ELSEVIER

Performance Evaluation 23 (1995) 31–52

PERFORMANCE EVALUATION
An International Journal

Performance evaluation of a cache-coherent multi-processor by iterative mean value analysis

Chien-Yuan Huang [1], Shi-Chung Chang *, Chern-Lin Chen

Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, ROC

Received January 1993; revised November 1993

Abstract

An iterative mean value analysis (MVA) algorithm is developed in this paper for a class of queueing networks (QNs) with forking (MVAF). The algorithmic development is motivated by the performance evaluation of a Multicube multiprocessor system and its cache coherent protocol proposed by Goodman and Woest, 1988. We model the system as a multichain, multiclass, mixed QN with forking phenomena. The MVAF algorithm that we propose approximates a forking by independent flows at each iteration and iteratively applies MVA to the approximate QNs without forking. We prove that MVAF converges to the exact performance measures for the QN with forking. Comparisons among performance evaluations by MVAF, simulation and the mean value performance analysis by Leutennegger and Vernon, 1988, demonstrate both the validity and effectiveness of MVAF.

Keywords: Performance evaluation; Multiprocessor; Cache coherence protocol; Queueing network model; Forking; Mean value analysis; Fixed-point iteration; Simulation

1. Introduction

The design of a high performance multiprocessor system is very complex and requires integration of various hardware and software technologies [11]. Designer's intuitions and experiences alone are not sufficient for designing such a complex system. To achieve a high performance/cost ratio system, namely, the competitiveness in the market, mathematical modeling for performance prediction and evaluation has become indispensable in the design process [1,12].

* Corresponding author.
[1] Currently a Ph.D. candidate of Department of Electrical Engineering, Pennsylvania State University.

SSDI 0166-5316(94)E0062-A

Mathematical models for computer systems range from relatively simple ones whose solution may be obtained analytically (e.g., product form queueing networks and the Mean Value Analysis), to very complex ones that are not analytically tractable and simulation is usually adopted to evaluate them (e.g., Timed Stochastic Petri Net) [1,12]. These models may each be useful at different stages of design. The former are usually fast in computation and provide insights for the first-cut design while the latter are time consuming but generate results closer to the reality.

In the earlier phases of developing a multiprocessor computer system, timeliness of decisions and a good grasp of key system characteristics are more important than accuracy of decisions to shorten the design cycle and to achieve the success of a design. For this purpose, what design engineers need are rapid modeling and performance prediction/evaluation tools that provide timely evaluations and help narrow down options for the later design phases. Such needs have therefore posed challenges to the development of advanced modeling, analysis and performance evaluation methodologies for designing a large-scale and complex multiprocessor computer system.

There have been many results in the literature on performance evaluation of multiprocessor systems with cache coherence protocols. Archibald and Baer presented a simulation model to compare various decentralized protocols [3]. A generalized Petri net technique was adopted in [10]. The computational complexity of the technique, however, makes it unsuitable for the application to large-scale systems. There are many analytical queueing network models and their corresponding approximate mean value analysis (MVA) proposed [8,12,13] for evaluating cache coherence protocols for a global-bus multiprocessor system [20].

Vernon et al. [14,19] developed a Mean Value Performance Analysis (MVPA). Their analysis is founded on system designer's good understanding of the computer system and their very basic skill on probabilistic modeling. Once a model is constructed, the analysis amounts to solving nonlinear equations in variables of mean performance measures. This approach has been applied to analyzing a Multicube architecture with a cache coherent protocol [8], which is more complex than the global-bus multiprocessor described in [10,20]. Though simple and intuitive, this approach has some disadvantages in effectiveness for performance evaluation due to modelling simplifications.

In this paper, we also adopt the two-dimensional Multicube architecture proposed in [8] as a study system, develop a queueing network-based performance evaluation approach for it, and compare our approach with MVPA. The 2-D Multicube architecture employs a snooping cache system over a grid of buses. It has a cache coherence protocol to maintain data consistency among snooping caches of different processor modules. While providing a view of a single shared memory to the programmer, it imposes no notion of geographical locality. The 2-D Multicube is therefore regarded as a potentially general purpose multiprocessor which supports a wide range of applications.

