An Easy-to-Use Approach for Practical Bus-Based System Design

Chung-Ho Chen, *Member*, *IEEE Computer Society*, and Feng-Fu Lin

Abstract—We present an easy-to-use model that addresses the practical issues in designing bus-based shared-memory multiprocessor systems. The model relates the shared-bus width, bus cycle time, cache memory, the features of a program execution, and the number of processors on a shared bus to a metric called request utilization. The request utilization is treated as the scaling factor for the effective average waiting processors in computing the queuing delay cycles. Simulation study shows that the model performs very well in estimating the shared bus response time. Using the model, a system designer can quickly decide the number of the processors that a shared bus is able to support effectively, the size of the cache memory a system should use, and the bus cycle time that the main memory system should provide. With the model, we show that the design favors caching the requests for a contention-based medium instead of speeding up the transfers although the same performance can be respectively achieved by the two techniques in a contention-free situation.

Index Terms—Bus-based shared-memory multiprocessor, memory system design, queuing delay model, system design.

1 Introduction

A bus-based shared-memory multiprocessor system (MP) provides a low-cost solution for high performance MIMD systems. The performance of an MP machine depends on the hardware design as well as the features of a program execution. There are different analysis techniques and tools that are used to evaluate the design and performance of an MP system under different system models [1], [2], [3], [4], [5], [6], [7], [8], [9]. Simulation techniques include multiple levels of detail in modeling of an MP system [1], trace-driven simulation [2], [3], [4], queuing model [5], mean value analysis [6], [7], and simulation tools [8], [9].

While previous works studied various aspects of the issues involved in achieving high performance, they did not address the practical problem of how to put various hardware parameters together for system design. That is, given some representative parallel programs such as the SPLASH-2 suite [10], how should a system designer decide quickly the number of processors that the shared bus is able to support effectively? How large is the cache memory that the system should use? Or how wide and long are the bus and bus cycle time to which the main memory system should be connected? In a bus-based MP system, the bus bandwidth ultimately limits the performance of the machine. In this paper, we present an easy-to-use model, from the view point of system design, to evaluate the design and performance of shared-bus multiprocessor systems. The model relates the shared-bus width, bus cycle time, cache memory, the features of a program execution, and the number of processors to a metric called request utilization. The request utilization is treated as a scaling factor for the effective average waiting processors to compute the queuing delay in processor cycles. We verify this model by detailed simulations using the SPLASH-2 parallel programs. For the average square error, the error value is less than 5 percent in the discrepancy of queuing delays between the results of the simulation and the results of the proposed model. For most of the cases, the model performs very well.

In a uniprocessor setting, it is possible to obtain the same performance by trade-off among the main memory speed, cache design, or data bus width. For the past, cache design has received the greatest attention, but rarely were there evaluations among these three features. In a bus-based multiprocessor configuration, determining which combination of the hardware techniques—larger caches, wider bus, or faster memory—is more favorable from the view point of performance is important. By using the model developed in this paper, we show that, in theory, due to the generation of less bus contention, a bus-based multiprocessor system favors the usage of a larger cache rather than a wider shared data bus or a faster shared-bus, although the same performance can be obtained by trading these three factors in a uniprocessor system. More generally, this study shows that the design favors the caching of the requests for a contention-based medium even though the same performance can be achieved by either caching the requests or speeding up the transfers in a contention-free situation.

The rest of this paper is organized as follows: In Section 2, we give the notation and develop the design model. We present the verification of the model in Section 3. Section 4 examines the trade-offs among cache memories, bus width, and bus speed. Our conclusions are presented in Section 5.

• *C.-H. Chen is with the Department of Electronical Engineering, National Cheng Kung University1, University Road, Tainun, Taiwan. E-mail: chchen@mail.ncku.edu.tw.*
• *F.-F. Lin is with Great China Technology Inc., 3F, No. 29, Lane 114, Sec 3, Chung Shan Rd., Chung-Ho City, Taipei Hsien, Taiwan. E-mail: peterlin@gctitw.com.tw.*

Manuscript received 14 Sept. 1998; revised 31 Jan. 1999.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 107396.

TABLE 1
Hardware and Program Execution Parameters

β_m	normalized average bus cycle time for a complete bus transaction.
C_e	the number of cycles executed for a processor for an application excluding queuing delay.
R	the number of data bytes read in full bus width by a processor upon read or write miss (if a write-allocate cache is used) for C_e cycles executed.
α	cache line flush ratio for C_e cycles executed $(0 \le \alpha \le 1)$. The number of bytes of dirty lines which are copied-back (flushed) for C_e cycles executed is represented by αR.
I_n	the number of invalidations for a processor during the execution of C_e cycles.
D	data bus width in bytes which can be any number $\in \{4, 8, 16, 32\}$.
L	cache line size in bytes.

2 A New Approach

From the view point of system design, we need to consider the architectural parameters that characterize a parallel program and the corresponding hardware. These parameters are described in Table 1 and most of them are self-explanatory. The bus cycle time β_m refers to the cycle time for a data transfer between the processor and the main memory. β_m is normalized with the CPU clock cycle time. In this paper, the bus cycle time β_m includes the bus arbitration time, the address decoding time, and the memory access time, but not the queuing delay cycles, if any, in an MP system. In a circuit-switched bus system, the bus cycle time is also the memory cycle time because the shared bus is held by the master until the entire transaction is completed. For instance, the block read/write of the IEEE Futurebus [7], the shared bus in the Proteus system [9], and the Tobus [11] all employ a circuit-switched protocol.

In a split transaction bus, the bus is released between a request and its corresponding response. Hence, the latency of a slave device is not included in the bus cycle time. The bus cycle time β_m for a split transaction bus includes the request cycle time and the data transfer cycle time. The Sequent Symmetry, Balance, and the SGI Challenge multiprocessors use the split transaction protocol [12], [13], [15]. Parameter R, α, and I_n relate to the shared-bus traffic contributed by each processor. The width D refers to the data bus width of a shared bus. In particular, the following memory characteristics are used for this study.

1. For a read or a write miss, a whole cache line is brought into the cache from the memory. All of the memory references are assumed to be cacheable.
2. The cache memory uses a write-back policy for handling write hits and write allocate for handling write misses. When considering an MP system, the popular MESI protocol is used for cache coherence.

For the analysis, we do not consider write-through caches and noncacheable traffic. However, it should be easy to extend our analysis for these two cases [16]. The conventional cache hit ratio or miss ratio is not adequate to measure the effect of shared memory references. It may give misleading information in evaluating the shared-bus capability. For instance, a write-back cache takes extra memory cycles when a dirty block is replaced [14]. A write-allocate cache produces read cycles when a write miss occurs [15]. Thus, the cache hit (miss) ratio cannot easily be converted to the number of memory references. To take account of all bus traffic, we define

$$M_C = \frac{\frac{(1+\alpha)R}{D} + I_n}{C_e}, \ 0 \le M_C \le 1 \tag{1}$$

to be the average number of memory requests per cycle for a processor. The parameter M_C depends on the cache memory used, the bus width, and the application. Note that the parameter C_e is the number of cycles for the execution of a particular application for a processor excluding the request queuing cycles. In the following, we describe how to use M_C to obtain the maximum number of processors that a shared bus is able to support effectively.

