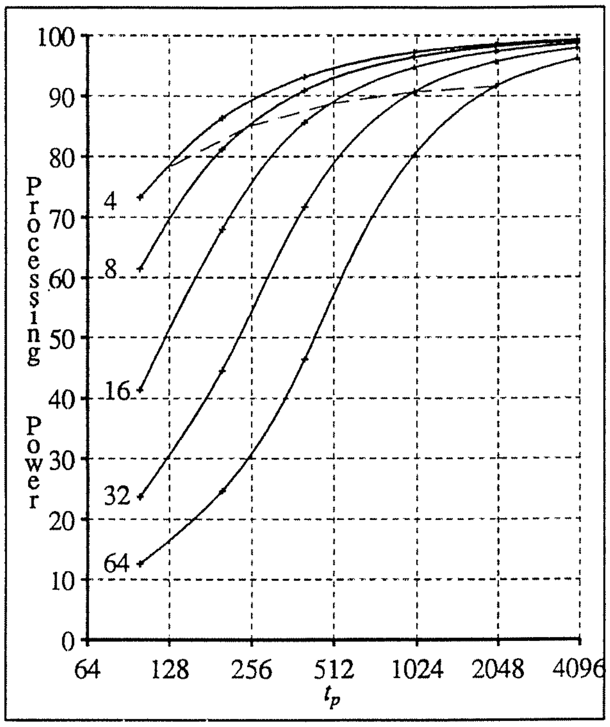

Figure 4.2(c): Effects of Cache Block Size for the 10 by 10 Multicube

First we note that for each cache block size in the 32×32 system, the performance drops dramatically for miss rates above some "critical" value. For block size of 4, this critical miss rate is approximately one miss per 200-300 bus cycles (i.e. per 10-15 microseconds in the case of 20 MHz buses). For block size of 64 bus widths, the critical miss rate region is on the order of one in every 1000-2000 bus cycles (i.e. 0.1% - 0.2% for 10 MMRPS and 20 MHz buses). If hit rates can be maintained above the critical value for the given block size over most of the operation of the system, then system performance can be expected to be quite reasonable (i.e. above 70% efficiency).

Very low average miss rates may be possible, given the large size of the snooping cache (16-64 megabytes). However, the steepness of the curves in the critical regions indicates possible stability problems if the miss rate is expected to vary much. Processing power curves for the 20×20 and 10×10 two-dimensional Multicube in figures 4.2(b) and (c) show that processing power degrades more gracefully as a function of miss rate for all block sizes as we move to smaller systems, and that higher miss rates can be tolerated for a given level of system efficiency. A design trade-off to be considered, although it involves other perhaps costly trade-offs, is to bound the size of the two dimensional system at some value lower than 32×32, and using the three-dimensional Multicube architecture for larger systems.[2] On the other hand, the processing power curve for blocksize of 64 still degrades rapidly for miss rates above one per 1000 bus cycles, even in the smaller two-dimensional system. Experimental evaluation of the memory reference characteristics of large-scale parallel programs is needed to determine the likelihood of the stability problem.

To determine the optimal block size for the system, it is important to consider the relationship between block size and cache miss rate. It is well known that decreasing the block size by a factor of k cannot decrease the miss rate, and cannot increase the miss rate by more than a factor of k. The dotted and dashed curves in figures 4.2(a)-(c) each start from different points on the processing power curve for a blocksize of 64 words, and indicate the worst case of doubling the miss rate, and perhaps a more "typical" case of multiplying the miss rate by 1.5, as the blocksize is cut in half. For example, in figure 4.2(a), starting at the processing power for $t_p = 2000$ on the curve for block size 64, the lower dashed curve shows the worst case, in which miss rate doubles as the block size is halved. In the best case, the miss rate remains unchanged as block size decreases. In perhaps a typical case, shown by the upper dashed curve, the miss rate increases by a factor of 1.5. The dotted curves show the same effects starting from the 95% efficiency range of the 64 block size curve.

Looking at the bounds and the "typical" relationship between cache block size and cache miss rate on the speedup curves, it appears almost certain that halving the block size from 64 to 32, and possible that halving the block size from 32 to 16, increases system performance. Below a blocksize of 16, it appears that performance decreases, or at best increases only very modestly. Thus, strictly from a performance standpoint, a cache block size of 16-32 words appears optimal. This blocksize translates to 128 - 256 bytes if the width of the data bus is 64 lines. Other costs for these "small" cache block sizes, such as the size of the large tag caches, must be weighed against the performance gains. However, the performance curves indicate that block sizes larger than 64 words (512 bytes) are not viable.

In section 4.7 we find that hardware latency reduction techniques do not alter the conclusions of this section.

4.4. Processing Power as a Function of System Size

Figures 4.3(a) and 4.3(b) show processing power of the two-dimensional Multicube as a function of system size, for one of the "optimal" block sizes, 16, and the limiting block size, 64, respectively. In each figure processing

[2] Note that for a given system size, the bus bandwidth per processor per hop in its path to remote memory increases as the number of dimensions in the Multicube increases.