A mixed queueing network (QN) model is first constructed for this 2-D Multicube system, where processors, buses and shared memories are modeled as servers while data block transactions among processors are modeled as the customers. The snooping cache coherence protocol determines the routing and class changes of customers in the network. In this QN model, there are finite buffers and forking phenomena during data transactions. Among the existing numerical algorithms for queueing network performance evaluation, mean value

analysis (MVA) algorithm is an exact solution technique for the product form queueing networks [12,17]. However, the QN/model of the 2-D Multicube does not have a product form due to (1) finite buffers which may causes blocking and (2) the existence of forking phenomena.

Exploiting problem features, we first assume that the effect of finite buffers to system performance can be neglected. We then develop an iterative MVA algorithm for a QN with forking (MVAF). In each iteration of the MVAF, a forking is approximated by two independent flows of open class and closed class of customers; the MVA algorithm for multichain, multiclass, mixed QNs is then applied to this approximated QN. The MVAF is essentially a Jacobi type of algorithm for finding the fixed point of the arrival rates of open-class customers in the QN model. We also prove that MVAF is a contraction mapping and converges to a unique solution, which is in commensurate with the results of [15].

The MVAF algorithm is validated by comparing its performance evaluation results with simulation results for the QN models of small scale (2×2 and 3×3) 2-D Multicube systems. Results from MVAF and simulation are very close, e.g., less than 5% difference in the measure of mean processor utilization. It seems that the larger the system, the smaller the difference. Performance evaluation results by using MVPA on the same sets of problems are similar in general trends but tend to be over optimistic, especially under high workload situations. MVAF is further applied to performance evaluation of large-scale systems and is compared to MVPA. Similar comparison conclusions are obtained except that the magnitude of differences may go up to 40%. Such differences are due to the superior modeling and analysis of our QN and MVA-based approach. The computation times for MVAF and MVPA are about in the same order. Though our approach requires some background on QN, we believe that the techniques developed in this paper are not only effective for performance evaluation of the 2-D Multicube system but also have a good potential for further extensions and applications.

The remainder of this paper is organized as follows. Section 2 briefly describes the architecture and the cache coherence protocol of the 2-D Multicube. Its mixed queueing network model is constructed in Section 3. Section 4 develops the MVAF algorithm with convergence analysis given in Section 5. Section 6 then provides some validations of the MVAF by simulation. Performance analysis of the 2-D Multicube using both MVAF and MVPA are also compared. Conclusions are finally given in Section 7.

2. A 2-D Multicube architecture and protocol

In this section, we briefly summarize, for our modeling and performance evaluation purpose, the architecture of the 2-D Multicube and its cache-coherent protocol proposed by [8].

2.1. System architecture

The 2-D Multicube architecture employs a snooping cache system over a grid of buses as shown in Fig. 1. Each processor is connected to a multi-level cache. The first-level cache, referred to as the processor cache, is a high performance cache designed with the traditional goal of minimizing memory latency of data accessing by the processor. A second-level cache, referred to as the snooping cache, is a large cache with a minimum of 64K DRAM designed to

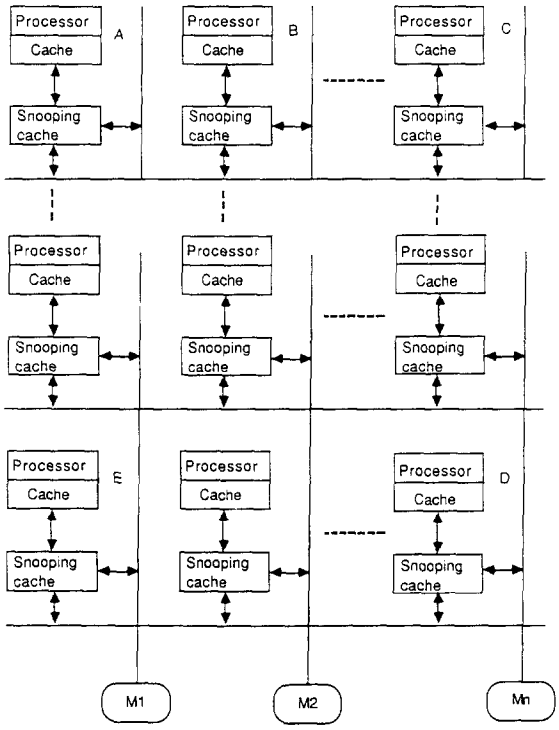


Fig. 1. The 2-D Multicube computer architecture.