2.1 Bus Request Utilization

The bus utilization U for a single processor is represented as follows.

$$U = \frac{(\frac{(1+\alpha)R}{D} + I_n)\beta_m}{C_e}, \tag{2}$$

that is,

$$U = M_C \beta_m \tag{3}$$

replacing with M_C defined previously. The numerator in (2) is the cycles spent in bus operations, while the denominator is the total execution cycles. In an MP system where k processors are present, the shared-bus *request utilization* is described by

$$U_D = \beta_m \left(\sum_{i=1}^{k} M_{Ci} \right), \ 0 \le U_D \le \beta_m \times k. \tag{4}$$

The notation M_{Ci} associates with processor i. For the shared bus to operate effectively, the limit of its *request utilization* U_D must be within 100 percent. That is,

$$\beta_m \left(\sum_{i=1}^{k} M_{Ci} \right) \le 1. \tag{5}$$

The limit on $U_D = 1$ provides an opportunity for each request to be completed without waiting if the requests are properly ordered without overlapping between each other.

If this limit is violated, the bus may be overloaded, where excessive queuing delay occurs. A system may operate at a request utilization greater than one. The request utilization is different from the bus utilization where its value is always less than one. In theory, (5) is similar to the utilization law, which states that the utilization of a resource is equal to the product of the throughput of that resource and the average service requirement of that resource [17]. In (5), the bus cycle time is the service requirement and the sum of the average memory requests per cycle of processors is the throughput.

The shared-bus request utilization, or request utilization for short, is determined by the bus cycle time β_m, bus width D, the number of processors k, the features of a program execution, and the employed cache memories. Consider a case where each processor with on-chip first level caches produces an M_C of 5 percent in a bus system designed with $\beta_m = 5$. Then, the maximum number of processors that the shared bus is able to support effectively is four. For an MP design, a system designer can estimate the M_C with simulations for the applications of interests by varying cache size, organization, bus width, and the bus cycle time. Then, the number of processors can be chosen to determine if the request utilization is within the limit.

2.2 The Contention Model

In an MP system, the shared-bus contention from multiple requests increases the CPU execution time. Traditionally, researchers have modeled the shared-bus as a closed queuing system and solved the problem with techniques such as the MVA (mean value analysis) method. However, there have been no positive reports on the accuracy of this method applied in parallel programs such as the SPLASH-2.[1] For a bus-based MP application, the CPU execution time of a processor is

$$E_{MP} = \left(C_e - \left(\frac{R(1+\alpha)}{D} + I_n\right)\beta_m\right) + \left(\frac{R(1+\alpha)}{D} + I_n\right)T_k, \tag{6}$$

where T_k represents the average shared bus request response time per bus access when k processors are present. Equation (6) is a direct modification from the denominator in (2). The first term in (6) is the total cycles minus the cycles for bus operation without queuing delay. The second term is the total bus cycles including queuing delay. The response time T_k is the sum of the bus cycle time (β_m) and the queuing delay cycles. The response time depends on the number of active processors, the characteristics of the processor requests, the bus cycle time, and the cache memories used. In a dynamic execution processor, a portion of the total bus cycles designated by $(\frac{R(1+\alpha)}{D} + I_n)\beta_m$ may be hidden by out-of-order execution. If this is the case, the bus cycles in (6) will only represent those cycles that do contribute to the total execution cycles. Using the request utilization (U_D) directly, we propose a new model for computing the queuing delay cycles.

First, we consider $U_D = 1$ in a system with k processors. In this case, the worst response time of the shared bus for a request is delayed by $k-1$ requests and delayed by β_m cycles from each of the $k-1$ requests. Therefore, the worst case response time is

$$\beta_m + (k-1)\beta_m = k\beta_m. \tag{7}$$

On the other hand, the best response time at $U_D = 1$ is β_m when no other bus request is waiting ahead of the current request. On average, at $U_D = 1$, the average response time is

$$\beta_m + \left(\frac{1}{2} + \left(\frac{k-1}{2} - 1\right)\right) \times \frac{k-1}{k} \times 1 \times \beta_m. \tag{8}$$

The request on average sees $(k-1)/2$ requests ahead (i.e., half of the requests) and delays $\beta_m/2$ cycles on average for the first waiting request and delays β_m cycles for the rest of the waiting requests. The total delay cycle is scaled by $\frac{k-1}{k}$ at $U_D = 1$. This is to exclude the current requester itself from the request utilization. For the general case of a given U_D, when a processor does a shared bus request, the average response time of that request is

$$T_k = \beta_m + \left(\frac{1}{2} + \left(\frac{k-1}{2} - 1\right)\right) \times \frac{k-1}{k} \times U_D \times \beta_m. \tag{9}$$

Essentially, the request utilization scales the average waiting cycles for each request. If $U_D > 1$, then the request utilization simply "amplifies" the average waiting cycles by increasing the effective number of the waiting processors (requests).

3 Contention Model Validation

We implement a shared memory simulator based on the Augmint simulation system to verify the queuing delay model. The Augmint is an event driven multiprocessor simulator for Intel x86 architecture [18]. Our memory simulator models the cache memory (direct-mapped with write allocate for write misses), the MESI coherence protocol, and the arbitration of the shared bus. The data bus width is 8-bytes in length. For the baseline memory latency (`x1` model), it takes 13, 21, 37 CPU clocks in loading a 32, 64, 128-byte cache line, respectively. The time to replace a modified line takes the same CPU clocks. A write-through cycle (invalidation) takes seven CPU cycles in the `x1` latency model. These memory cycles are converted to the average bus cycle time per bus transfer (β_m) according to the percentage of references of each type when used in the model. The simulator models the back-off and retry of a reference that hits on a modified line in the other cache as the real operation of the Pentium processor does. The number of processors used are 2, 4, 8, 16, and 32. We use the SPLASH-2 parallel programs as the benchmark. Table 2 provides the characterization of the SPLASH-2 programs used in the simulation for a 32-processor configuration.

Table 3 depicts some typical M_Cs for the SPLASH-2 programs for a 32-processor configuration. The value is the

1. In another effort, we also try the close queuing model and solve it with the MVA method but only discover that the MVA method significantly under estimates the queuing delay cycles.

TABLE 2
Featrues of Program Executions for a 32-Processor Configuration (512KB Cache, Line Size = 32 Bytes)

Code	Problem size	Total reads	Total writes	Read from shared state	Write to shared state
Barnes	256 particles	8039065	2829498	896086	7710
FFT	64K points	690669	241018	32954	1057
LU	128x128 matrix	11151811	2477434	1108039	3963
Ocean	18x18	6951500	2619740	1288967	38149
Radix	radix 1024 keys 16384	25301447	12068961	2780863	12528
Water-nsq	64 molecules	17779975	6436081	1840638	16218
Water-sp	64 molecules	5061311	1823805	638922	2621

TABLE 3
M_C for a 32-Processor Configuration with `x1` Latency in Loading a Cache Line

	Barnes	FFT	LU	Ocean	Radix	Water-nsq	Water-sp	Average
(8K,32,x1)	0.0104	0.0063	0.0019	0.0162	0.0203	0.0127	0.0072	0.0107
(8K,128,x1)	0.0243	0.0198	0.0073	0.0373	0.0340	0.0305	0.0188	0.0246
(512K,32,x1)	0.0039	0.0050	0.0004	0.0109	0.0080	0.0042	0.0016	0.0049
(512K,128,x1)	0.0110	0.0098	0.0004	0.0291	0.0122	0.0081	0.0040	0.0106

average for each processor. Because of high hit ratio, the M_C is generally very low for these applications. The number of invalidations (I_n) which is not shown in the table is extremely low compared to the bus transfers for read/write misses and replacement. In the simulator, the instruction references are assumed to be cache hits and, therefore, the M_C in Table 3 only takes account of the data cache miss traffic. For a chosen configuration, the model uses the same amount of bus traffic obtained from the simulation in determining the queuing cycles. Therefore, the queuing cycles computed from the model and those obtained from the simulation can be fairly compared on the same basis, i.e., the same amount of bus traffic.