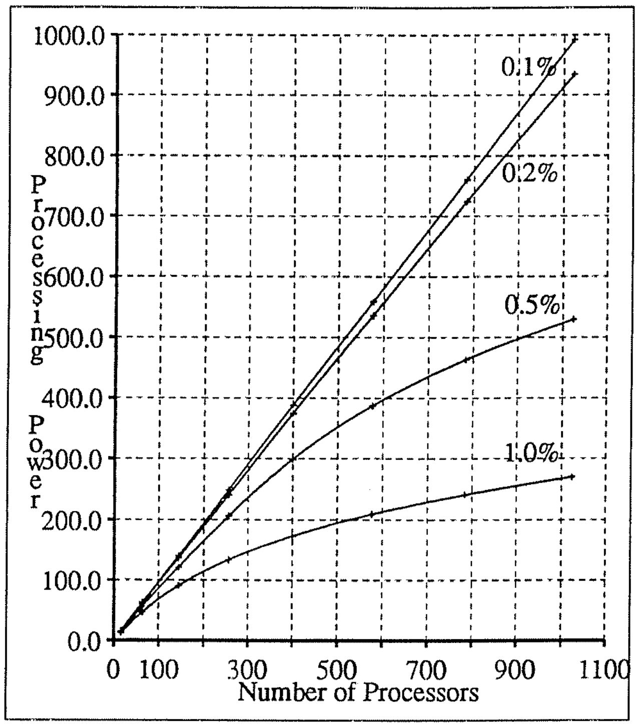

Figure 4.3(a): Processing Power versus System Size, Cache Block Size = 16

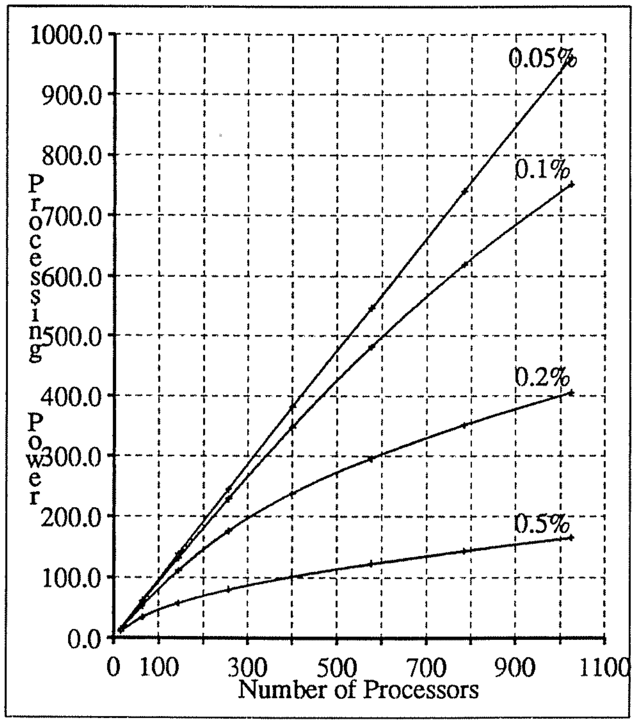

Figure 4.3(b): Processing Power versus System Size, Cache Block Size = 64

power curves are drawn for several cache miss rates, assuming 10 MMRPS. Figure 4.3(a) shows that for miss rates of 0.1 - 0.2 percent, nearly linear speedups are achieved for up to 1024 processors. The figure also indicates that for a miss rate of 1 percent, large increases in system size yield relatively modest increases in total processing power. Similar conclusions can be drawn for lower miss rate values from the curves for cache block size of 64 in figure 4.3(b).

4.5. The Effect of Broadcast Invalidations

This section investigates the effects of invalidation traffic on the system performance. The number of invalidations depends on the fraction of write requests to unmodified blocks. The relevant parameters in our model are $p_{read-mod}$ and p_s. In the above experiments, these parameters were held constant at 0.2 and 0.8, respectively. Realistic values of these parameters are highly workload-dependent, and unknown at this time. In this section we investigate the general effect of broadcast invalidations, by varying $p_{read-mod}$ while holding p_s=0.8 constant. Results for varying p_s while holding $p_{read-mod}$ constant lead to similar conclusions.

Figure 4.4 shows the effect of varying $p_{read-mod}$ for cache block sizes of 4 (solid curves), 16 (dashed curves), and 64 (dotted curves) in the 32×32 Multicube. The five curves for block size of 4, are for five values of $p_{readmod}$: 0.1, 0.2, 0.3, 0.4 and 0.5. The two dashed curves for block size 16 are for $p_{read-mod}$ values of 0.1 and 0.5. The curve for block size of 64 represents all values of $p_{read-mod}$ between 0.1 and 0.5. We see that the primary effect of increasing invalidation traffic is to move the critical miss rate region to the right (smaller miss rates), and that this effect decreases as block size increases. At the optimal block sizes of 16 - 32 words, the effect is fairly small, and at block size of 64 the effect is negligible, over the range of invalidations of 8 - 40% of the cache misses. This is reasonable, since each invalidation request requires only one bus cycle, whereas the data transfer requires 18 - 34 cycles.

The experiments reported in this section are very approximate, since $p_{read-mod}$ and p_s are not independent, (i.e. a larger value of $p_{read-mod}$ should imply a smaller value of p_s), and since increasing $p_{read-mod}$ decreases the frequency of write-back operations. This second issue could be addressed by introducing conditional read-mod probabilities for modified and unmodified blocks. Since values for these parameters are unknown at this time, we chose the more approximate experiment.