minimize bus traffic between the local and shared memories [7]. Consistency between the two cache levels is maintained by using a write-through strategy to assure that the processor cache is always a strict subset of the data in the snooping cache. Every data block in the system has a copy kept by one of the shared memory modules, which is considered its home memory module. The controller of each snooping cache also monitors a row bus and a column bus in order to maintain data consistency among snooping caches and shared memories. The interconnection topology may be viewed as a collection of conventional single-bus multiprocessors connected by an orthogonal set of buses which transparently extends the snooping cache protocol to higher dimensions.

2.2. *The cache coherence protocol*

The 2-D Multicube architecture allows a cache-coherent protocol for which most bus requests can be satisfied with no more than twice the number of bus operations required for a single-bus multiprocessor. There are three states of a data block in a memory according to [8]: modified, shared and invalid.

(1) *modified:* blocks that were written by a processor, but have not been read or written by any other processors,

(2) *shared:* a cache block that does not exist in any cache in state modified,

(3) *invalid:* a cache block that has at least one of its copies in the system modified by any other processor.

Note that if a block is in state modified, no other caches have such a copy. Each snooping cache controller maintains a modified block table (MBT) that contains addresses of all cache blocks that are in state modified in any one of the caches along the same column as the snooping cache. Main memory contains the current values of all shared cache blocks.

A cache miss happens when the data in the snooping cache can not support the processing of the processor in the same module. There are two types of cache misses: a *read miss* happens when a processor only reads the requested data while a write miss occurs when a processor intends to modify the requested data.

Under the cache-coherent protocol, four types of cache block transactions may be initiated by a snooping cache controller associated with a cache miss: READ, READ_MOD, WRITE_BACK and INVALIDATION, which are described as follows.

READ transactions

Suppose that a controller, say controller A in Fig. 1, issues to its row bus a READ request for a block not in the caches of processor A. If the requested block is in a shared state, a controller located at the intersection of the row bus and the column connecting to the home module of the requested block accepts the request, say controller B in this case. If B has the requested block in its cache, it sends the data to A; else it requests the column bus and passes the request to the home memory (M2). The memory controller of M2 then responds by sending the requested data to the requesting controller if the block is in a valid state. Otherwise, the memory controller takes no actions.

If the requested block is in a modified state, then one controller (e.g., C) located at the intersection of the requesting row and the column along which a cache keeps the requested block will accept the request after looking into its MBT. Controller C then relays the request to its column bus. All controllers on the column delete the block address from their MBT and the controller of the snooping cache that has the data block (D) fetches the data, changes its state from modified to shared and returns the data to A through the reverse route. Along this returning path to A, a controller on its home column (B) will also pick up a copy of the data block, and send it to the home memory (M2) with its state changed to "shared".

READ_MOD transactions

A controller, say A, issues a READ_MOD request to its row bus when a write miss occurs. If the requested block is "modified", the transaction procedure is the same as that for the READ transaction except that the return path is D–E–A. When controller E forwards the block to A in a modified state, all controllers along the same column add the block address to their MBTs.

If the requested block is "shared", the steps are that after the READ_MOD request is forwarded to it home memory (A–B–M2), M2 returns the requested data through its column bus and an appropriate row bus to A, and initiates an invalidation transaction concurrently to ensure that other copies of the block in the system are purged. After the reception at the requesting controller, a column bus operation is performed (by controller A) to update each MBT along the same column with the address of the newly acquired data block.

WRITE_BACK transactions

A WRITE_BACK transaction occurs during the returning procedure of a READ transaction for a modified data block. When the required data block is transmitting on the row bus of

the requesting controller, the controller at the intersection of the block's home column and the row will also pick up the data, and relay it to its home memory. The data block is then written back to the memory and changed to a "shared" state.

INVALIDATION transactions

An invalidation request is sent along with the requested data block in its returning path for a READ_MOD transaction of a "shared" data block. The invalidation request is broadcasted to all controllers by controllers along the returning path so that other copies of the data are purged.