As observed from Table 3, the M_Cs of a system using 8KB caches are higher than that using 512KB caches, though the 512KB caches result in more sharing (more I_n). The M_C of using an 8KB cache in each processor is about twice that of using a 512KB cache. In Table 3, the first column also shows the cache line size used.

3.1 Comparisons of Queuing Cycle Percentages

In this section, we present the percentage of the queuing cycles obtained from the simulation as well as computed with the model for each of the programs. The percentages of queuing cycles obtained from the simulation are indicated by the "+" sign while the results computed with the model are shown by the "o" symbol in the figures. The percentage of queuing cycles is based on the total program execution time.

The results for the Barnes, Radix, and Ocean program are given in Fig. 1, Fig. 2, and Fig. 3, respectively. In general, the model predicts very close results to those of the simulation. For these programs, we find that a larger discrepancy between the simulation results and the model values occurs when the shared bus is highly utilized. The model tends to estimate more queuing cycles for the cases when more than 40 percent of the total execution time comes from the queuing delay. However, with this amount of bus request delay, the request utilization is usually more than 100 percent. The values of the request utilization for each configuration are given in Fig. 5. As expected, the (8KB,128) cases have the highest request utilization among those in consideration.

Fig. 4 gives the results for the Water-sp program. The model over estimates the queuing delay by about 10 percent in the system using 8KB caches as the processor number is greater than eight. The reason for this is that most of the bus requests are made with a timing biased in the less contention situation. For the 512KB cache systems, the model in general under estimates the delays. However, in the 512KB cases, the percentage of the queuing delay cycles is already quite small. The request utilization is less than 0.5 for the 512KB cases, as shown in Fig. 5d.

For the Water-nsq program, the model performs well in most of the cases, as illustrated in Fig. 6. The results for the FFT program are given in Fig. 7. For the (8KB, 128) case, the model predicts accurately in most of the configurations. For the other configurations, as the processor number is greater than eight, the model under estimates the queuing cycles. This is because the FFT program runs a significant number of cycles that bias in the high contention side as compared with the model using average. The request utilization for the Water-nsq and the FFT program is shown in Fig. 9.

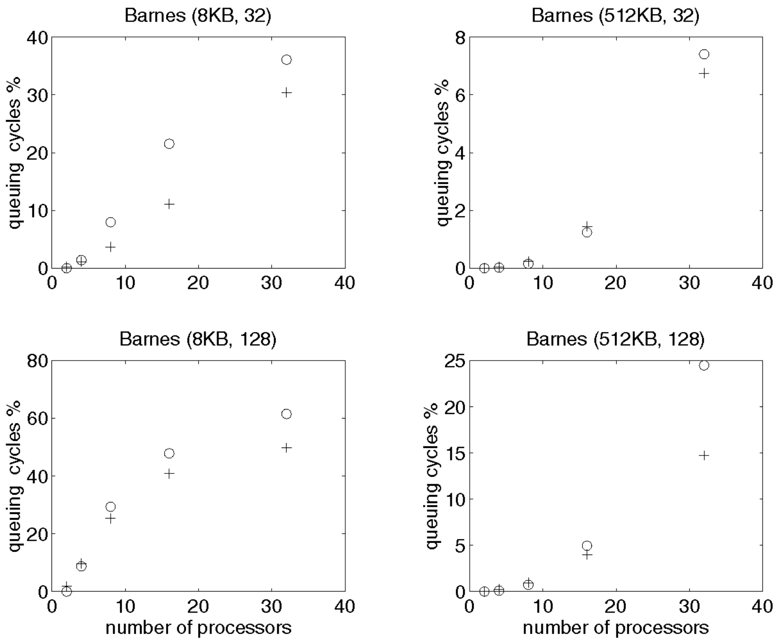

Fig. 1. Barnes: "+" indicates the result from simulation. "o" indicates the result computed by the model.

The LU program has a very high hit ratio. This makes it difficult to reach a point with a sufficient number of memory references for the model to work well in average. As shown in Table 3, the M_C of the LU program is the smallest among the programs we evaluated. The comparisons between the simulation and the model results are presented in Fig. 8. Observe that the percentage of the queuing cycles is quite small, especially for systems with 512KB caches. The corresponding request utilizations are also quite small, as illustrated in Fig. 9.

Several common observations are obtained from the above results. First, using a smaller cache, such as 8KB, results in a higher percentage of queuing cycles than using a larger cache, such as 512KB, although the latter tends to have more sharing traffic. This is because the M_C is higher when using a smaller cache. Also, we note that using a line size of 128 bytes in most of the configurations incurs more queuing cycles than using a line size of 32 bytes. This high delay cycle is due to the high request utilization.

Examining the queuing delays and the request untilizations, we can easily find that, when the request utilization is less than 100 percent, the incurred queuing cycles are generally less than 30 percent. A designer can use (4) directly to determine the target request utilization by varying the bus cycle time and the number of the processors given a known M_C. From a higher level point of view, the application requirements are specified implicitly by the average number of memory references per cycle (M_C). There is no need to know the exact value of parameter $R, C_e, \alpha,$ and I_n.

3.2 Model Validation

In this section, we present the error values between the simulation results and the results computed with the model for the above applications. For the seven SPLASH-2 programs, first we show the average queuing delays from the results of the simulation and the model. The result is given in Fig. 10. The largest discrepancy between the simulation and the model is about 9 percent in terms of queuing cycle percentage for the (8KB, 128, x1) instance in a 32-processor configuration. Most of the other cases have very close matches between the results of the simulation and the model.

To examine whether the model is robust enough against variations, we perform more simulations by varying the memory latency, cache size, and line size. The average queuing delays are presented in Fig. 11 and Fig. 12. The memory latency used in Fig. 11 is increased to three times those in the x1 model in Fig. 10. This memory latency when used is indicated with x3 in the figures. Fig. 11 shows that the model again performs very well in estimating the queuing cycles. In Fig. 12, the results are given with the use of a 32KB cache with different line sizes and memory latency. The model is very accurate in estimating the queuing delay cycles for these variations.

In addition to evaluating the errors in terms of the average values of the applications, we also compute the mean square of error values between the results of the simulation and the model. The mean square of error value is computed as follows:

$$Mean\ square\ error = \frac{\sqrt{\sum_{i=1}^{N}(M_i - S_i)^2}}{N}, \qquad (10)$$

where M_i and S_i are the result values of the model and the simulation, respectively. As given in (10), with the mean square error, each of the differences between the simulation values and the model values, no matter if it is plus or minus, contributes to the average error value. The mean square errors and the errors using the average values of the

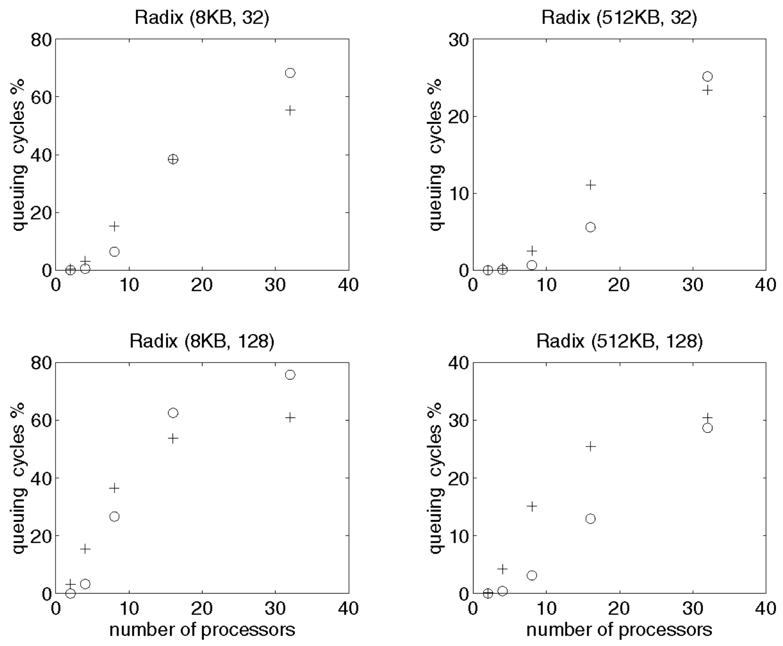

Fig. 2. Radix: "+" indicates the result from simulation. "o" indicates the result computed by the model.

applications are given in Tables 4 and 5 for all of the configurations evaluated in this work. As observed from Table 5, the largest error of average value is about 12.6 percent in queuing cycle percentage, while the corresponding mean square error is 5 percent. All other configurations show errors much less than this value. The error values have indicated that the model in general has produced satisfactory results in estimating the queuing delays.