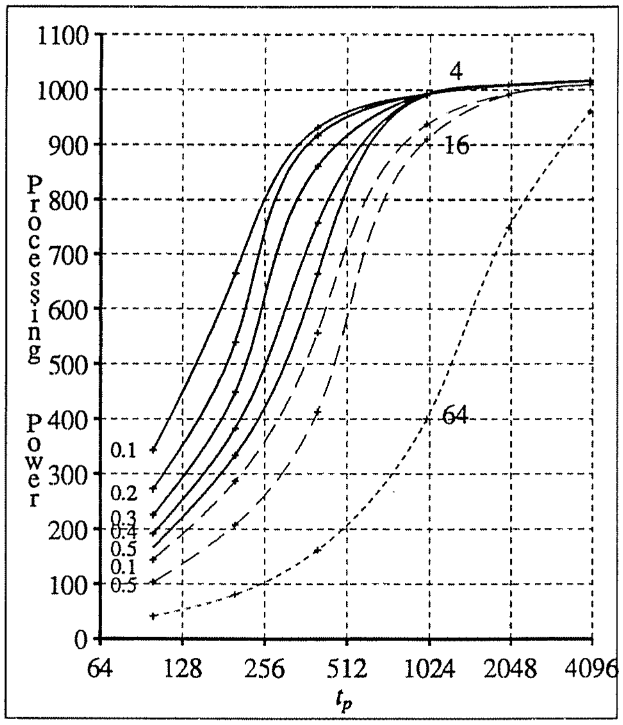

Figure 4.4: Effects of $p_{read-mod}$ for Cache Block Size of 4, 16, and 64

4.6. Row and Column Bus Utilizations

An important property of the Multicube system is the relative utilizations of the row and column buses. In particular, system performance is optimized when the row and column buses are nearly equally utilized (including consideration of the relative bus speeds). The cache coherence protocol distributes the demand for bus bandwidth nearly equally between the row an column buses, with the exception of invalidate and write-back traffic. The amount of invalidate traffic on the row buses versus write-back traffic on the column buses determines which bus is more heavily utilized. Again, these parameters are highly workload-dependent. In this section we report the relative utilizations of the buses for the $p_{read-mod} = 0.2$ and $p_s = 0.8$. Note that it is straightforward to obtain these utilizations from our model.

For a 32×32 system, a *READ-MOD* request for an unmodified block causes an invalidation request of duration one cycle on each of 31 row buses. A *READ* request for a modified block causes a write-back to memory of duration (block size + 1) bus cycles on one column bus. For $p_{read-mod} = 0.2$ and $p_s = 0.8$, these events occur equally often.

That is, each occurs in 16% of the cache misses. Thus we expect roughly equal column and row bus utilizations for cache block size of 32, higher utilization of the row buses for smaller block sizes, and higher utilization of the column buses for larger block sizes.

Figures 4.5(a) and (b) show the row and column bus utilization for a 32×32 system with block sizes of 4 and 64, respectively. Figure 4.5(a) shows that for a block size of 4, the row bus is much more utilized than the column, but both bus utilizations decline dramatically for cache miss rates below one per 400 cycles. The results for block size of 8 and 16 (not shown) are similar, but show increasingly more equal row and column utilizations. At block size 32 (also not shown), the buses are equally utilized. Figure 4.5(b) shows that for block size of 64 the column is now more heavily utilized than the row, due to the write-back traffic. We note that the row buses are still 90% utilized for miss rates higher than one per 1000 bus cycles. Thus, the inequality in bus utilizations is not responsible for the generally low performance of this block size.

4.7. Latency Reduction Techniques

Three techniques to reduce the delay for data transfers from remote memory (i.e. the data transfer latency) are considered in this section. Each technique has some cost in terms of the complexity of the system hardware. The first technique, which is also the least expensive to implement, is to supply the requested word of data to the processor as soon as it is received on the bus, rather than to wait until the whole block has been received. On the average, the processor only waits for one half of the last data transfer, which reduces the data transfer latency by about 25%. In the degenerate cases where the requested data is supplied by a processor on the same row or column, this technique would reduce the transfer latency by 50%. We call the technique the "supply immediately" technique.

The second technique is to identify the requested word in the *READ* or *READ-MOD* operation, and to transmit this word first in the data transfer operation. The cache could then supply the word as soon as the first word of the block is received, eliminating the latency of the last data transfer. This reduces the total data transfer latency by about 50% in most cases. We call the technique the "requested word first" technique.

The third technique, which we call the "pipeline" technique, is the most sophisticated technique and the most costly to implement. In the simpler schemes, a cache controller that forwards data from a column bus to a row bus, or vice versa, buffers a block it receives on one bus and then sends the data on the other bus. In the pipeline technique, the forwarding controller is able to pipeline the transmission on the outgoing bus with the reception of data