3. A queueing network simulation model

To evaluate the performance of the 2-D multicube architecture and its protocol, a queueing network-based simulation environment is first established. A queueing network consists of different classes of customers, servers, buffers, interconnection of servers/buffers and service discipline at each server. In modeling the 2-D multicube system, servers correspond to hardware resources, buffers correspond to memory storages and customers correspond to data block transactions.

Now consider a processor module, where the processor and its first-level cache are busy with processing except when a cache miss occurs at the processor module. In that occasion, the processor generates a request for data block transaction which goes through the snooping cache controller to the interconnection bus. As uniprogramming is assumed at each processor, the processor then holds until the requested data is acquired. So we model the processor and the first-level cache as a *processor server* with no buffer, where the departure of a customer corresponds to the generation of a cache miss request and the server stays idle until the customer returns. The snooping cache and controller of the processor module not only handle transactions from and to their corresponding processor module but also relay transactions among other processor modules through buses. We therefore model the controller as a *snooping cache server* and the cache as an infinite buffer associated with it since the snooping cache is relatively large.

A data block transaction may involve the access of a few row and column buses. If a bus is being accessed by a transaction and there is another transaction requesting the bus at the same time, the requesting transaction waits in its current snooping cache server (i.e., it is blocked) until the bus is free. When a transaction accesses a main memory module through its home column bus, it ties up both the memory module and the bus. We therefore model a column bus and its associated memory module into a *column bus server* and model a row bus as a *row bus server*, where all bus servers are of no buffer spaces.

Data transaction requests due to cache misses at a processor are modeled as *closed-class customers* in our QN model. This is based on the fact that a processor server generates one transaction request at a time and holds until the request is fulfilled to resume its processing. As there are four types of transactions and there are also various routing requirements among transactions, we further define classes of customers and the interconnection among servers according to the cache coherence protocol.

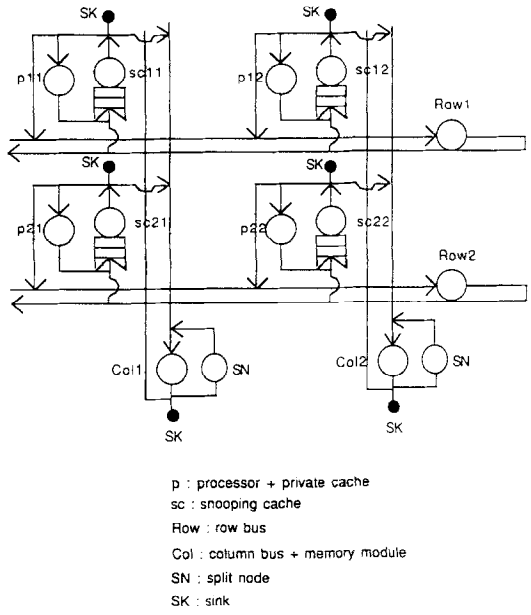


Fig. 2. The mixed queueing network model.

The class of a customer may be changed upon its departure from a server when there are alternative routes, with the *new class label* specifying the next server to visit and the service type to receive. A customer may split into a number of new customers after certain type of services, for instance, the broadcasting of invalidation signals due to READ_MOD transactions, write-back operations due to READ transactions, etc. A split function is built into the corresponding servers. As the customers originated from invalidation and write-back transactions do not go back to their generation server, sink nodes are created to absorb them after the completion of their required services. Such customers are naturally modeled as *open-class customers* and the model for the 2-D Multicube is therefore a mixed queueing network which consists of multiple closed classes of customers and open classes of customers as shown in Fig. 2.