4 Better Caching, Faster Memories, or Wider Bus?

In a uniprocessor setting, it is possible to trade-off among the main memory speed, cache design, or data bus width to obtain the same performance. In a multiprocessor configuration, the trade-off from the view point of performance favors the design that produces less bus contention. Which combination of these hardware techniques leads to the desirable result? In this section, we answer this question by developing a theoretical model. First, we define some

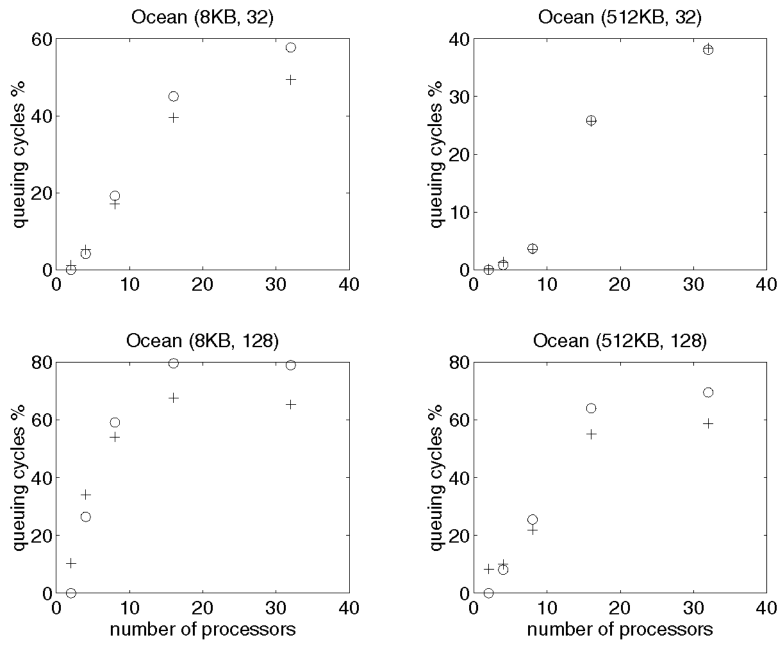

Fig. 3. Ocean: "+" indicates the result from simulation. "o" indicates the result computed by the model.

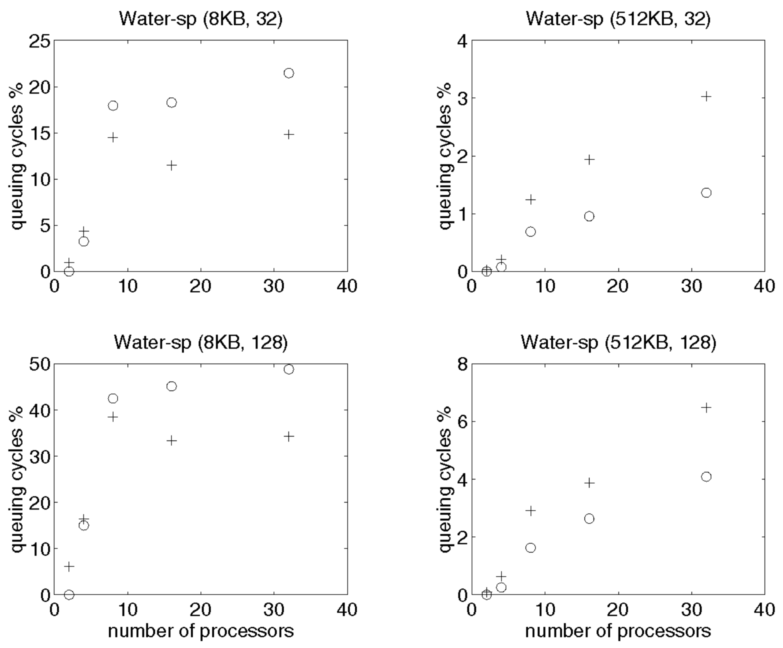

Fig. 4. Water-sp: "+" indicates the result from simulation. "0" indicates the result computed by the model.

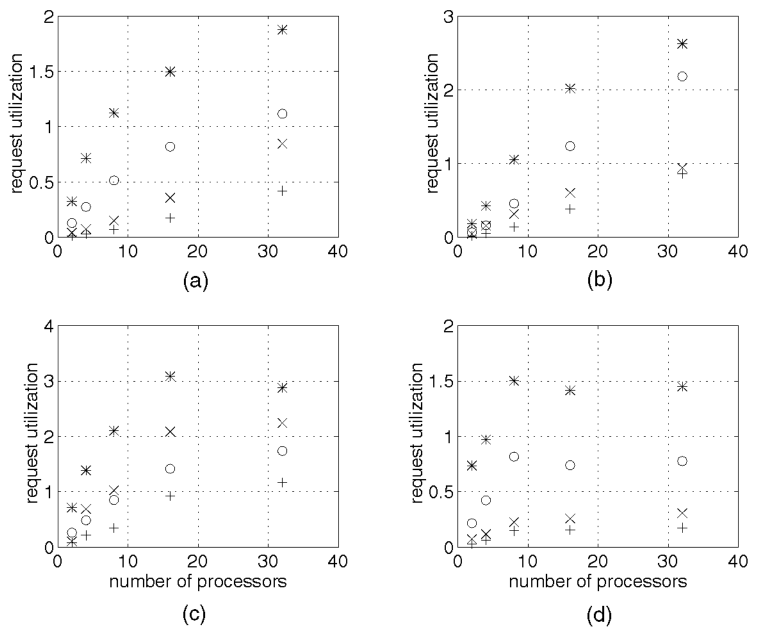

Fig. 5. Request utilization for 13 (37) CPU clocks in loading a 32-byte (128) cache line. (8KB, 128) = *, (8KB, 32) = o, (512KB, 128) = x, (512KB, 32) = +. (a) BArnes, (b) Radix, (c) Ocean, (d) Water-sp.

notations that represent part of the execution time cycles as designated in Table 6.