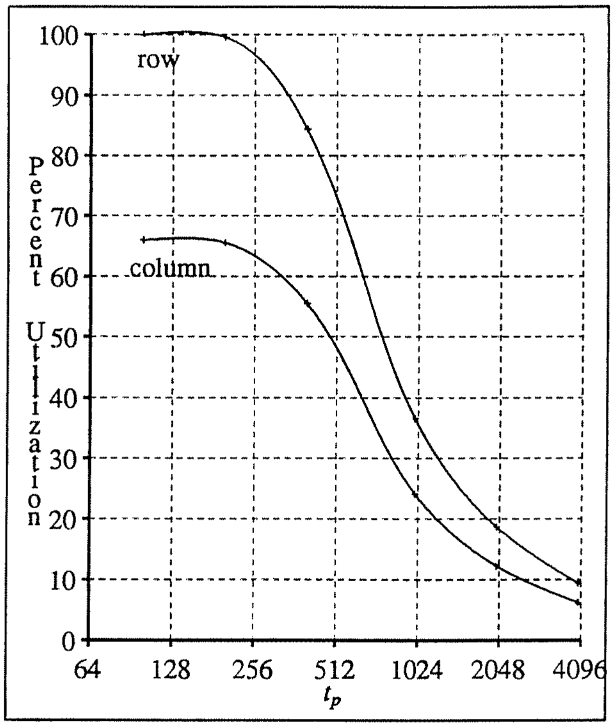

Figure 4.5(a): Bus Utilizations for Cache Block Size = 4

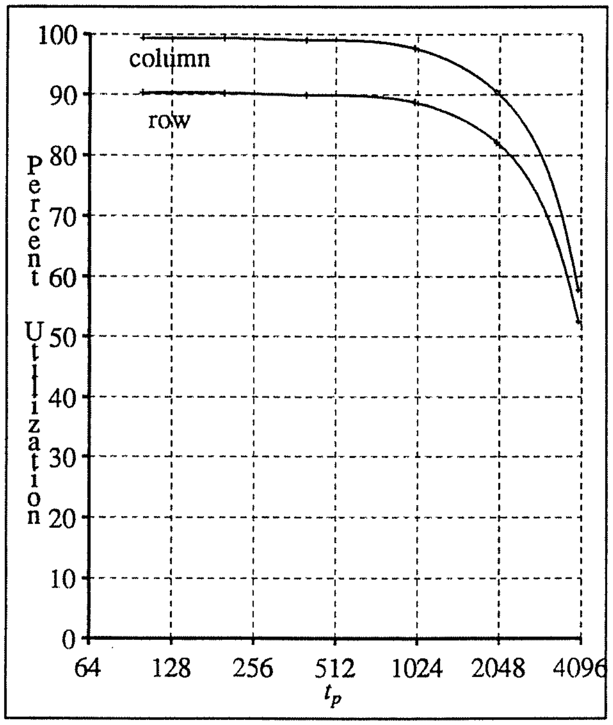

Figure 4.5(b): Bus Utilizations for Cache Block Size = 64

on the incoming bus. The pipeline technique also includes the requested-word-first technique, and eliminates all but two cycles of latency on each bus, regardless of block size.

It is important to note is that although these techniques reduce latency, they do not decrease the demand for the buses. The buses are still in use for the entire block transfer, and supplying the processor earlier will result in receiving the processor's next request sooner. As a result the amount of time spent waiting for a bus may increase, reducing the advantages of the reduction in latency. The mean value model is useful for estimating the actual performance gains with queueing times included, which is the subject of this section.

Figure 4.6(a) shows the processing power estimates for the original system and the three latency reduction techniques for a 32×32 system with cache block size of 64. Figure 4.6(b) shows the same curves for a 10×10 Multicube. Note the different range of values for t_p in each graph. Estimated relative performance gains for the latency reduction techniques were smaller for cache block sizes less than 64, and for the values of t_p not shown in the graphs.

Figure 4.7 shows the maximum relative increase in processing power for the pipeline technique as compared with the original system for size 10×10, 20×20 and 32×32. The maximum estimated increase in processing power for a 10×10 system is 8.5%, which occurs at a miss rate of one per 1000 cycles. For the 20×20 and 32×32 systems, maximum increases in processing power are only 5% and 3%. Furthermore, the maximum performance gains in each case occur at miss rates where performance is already acceptable. We find that the latency reduction techniques only improve performance at operating points where the buses are not heavily utilized. At miss rates where the buses are heavily utilized the waiting times outweigh any benefits obtained from the latency reduction. We thus conclude that the latency reduction techniques are not worth any substantial implementation cost.