We characterize the CPU execution cycles into nonmemory and memory reference cycles. They are represented by

$$C_e = E_i \cup \left(E_m - \frac{R}{L}\right)h_t \cup \left(\frac{R(1+\alpha)}{D} + I_n\right)\beta_m, \qquad (11)$$

where the total execution cycle C_e is the union of three major cycle counts, that is, from the total execution cycles for nonmemory instructions, the cache hit cycles, and the external memory reference cycles. These three cycle counts may overlap with each other depending on the hardware used, especially for current dynamic execution machines. In (11), $(E_m - R/L)h_t$ represents the cycles of load/store instructions that hit in the data cache since R/L represents the number of load/store instructions that cause cache misses. If instruction fetching contributes to the execution time, the effect is similar to increase R. For an in-order issue and in-order completion scalar processor, the execution time E_S can be simply represented as

$$E_S = E_i + \left(E_m - \frac{R}{L}\right)h_t + \left(\frac{R(1+\alpha)}{D} + I_n\right)\beta_m. \qquad (12)$$

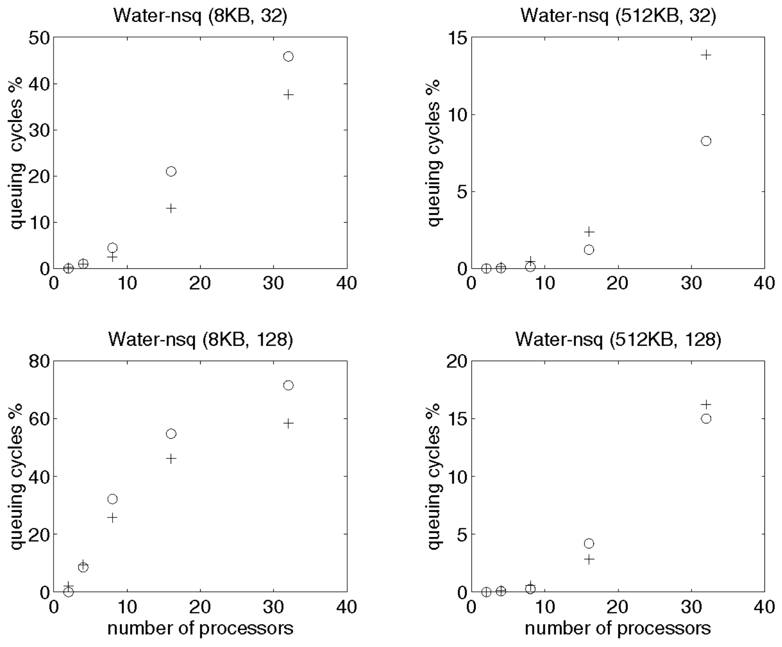

Fig. 6. Water-ns: "+" indicates the result from simulation. "o" indicates the result computed by the model.

In this case, we assume that the cache is full blocking and there are no write buffers. Therefore, the miss and write-back cycles all contribute to the execution time. If we improve the system by using certain hardware mechanisms, such as superscalar capability, nonblocking cache, bypassing write buffers, or a combination of these, the net effect scales the execution time E_S with a speedup factor, sp. In this system, the improved execution time is

$$E_I = \left(Ei + \left(E_m - \frac{R}{L}\right)h_t + \left(\frac{R(1+\alpha)}{D} + I_n\right)\beta_m\right) \times sp, \tag{13}$$

where $0 < sp < 1$. To focus on the trade-offs among caching, bus width, and bus cycle time, we consider the cycle times represented by (13) for a speed up factor of sp. In this way, we isolate the effect of other architectural factors on the execution time. In a uniprocessor system, one can design to obtain the same performance (execution time) by using either the combination of a larger cache (better caching)

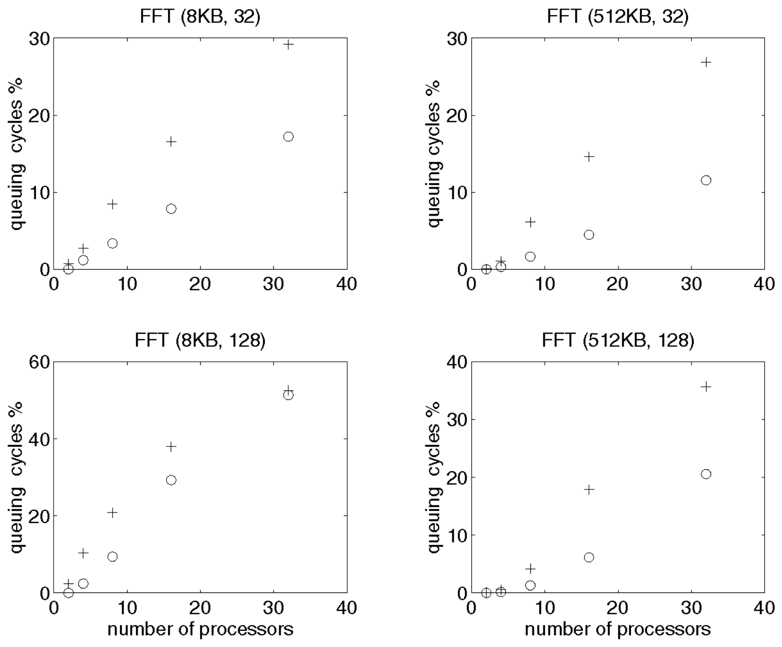

Fig. 7. FFT. "+" indicates the result from simulation. "o" indicates the result computed by the model.

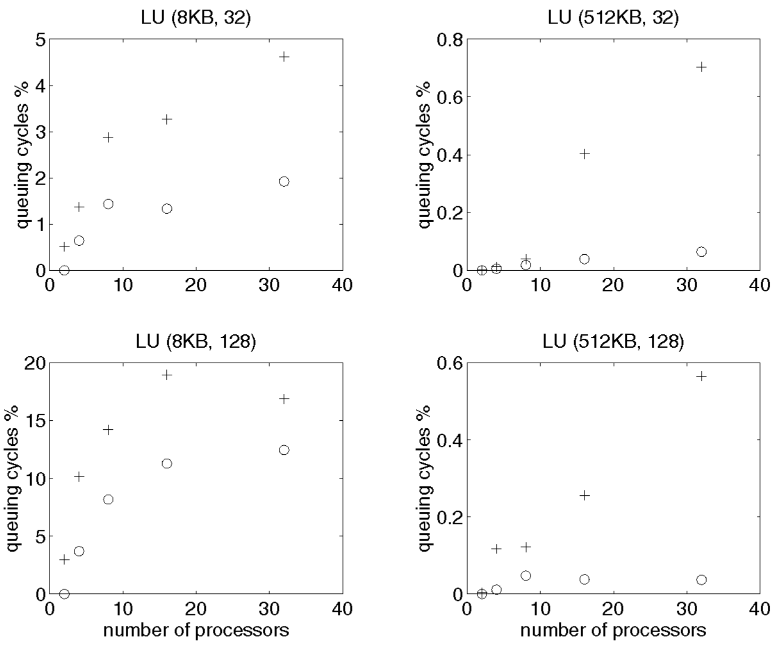

Fig. 8. Lu: "+" indicates the result from simulation. "o" indicates the result computed by the model.

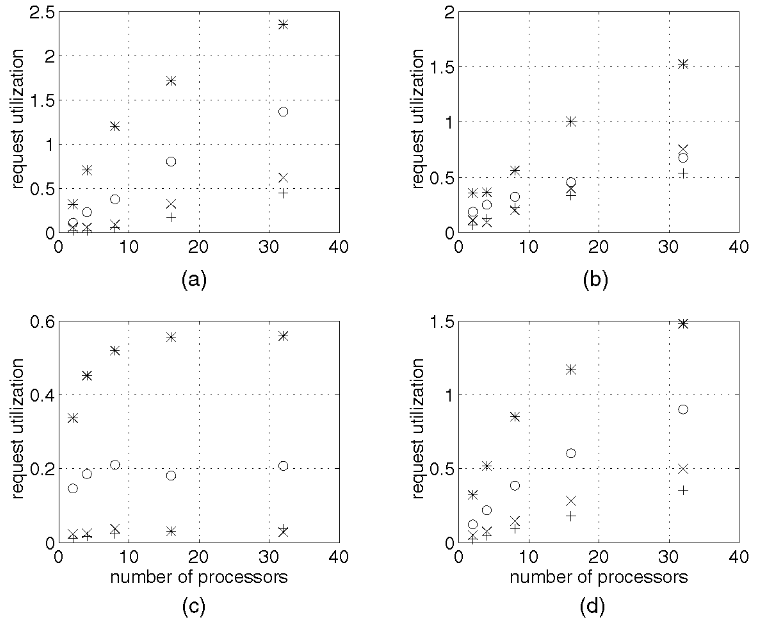

Fig. 9. Request utilization continued for 13 (37) CPU clocks in loading a 32-byte (128) cache line. (8KB, 128) = *, (8KB, 32) = o, (512KB, 128) = x, (512KB, 32) = +. (a) Water-nsq, (b) FFT, (c) LU, (d) Average.

with a slower memory or the combination of a smaller cache with a faster memory.

Suppose that we have two systems that have the same execution time in a uniprocessor setting. The first is System I that uses a larger cache and a slower memory, while the second is System II that uses a smaller cache and a faster memory. For System I, the larger cache results in a smaller M_C than System II. Let the execution time of System I be represented by E_1 as follows:

$$E_1 = \left(E_{i1} + \left(E_{m1} - \frac{R_1}{L}\right)h_t + \left(\frac{R_1(1+\alpha_1)}{D} + I_{n1}\right)\beta_{m1}\right) \times sp \tag{14}$$

and the execution time of System II be represented by E_2 given by

$$E_2 = \left(E_{i2} + \left(E_{m2} - \frac{R_2}{L}\right)h_t + \left(\frac{R_2(1+\alpha_2)}{D} + I_{n2}\right)\beta_{m2}\right) \times sp. \tag{15}$$

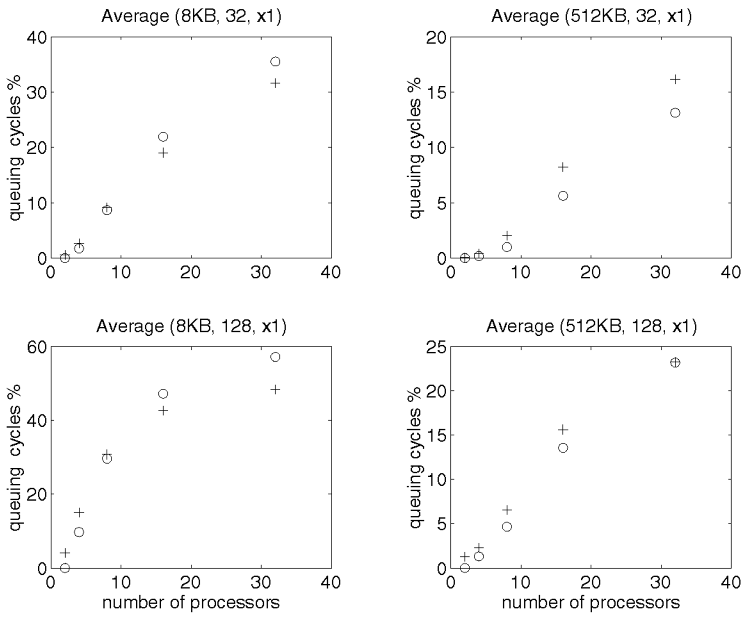

Fig. 10. This result shows the average queuing delays of the seven SPLASH-2 programs with x1 latency in loading a cache line.

If $E_1 = E_2$, we can find design parameters such that $M_{C2} > M_{C1}$ and $\beta_{m2} < \beta_{m1}$ with the conditions that $E_{i1} = E_{i2}$ and $E_{m1} = E_{m2}$. These conditions ensure that the same application is run for comparison. Examining $E_1 = E_2$, it is clear that the equality is based on the memory delay time, which is the sum of the cache hit and miss cycles.

In theory, given a design specified by $\{M_{C1}, \beta_{m1}\}$ for System I, we can determine a value of M_{C2} (or β_{m2}) for System II that uses β_{m2} (or M_{C2}) such that the two systems have the same execution time. Suppose that the average number of memory reference per cycle for System I is 0.01 and the bus cycle time is 10 CPU clocks. For a uniprocessor setting, the design parameters for System II that has the same performance as System I are shown in Fig. 13a. The y axis is the M_{C2}, while the x axis is the bus cycles. Each (x, y) coordinate indicated by the "x" sign on the curve specifies the design parameters for System II that has the same performance as System I. Alternatively, the systems with the M_C and the bus cycle time specified by the "x" points have the same performance in a uniprocessor setting.

For the above example, the design parameter for System I is located at (10, 0.01) designated with a circle in Fig. 13a. If System II uses an M_C of about 0.02, it means that the cache used by System II is smaller than that of System I. To

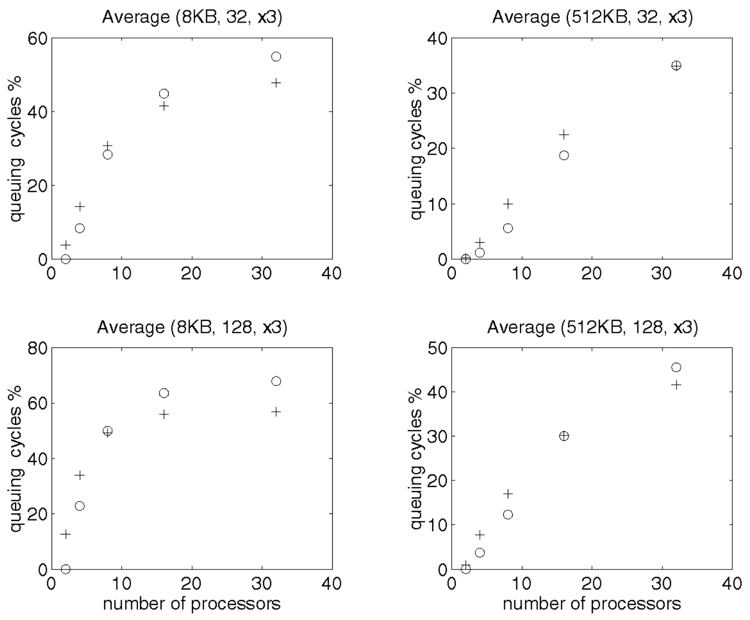

Fig. 11. The result shows the average queuing delays of the seven SPLASH-2 programs with x3 latency in loading a cache line.

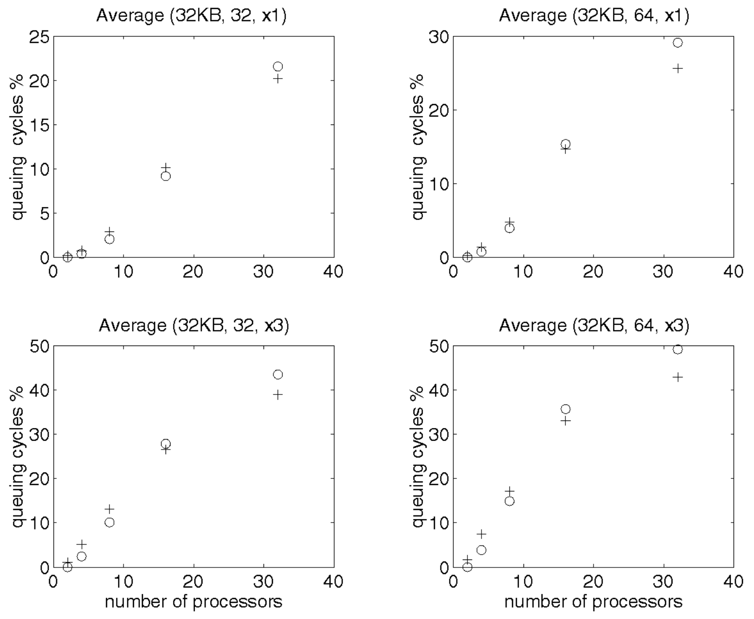

Fig. 12. Using a 32-KB cache in each processor, the result shows the average queuing delays of the seven SPLASH-2 programs.