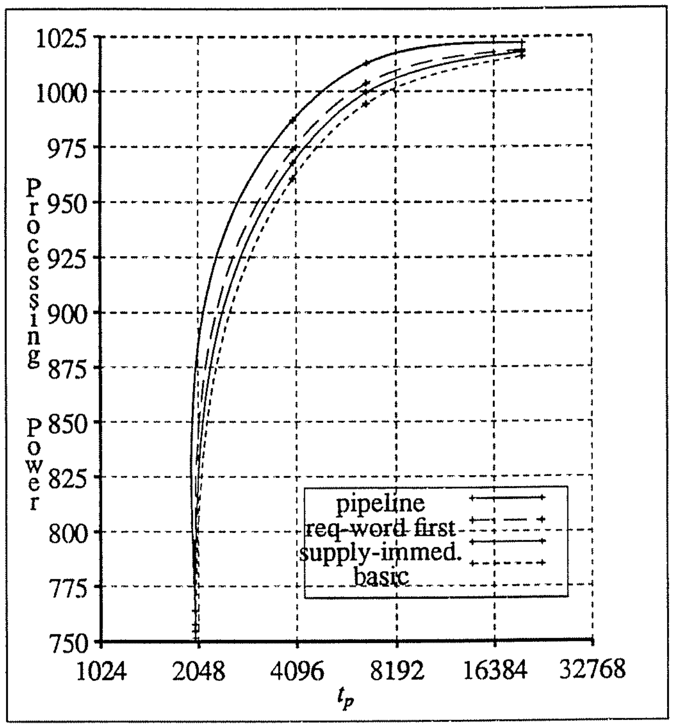

Figure 4.6(a): Latency Reducing Techniques for the 32×32 Multicube

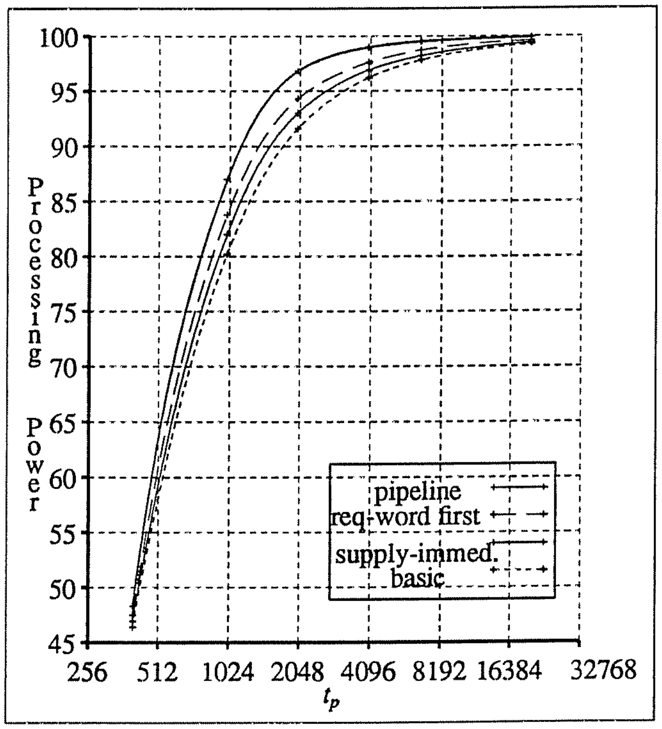

Figure 4.6(b): Latency Reducing Techniques for the 10×10 Multicube

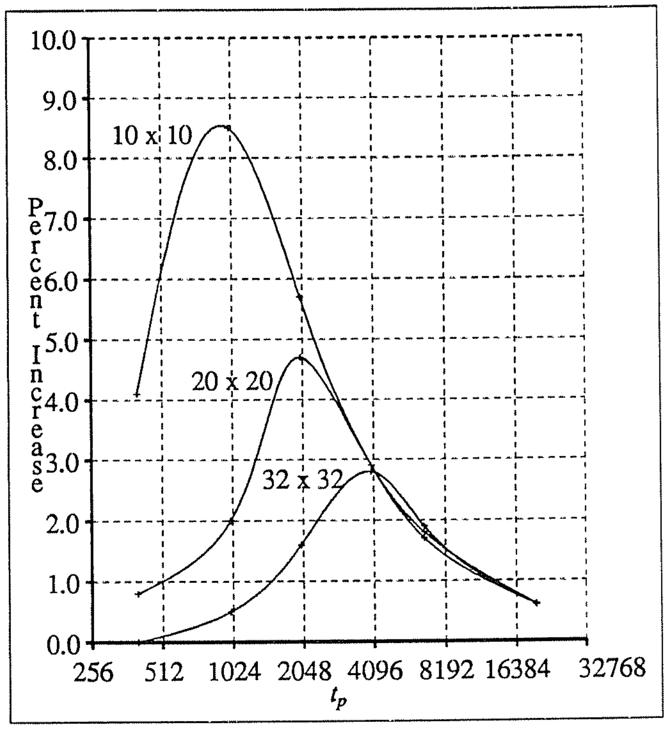

Figure 4.7: Percent Increase in Processing Power for Latency Reduction

5. Conclusions and Future Work

We have developed an approximate, customized, mean-value analysis of bus interference in the *Wisconsin Multicube*. The model includes FCFS scheduling at the bus queues with deterministic bus access times, asynchronous broadcast invalidations, and asynchronous memory write-back operations. The level of detail in the model is compatible with the level of detail known about the real system at this time. Furthermore, we believe the level of detail in the model is the appropriate level of detail for studying initial design trade-offs and for initial system capacity planning.

We have used our model to investigate several system feasibility and design questions. In particular, our results indicated that a 1024-processor Multicube can operate at 70 - 90% of its peak processing power, if cache miss rates are no larger than one per 1000 bus cycles (i.e. 50 microseconds for 20 MHz buses, or 25 microseconds for 40 MHz buses). In addition, the model estimates indicated that, strictly from a performance perspective, the cache block size should be on the order of 16 - 32, and no larger than 64, times the width of the data bus.

Our results for the effect of system size on processing power showed that nearly linear speedups are possible for up to 1024 processors, for cache miss rates no larger than one per 1000 (16-word cache blocks) or 4000 (64-word cache blocks) bus cycles. We also presented results that show the primary effect of invalidation traffic in the system is to increase the rate at which processing power declines for cache miss rates above some "critical value". Finally, we presented results that indicate various hardware techniques for reducing remote memory access latency are not worth any significant implementation cost.

For the 32×32 Multicube, our results show that for each cache block size, performance drops dramatically for miss rates above a "critical value". Thus, system stability may be an issue. Processing power curves for the 20×20 and 10×10 Multicube, show that processing power degrades more gracefully as a function of miss rate. Experimental evaluation of the memory reference characteristics of large-scale parallel programs is needed to determine the significance of the stability problem. A design choice to be considered, although it involves costly implementation trade-offs, is to bound the size of the two-dimentsional system at some value lower than 32×32, and to use the three-dimensional Multicube architecture for larger systems.

Our plans for the future include improvements and extensions to the model, and study of additional system design issues. In particular, the model developed in this paper ignores cache and main memory interference, and assumes uniform cache miss rates across all processors, and uniform memory access probabilities for all requests that miss in the cache. Cache interference may be more significant in the Multicube than in single-bus multiprocessors, since requests from more than one bus can interfere with each other and with the processor requests. Non-uniform miss rates and memory access probabilities would also be interesting features to investigate.

Additional system design issues that might be studied with extensions to the model in this paper include the performance advantages of various optimizations in the cache coherence protocol, the effects of I/O traffic on system performance, a more detailed analysis of the effects of the workload sharing parameters, and the performance of the three-dimensional Multicube. Other issues to be studied that will require more detailed analysis and/or simulations include the correctness of the cache protocol, and the transient behavior of the system at program start-up time. In all of these studies, data on the memory reference characteristics of real, large-scale, parallel programs are crucial.

Acknowledgments

We would like to thank Jim Goodman and Phil Woest for many fruitful discussions in the development of the model presented in this paper, and Mark Hill and Tom Reps for very helpful comments on the analysis.

References

[GoW87] Goodman, J. R., and P. J. Woest, "The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor", submitted for publication.

[HeTr82] Heidelberger, P., and K.S. Trivedi, "Queueing Network Models for Parallel Processing with Asynchronous Tasks," *IEEE Trans. on Computers*, Volumer C-32, No. 11, November 1982, p. 1099.

[LaVZ88] E. D. Lazowska, M. K. Vernon, and J. Zahorjan, "An Accurate and Efficient Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols", Computer Science Dept. Technical Report #746, Univ. of Wisconsin - Madison, January 1988.

[LZGS84] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik, *Quantitative System Performance*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[Reis79] M. Reiser, "A Queueing Network Analysis of Computer Communication Networks with Window Flow Control,"*IEEE Trans. Comm.*, Volume COM-27, 1979, pp. 1199-1209.

[Schw79] P. Schweitzer, "Approximate Analysis of Multiclass Closed Networks of Queues", *JACM*, Volume 29, No. 2, April 1981, pp. 358-371.