TABLE 4
Error of Average Value and Mean Square Value for Memory Latency Model x1

	$\frac{\sum_{i=1}^{N}(M_i-S_i)}{N}$ (ave. value)					$\frac{\sqrt{\sum_{i=1}^{N}(M_i-S_i)^2}}{N}$ (mean square val.)				
No. of CPU	2	4	8	16	32	2	4	8	16	32
(8KB,32,x1)	-0.62	-0.96	-0.51	2.87	3.9	0.27	0.50	1.72	2.59	3.30
(8KB,128,x1)	-4.13	-5.35	-1.13	4.53	8.87	1.90	2.53	2.73	3.54	4.39
(32K,32,x1)	-0.18	-0.35	-0.81	-0.98	1.37	0.08	0.18	0.85	1.45	2.47
(32K,64,x1)	-0.27	-0.65	-0.80	0.64	3.50	0.12	0.37	1.28	2.02	3.25
(512K,32,x1)	-0.04	-0.24	-1.03	-2.60	-3.02	0.02	0.13	0.70	1.67	2.36
(512KB,128,x1)	-1.24	-0.96	-1.87	-2.06	-0.08	1.19	0.60	1.84	2.78	3.03

TABLE 5
Error of Average Value and Mean Square Value for Memory Latency Model x3

	$\frac{\sum_{i=1}^{N}(M_i-S_i)}{N}$ (ave. value)					$\frac{\sqrt{\sum_{i=1}^{N}(M_i-S_i)^2}}{N}$ (mean square val.)				
No. of CPU	2	4	8	16	32	2	4	8	16	32
(8KB,32,x3)	-3.76	-5.93	-2.44	3.35	7.15	1.55	2.64	2.52	2.73	3.67
(8KB,128,x3)	-12.62	-11.18	0.64	7.66	11.11	5.00	4.67	1.86	3.44	4.46
(32K,32,x3)	-1.10	-2.78	-2.95	1.35	4.57	0.49	1.44	2.35	2.20	2.75
(32K,64,x3)	-1.65	-3.63	-2.24	2.59	6.25	0.73	1.94	2.52	2.68	3.21
(512K,32,x3)	-0.26	-1.87	-4.41	-3.80	0.12	0.18	1.04	2.51	2.36	2.06
(512KB,128,x3)	-1.01	-4.07	-4.74	-0.12	3.92	0.64	2.41	3.17	2.53	2.40

achieve the same performance, System II must use a faster main memory system. The bus cycle time for System II in this case is 5, which can be found on the left side of Fig. 13a. On the other hand, if System II uses a larger cache than System I, it can use a slower main memory for the same performance. For this situation, the design parameters can be found on the right side of the curve.

Fig. 13b illustrates the execution time if the corresponding design parameters (in Fig. 13a) are used in a multiprocessor configuration. The execution time including

TABLE 6
Execution Time Parameters

E_i	the total execution cycles for all internal (non-memory reference) instructions.
E_m	the number of load/store instructions.
h_t	cache hit cycle time normalized with the CPU clock cycle.

queuing delay is computed with (6) and normalized with parameter C_e. Eight processors are used in this example.[2] Observe that if System II is designed with the parameters on the left side of the curve in Fig. 13a, System II will have a higher execution time than System I. On the other hand, if System II uses a design with the parameters on the right side of the curve in Fig. 13a, System II will have a higher performance than System I. This is due to the difference in queuing overheads incurred in multiprocessor configurations, as given in Fig. 13c. For instance, the queuing overhead for System II that uses $M_{C2} = 0.02$ and $\beta_{m2} = 5$ is about 17.8 percent. This overhead is greater than System I, which has about 17.3 percent, as given in Fig. 13c. If System II uses a larger cache and a slower memory than System I, then the queuing overhead of System II will be smaller than System I. Observe that the difference of queuing delays between the two systems increases as the designs use parameters located on the two ends of the curve.

The result here indicates that the better choice for a shared-bus multiprocessor system is the one with a larger cache (smaller M_C). The reason that a slower memory and a larger cache outperform the combination of a faster memory and a smaller cache in an MP system is because the first combination results in a lower bus request utilization. Fig. 13d illustrates this evidence where the request utilization for the combination of a smaller cache and faster memory is higher than the combination of a larger cache and a slower memory. Thus, we have the following theorem.

Theorem 1. *Given the same performance from the combination of a faster memory and a smaller cache and the combination of a slower memory and a larger cache in a uniprocessor setting, the choice for higher performance in a shared data path MP configuration is the combination that uses a slower memory and a larger cache.*

Proof. The reason for the above conclusion is due to the lower bus request utilization for the combination that uses a slower memory and a larger cache, as shown in Fig. 13d. □

Another scenario that has the same performance in a uniprocessor setting is when the bus width is changed for a given bus cycle time. For instance, suppose that a system with a wider 128-bit data bus and a smaller cache and a system with a 64-bit data bus and a larger cache have the same performance in a uniprocessor system. The question of which configuration is more favorable in an MP system in terms of performance is answered in the following theorem.

Theorem 2. *If it is given that the performance from the combination of a wider data path and a smaller cache is the same as that achievable from the combination of a narrower data path and a lager cache for the same bus cycle time in a uniprocessor setting, the choice for higher performance in a shared data path MP configuration is the combination that uses a narrower data path and a larger cache.*[3]

Proof. The system that uses a wider data path for a given bus cycle is equivalent to using a faster memory instead of the wider data path. Similarly, the system that uses a narrower data path can be replaced with an equivalent slower memory for the narrower data path. From Theorem 1, the combination that uses a slower memory (i.e., narrower data path) and a larger cache results in lower bus request utilization. Therefore, the choice for higher performance in a shared data path MP configuration is the combination that uses a narrower data path and a larger cache. □

From the above analysis, we conclude that the same performance level can be achieved by trading among the memory cycle time, the bus width, and the cache memory size in a uniprocessor system, but the favorable choice for a shared-bus multiprocessor system is the combination that uses a larger cache. In other words, the combination of more requests with a faster response results in a lower performance than the combination of fewer requests and a slower response. More generally, we can state the following result.

Theorem 3. *Given the same performance in terms of total number of cycles for either caching the requests or speeding up the transfers in a contention-free situation, the design favors caching the requests in a contention-based medium.*

5 Conclusion

We have presented an analytical model for the design of contention-based multiprocessor systems. The model relates the shared-bus width, bus cycle time, cache memory, the features of a program execution, and the number of processors to a metric called request utilization. The model of bus contention is verified with a detailed cycle-level simulation using the SPLASH-2 parallel programs. The mean square of error value is less than 5 percent between

2. The replacement ratio α is assumed to be 0.5, which is a typical value observed from the simulation of this study as well as from our early work [19], [20]. The number of invalidations is very small, so we assume zero in the computation.

3. In this paper, the cache size is chosen from 8KB to 512KB, and the average bus cycle time ranges from one CPU clock cycle to 20 CPU clock cycles. Note that all traffic in this study is assumed to be cacheable.