Appendix A

The conditional probabilities that a requested data block is on the same row, same column, or neither, as the requesting processor, given that the data block is unmodified or modified, are as follows:

$p_{co|s}$ = probability own column given that it is unmodified = $\frac{1}{N}$

$p_{f|s}$ = probability foreign column geven that it is unmodified = $\frac{N-1}{N}$

$p_{co|x}$ = probability own column given it is modified = $\frac{N-1}{N^2-1}$

$p_{ro|x}$ = probability own row given it is modified

$$= (\frac{N(N-1)}{N^2-1})(\frac{1}{N}) = \frac{N-1}{N^2-1}$$

$p_{f|x}$ = probability not same row or column given it is modified

$$= (\frac{N(N-1)}{N^2-1})(\frac{N-1}{N}) = \frac{(N-1)^2}{N^2-1}$$

The above probabilities are used in the following tables for $p_{i,j,k|S}$ and $p_{i,j,k|X}$:

$p_{k,i,j\|x}$	$j=o$		$j=f$	
	$i=r$	$i=c$	$i=r$	$i=c$
$k=A$	$1-p_{co\|X}$	$p_{co\|X}$	0	$p_{f\|X}$
$k=D$	$p_{ro\|X}$	$1-p_{ro\|X}$	$p_{f\|X}$	0
$k=AD$	0	0	0	0

$p_{k,i,j\|S}$	$j=o$		$j=f$	
	$i=r$	$i=c$	$i=r$	$i=c$
$k=A$	1	0	0	0
$k=D$	$p_{f\|S}$	0	0	0
$k=AD$	0	$p_{co\|S}$	0	$p_{f\|S}$

Appendix B

In this appendix we present the modifications to the mean value equations for the processor sharing scheduling discipline on the buses. Calculation of response time is the same as in equation (1) of section 3.2. Calculation of $R_{i,j,k}$ in equation (2) requires changing $W_{i,j}$ to $W_{i,j,k}$. This is because an arriving request must share the bus with all the other requests present for as long as it requires service. $W_{i,j,k}$ is calculated as the product of the arriving request's service time and the mean queue length found by the arrival. All the needed equations for the processor sharing scheduling discipline are presented below:

$$R = t_p + p_s\, d_{mem} + p_x\, d_{cache} + \sum_{\substack{i \in \{r,c\} \\ j \in \{o,f\} \\ k \in \{A,D,AD\}}} R_{i,j,k},$$

$$R_{i,j,k} = (p_x\, p_{k,i,j|x} + p_s\, p_{k,i,j|s})(W_{i,j,k} + t_{k,i}).$$

$$\overline{Q}_{i,k|o} = (N-1)\, \frac{R_{i,o,k}}{R} + N\, \frac{R_{i,f,k}}{R}.$$

$$\overline{Q}_{i,k|f} = N\, \frac{R_{i,o,k}}{R} + \left[N\,(N-1) - 1\right] \left[\frac{1}{N-1}\right] \frac{R_{i,f,k}}{R},$$

$$W_{i,j,k} = (t_{k,i}) \sum_{k' \in A,D,AD} (\overline{Q}_{i,k|j})$$

The row and bus utilizations are not needed to calculate the response times as they were in the FCFS case, but if the utilizations are desired to be known, they are calculated the same way as before:

$$D_{i,j,k} = (p_x\, p_{k,i,j|X} + p_s\, p_{k,i,j|S})\, t_{k,i}.$$

$$U_{i,j,k} = N \times \frac{D_{i,j,k}}{R}.$$

$$U'_{i,j} = U_{i,j,A} + U_{i,j,D} + U_{i,j,AD},$$

$$U''_{i,k} = U_{i,o,k} + U_{i,f,k},$$

$$U_i = U'_{i,o} + U'_{i,f}.$$

Appendix C

In this appendix we compare the approximate mean value equations with equivalent exact PFQN equations. The models evaluated in this section are of an earlier design of the system and are somewhat simpler. In particular, the models do not consider the degenerate cases where the desired data is held on the same row or column. As a result, every cache miss requires four bus accesses, two for the address and two for the data.

In Figure C.1 we show the queueing network equivalent to the earlier mean value model of bus traffic in the *Wisconsin Multicube*. The network is drawn for a 3×3 system, and contains the routing for one closed chain of customers. This chain, called R1, models the requests that originate at processors on row 1. The delay servers P1-P3 represent the local processing time of the processors on rows 1-3, the queues R1-R3 represent the three row buses and the queues C1-C3 represent the three column buses. The delay servers CD1-CD3 represent the latency of the caches on column 1-3, respectively. M1-M3 represent the memory latency for global memory modules on columns 1-3. Queues R1-R3 and C1-C3 have three incoming classes: the left incoming class is type A accesses, the middle incoming class is type AD accesses and the right incoming class is type D accesses. There are three line types in the figure. The solid line represents class A, the dotted line class AD and the dashed line class D. In the figure a chain that enters a queue as a certain class also exits the queue as that class. Class changes are shown by the line type changes at the inputs to the queues. For example a type A customer enters queue R1 and the exits as a solid line and goes to any of the three queues C1-C3 as either a type A access or type AD access.

We now trace the routing in the queueing network model for a request for an unmodified block and a request for a modified block. In either case, the request leaves P1 and enters R1 as a class A access, representing the row bus request for the block. The request then follows the the routing for requests to unmodified blocks with probability p_s, and follows the routing for requests to modified blocks with probability p_x.

(1) Request for an unmodified data block: The request next goes to either C1, C2 or C3 and enters as an AD access. Assuming it went to Ci, it then goes to Mi to acquire the desired data block. It then goes to R1 entering as a D transfer Finally the data returns to P1.

(2) Request for a modified data block: The request next goes to C1, C2 or C3 as an A access. Assuming C2 is chosen, it then goes to CD2 for the cache to supply the data block. Next it enters R2 or R3 as a D access to transfer the data block on the row bus of the supplying cache. If it chooses R2(R3) this implies that the cache supplying the data block was on R2(R3). From R2 or R3 it next enters C1 as a D access from where it returns to P1.

Each of the three closed chains in the QN model has a population of three customers. We compute the processing power estimate from the PFQN results by multiplying the mean queue length at P1 by 3.

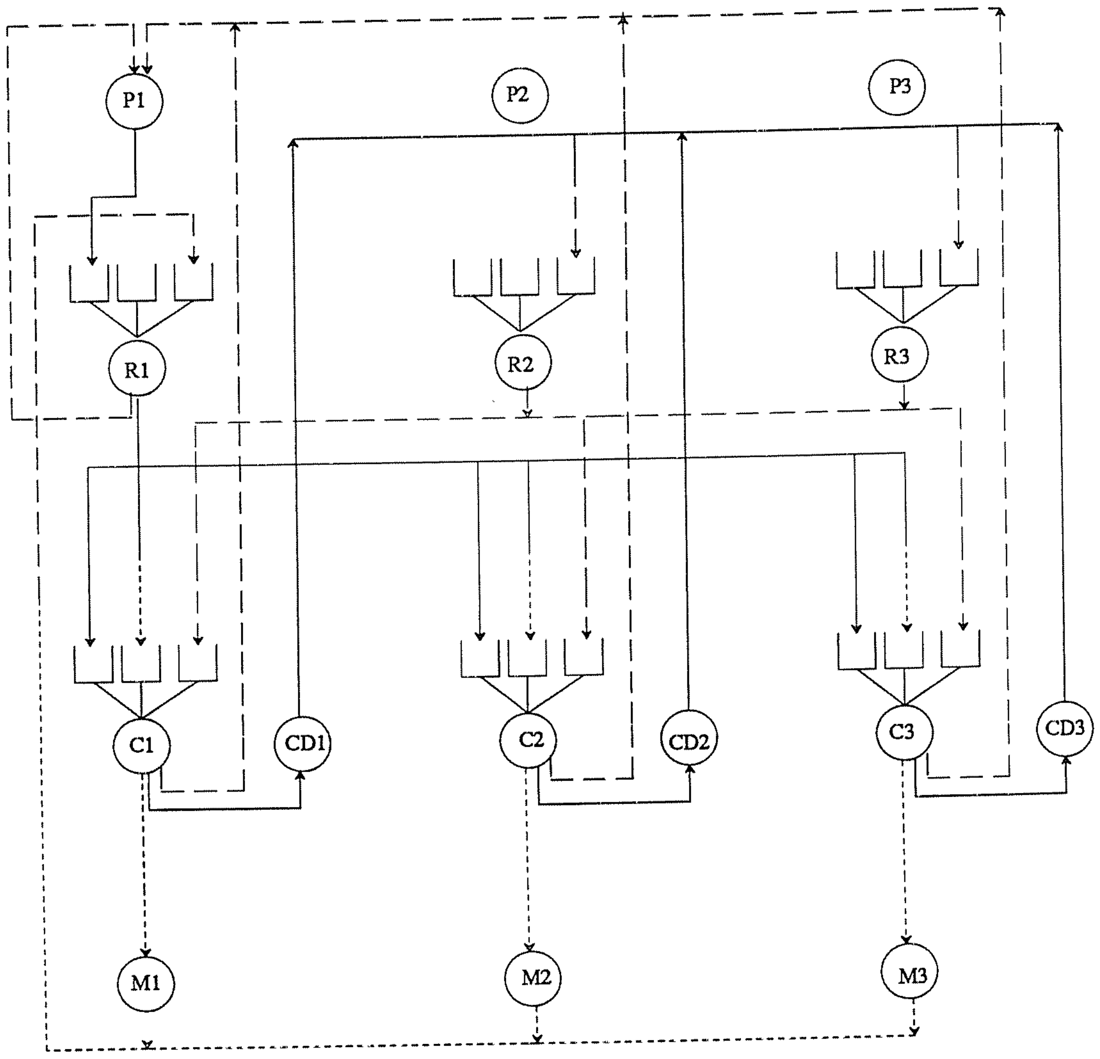

Figure C.1: PFQN Model of a 3 $\times$ 3 Wisconsin Multicube System

We compared the PFQN model with the equivalent approximate mean value equations over a range of inter-request times under fixed assumptions of cache and memory latency of 50 bus cycles, p_x = 0.2 and p_{rm} = 0.2. The range of inter-request times, 1 - 4000 bus cycles corresponds to hit rates of 0 - 99.85% for 10 MIPS processors and 10 MHz. buses, caused the bus utilization to vary from 0 - 65%. The error for the two methods increased as the utilization of the buses increased regardless of the block size chosen. With only a nine processor system a bus utilization of 65% was the highest that could be attained. The fact that the percent error increases with the bus utilization is not a problem since the *Wisconsin Multicube* is only feasible for utilizations less than 65% and maximum error at utilizations of 65% was 5%. Table C.1 shows the percent error versus the bus utilization for this experiment. Figure C.2 also shows the percent error versus the bus utilization. As noted before, the maximum observed error was 5.0%.

Table C.1: Percent Error in the Approximate Mean Value Model

Block Size:	4	8	16	64
Min. Error:	0.00	0.00	0.00	0.01
Max. Error:	1.24	1.99	3.13	5.00
Avg. Error:	0.22	0.44	0.69	1.62