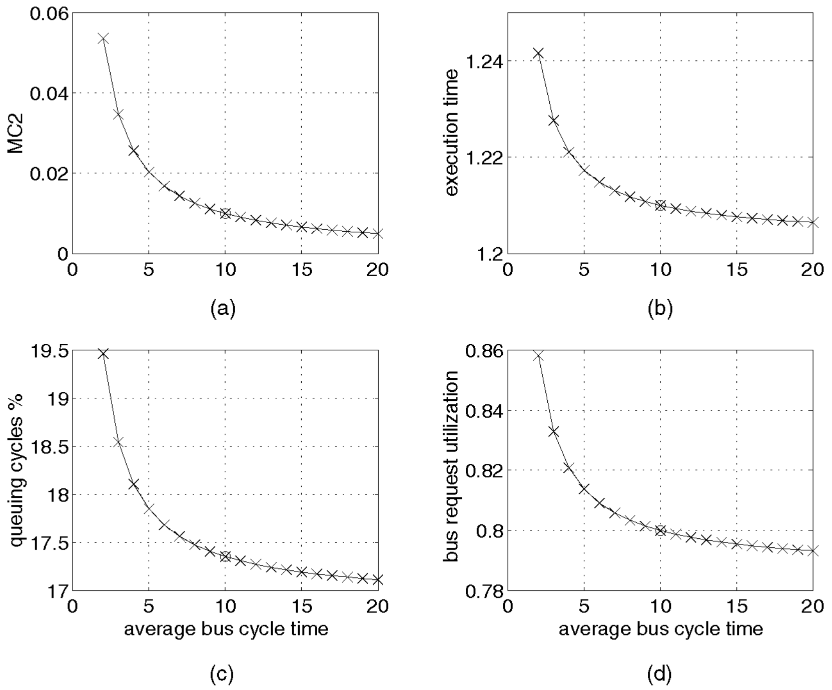

Fig. 13. Performance metrics of multiprocessor configurations given the same performance in a uniprocessor setting. ($M_{C1} = 0.01$ and $\beta_{m1} = 10$ for System I. The number of processors = 8, L = 32 bytes, D = 8 bytes, $\alpha_1 = \alpha_2 = 0.5$, $I_{n1} = I_{n2} = 0$.)

the results of the model and the simulation. The model can be used to assist the design of a bus-based MP system by finding a satisfactory request utilization or bus response time.

Using the model developed in this paper, we have obtained the following specific results. First, given the same performance in a uniprocessor system, a slower memory and a larger cache outperform the combination of a faster memory and a smaller cache in shared-bus multiprocessor systems. Similarly, given the same performance in a uniprocessor system, the combination of a larger cache with a narrower shared bus has a better performance than the combination of a smaller cache and a wider shared bus. More generally, this work has shown that, given the same performance in terms of total cycle count for either caching the requests or speeding up the transfers in a contention-free situation, the design favors caching the requests for a contention-based medium. In this work, we have studied the system design for a representative suite of parallel programs. An MP system involves the use of many other applications. A designer can use the approach presented here to evaluate the design so that the system can best meet the need of these applications.

Acknowledgments

We thank Prof. Arun Somani for reading and commenting on an early draft of this paper. Also, thanks to the anonymous referees who provided helpful inputs to improve the work. This work was supported in part by the funding of NSC85-2213-E-224-021, Taiwan, Republic of China.

References

[1] M. Rosenblum, S.A. Herrod, E. Witchel, and A. Gupta, “Complete Computer System Simulation: The SimOS Approach,” *IEEE Parallel and Distributed Technology,* pp. 34-43, Winter 1995.

[2] C.A. Prete, G. Prina, and L. Ricciardi, “A Trace-Driven Simulator for Performance Evaluation of Cache-Based Multiprocessor System,” *IEEE Trans. Parallel and Distributed Systems,* vol. 6. no. 9, pp. 915-929, Sept. 1995.

[3] D. Thiebaut, J.L. Wolf, and H.S. Stone, “Synthetic Traces for Trace-Driven Simulation of the Cache Memories,” *IEEE Trans. Computers,* vol. 41, no. 4, pp. 388-410, Apr. 1992.

[4] R. Giorgi, C.A. Prete, G. Prina, and L. Ricciardi, “A Hybrid Approach to Trace Generation for Performance Evaluation of Shared-Bus Multiprocessor,” *Proc. 22nd EUROMICRO,* pp. 207-214, 2-5 Sept. 1996.

[5] B.L. Bodnar and A.C. Liu, “Modeling and Performance Analysis of Single-Bus Tightly-Coupled Multiprocessors,” *IEEE Trans. Computers,* vol. 38, no. 3, pp. 464-470, Mar. 1989.

[6] M.K. Vernon, E.D. Lazowska, and J. Zahorian, “An Accurate and Efficient Performance Analysis Techniques for Multiprocessor Snooping Cache-Consistency Protocols,” *Proc. 15th Int'l Symp. Computer Architecture,* pp. 308-315, May 1988.

[7] M.C. Chiang and G.S. Sohi, “Evaluating Design Choices for Shared Bus Multiprocessors in a Throughput-Oriented Environment,” *IEEE Trans. Computers,* vol. 41, no. 3, pp. 297-317, Mar. 1992.

[8] M.A. Holliday and M.K. Vernon, “Exact Performance Estimates for Multiprocessor Memory and Bus Interference,” *IEEE Trans. Computers,* vol. 36, no. 1, pp. 76-85, Jan. 1987.

[9] A.K. Somani, C.M. Wittenbrink, R.M. Haralick, L.G. Shaprio, J.N. Hwang, C.H. Chen, R. Johnson, and K. Copper, “Proteus System Architecture and Organization,” *Proc. Fifth Int'l Parallel Processing Symp.,* pp. 287-294, 1991.

[10] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” *Proc. 22nd Int'l Symp. Computer Architecture,* pp. 24-36, June 1995.

[11] K. Sakamura, R. Sano, and K. Honma, “Introducing Tobus, the System Bus in the TRON Architecture,” *IEEE Micro,* pp. 47-59, Apr. 1988.

[12] T. Lovett and S. Thakkar, “The Symmetry Multiprocessor System,” *Proc. Int'l Conf. Parallel Processing,* pp. 303-310, 1988.

[13] S. Thakkar, P. Gifford, and G. Fielland, "The Balance Multiprocessor System," *IEEE Micro,* Feb. 1988.
[14] A.J. Smith, "Cache Memories," *ACM Computer Surveys,* vol. 14, pp. 473-530, Sept. 1982.
[15] J.L. Hennessy and D.A. Patterson, *Computer Architecture, A Quantitative Approach,* second ed. Morgan Kaufmann, 1996.
[16] C.-H. Chen, "Exploring the Design Space of Cache Memories, Bus Width, and Burst Transfer Memory Systems," *J. Chinese Inst. of Engineers,* vol. 21, no. 3, pp. 269-282, 1998.
[17] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queuing Networks Models.* Prentice Hall, 1984.
[18] A. Sharma, N.-T. Nguyen, and J. Torrellas, "Augmint—A Multiprocessor Simulation Environment for Intel x86 Architectures," CRD Technical Report 1463, Univ. of Illinois at Urbana-Champaign, 1995.
[19] C.-H. Chen and A.K. Somani, "A Unified Architectural Tradeoff Methodology," *Proc. 21st Int'l Symp. Computer Architecture,* pp. 348-357, Apr. 1994.
[20] C.-H. Chen and A.K. Somani, "Architecture Technique Trade-Offs Using Mean Memory Delay Time," *IEEE Trans. Computers,* vol. 45, no. 10, pp. 1,089-1,100, Oct. 1996.

Chung-Ho Chen received his MSEE degree from the University of Missouri-Rolla in 1989 and the PhD degree in electrical engineering from the University of Washington, Seattle, in 1993. He was with the Department of Electronic Engineering, National Yunlin University of Science and Technology from August 1993 to July 1999. He is currently an associate professor with the Department of Electrical Engineering, National Cheng Kung University. His research interests include parallel computer architecture, VLSI systems, data communication, and distributed computer systems. Dr. Chen is a member of the IEEE Computer Society.

Feng-Fu Lin received his MS degree in electronics and information engineering from the National Yunlin Institute of Technology in 1996. He is currently with the R&D Department of the Great China Technology Inc., Taiwan.