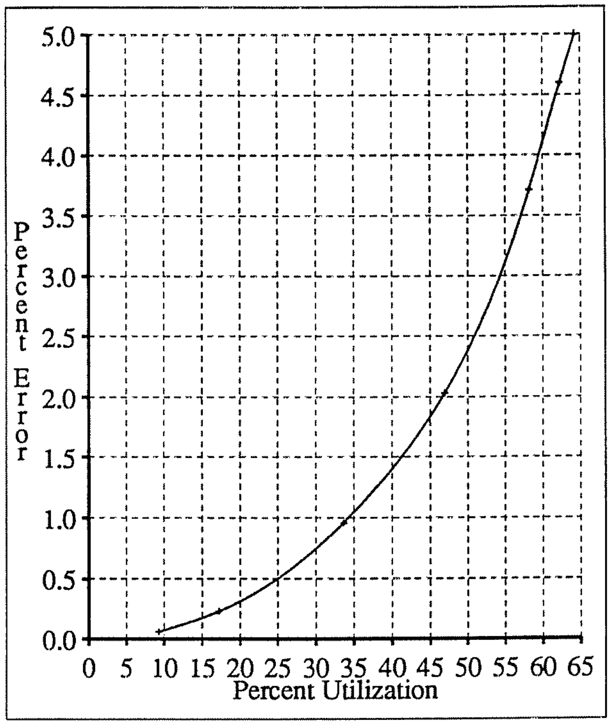

Figure C.2: Percent Error in the Approximate Mean Value Model

Appendix D

In this appendix we show the effects of the non-product-form assumptions in the mean-value model of Multicube. These assumptions are: FCFS service at the bus queues for more than one class of customers, deterministic bus access times, and asynchronous invalidate requests. Figures D.1 and D.2 shows the results of mean value models which contain only a subset of these assumptions, for a 32×32 system. In both figures, $p_x = 0.2$, $p_{read-mod} = 0.2$, and memory write-backs are ignored. Figure D.1 assumes a cache block size of 4, and Figure D.2 assumes a block size of 64 bus widths. The top curve is for FCFS scheduling with deterministic access times. The second curve from the top is for the product-form model (i.e. PS scheduling, no invalidates). The third curve is for FCFS scheduling with exponential bus access times. The bottom curve is for FCFS scheduling, deterministic bus access times, and invalidate traffic.

Comparing the FCFS exponential results to the product-form model results we see that the FCFS policy results in at most 1.3% (4.6%) lower processing power, for a block size of 4 (64). For processing power above 900 the difference in the two models is less than 0.5% for block size of 4 and 1.4% for a blocksize of 64.

The deterministic access times yield higher processing power, although once again the difference is not major. When compared with the product-form model, the FCFS model with deterministic access times differs by at most 5.3% (2.7%) for block size equal to 4 (64).

The model which includes invalidate traffic shows the lowest performance. As the overall cache miss rate increases, system performance degrades by as much as 40.3% for a block size of 4, relative to the FCFS scheduling with deterministic access times and no invalidations. For a block size of 64 the maximum observed decrease was 5.7%. Note, however, that at processing powers above 900, the degradation due to invalidations is less than 2.5% for block size of 4, and only 0.1% for a block size of 64.

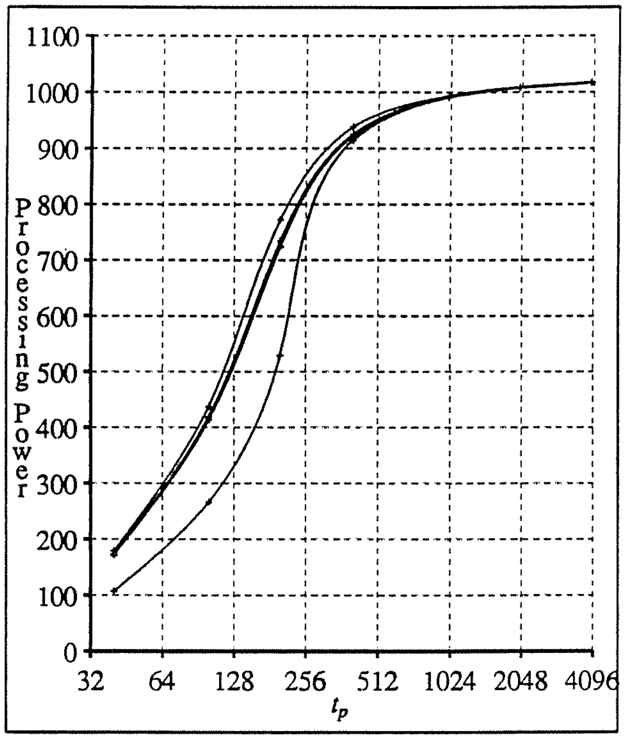

Figure D.1: Processing Power for Non-Product Form Extensions, BS = 4

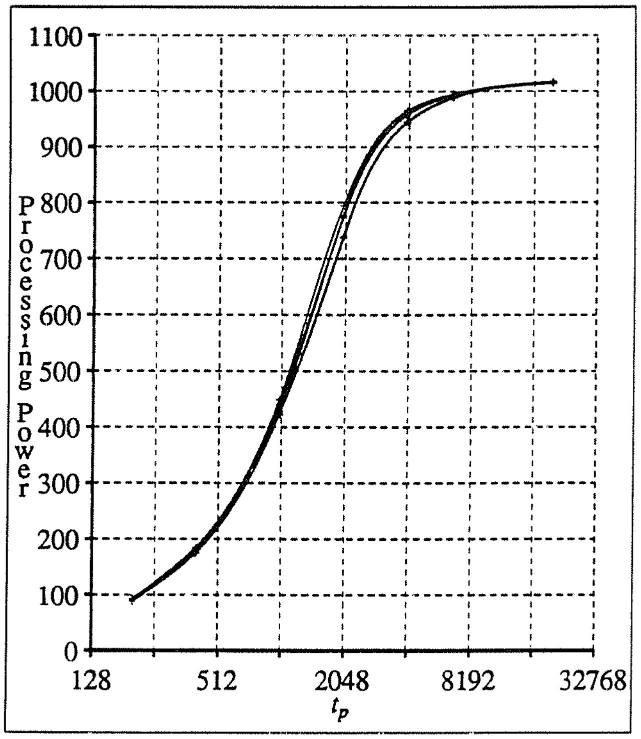

Figure D.2: Processing Power for Non-Product Form Extensions, BS = 64