Figure 3 gives an example illustrating the various paths of a customer generated by a processor module in a 2-D Multicube system. Assume that a program (or a task) is being executed in processor p11 and is modeled as a closed-class customer p. When there happens a cache miss, the customer goes to sc11 snooping controller server and changes class from p to pc. Because we use the probabilistic workload model, each alternative routing is associated with a probabilistic distribution. If this cache miss requests a shared block, it broadcasts the requested cache block address on the row1 bus and becomes a class s11 customer. Suppose that the snooping cache controller sc12 detects the request and forwards it to one of the memory modules along co12 bus to access the data. The customer class changes from s11 to m11 during memory accessing. After accessing, the customer is probabilistically routed to two possible routes: one for a READ only request and one for a READ_MOD request. For the latter, the invalidation customers are spawned to do the invalidations in sc21 and sc22 and go to the sink

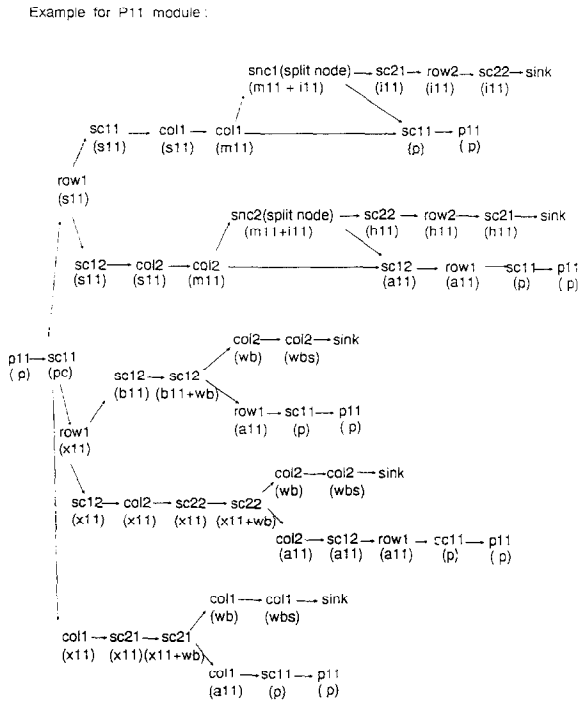


Fig. 3. Example of paths for a customer generated by a processor module.

after the invalidation; the requested block is returned to processor p11 following the reverse path direction of the request. Other cumbersome details in constructing Fig. 3 are omitted.

4. Mean value analysis

As there exist (1) finite buffers and (2) forking phenomena, the QN model developed in Section 3 is not a product form QN. Existing mean value analysis (MVA) algorithms [12,17] cannot be directly applied to it for a quick performance evaluation. Approximate MVA algorithms for queueing networks with finite buffers, are limited to single-chain queueing networks [2,6]. Since the snooping cache of each processor module is assumed to be quite large, we expect only low to medium bus load due to cache miss. It in turn implies that there may be only minor blocking caused by bus and memory contentions at processor and snooping cache servers. We therefore neglect the blocking effect and approximate the QN by one with infinite buffers.

The difficulty remains to be overcome is then the forking phenomena. Observe that some open-class customers and a closed-class customer are generated after each forking in the QN, that all the open classes are generated by forking and that the generation rate of an open class depends on the throughput of its associated closed-class customer at the forking server. Such observations suggest a variations of the QN model where each open class of customers is generated by an external source node (rather than from forking) in an average rate equal to the

forking server's throughput. This variation facilitates the development of an iterative MVA algorithm for the QN model with forking.

The idea is that by first assuming independence between the open-class sources and the throughput of the closed-class at a forking server, open-class customers and closed-class customers in the approximate QN are decoupled. Under the exponential distribution assumption, the approximate QN then becomes a multi-chain, multi-class, product-form mixed QN. Given a set of open-class arrival rates, we can easily extend the existing MVA algorithms to it and compute the throughputs of closed-class customers at all forking servers. The mean arrival rates of open-class customers are then updated by the average throughputs of their associated closed classes respectively. This procedure iterates until the average rates of open-class arrivals converge.

To formalize the development of the MVA algorithm for the forking QN model (MVAF), let us first define some notations.

Notations:

R_c = number of chains in the QN,
J = number of service centers in the QN,
K = number of customer classes in the QN,
N_r = maximum population of chain r, $r = 1, \ldots, R_c$,
$\boldsymbol{N}$ = $(N_1, N_2, \ldots, N_{R_c})$,
n_r = population index of chain r, $r = 1, \ldots, R_c$,
$\boldsymbol{n}$ = $(n_1, n_2, \ldots, n_{R_c})$,
$\nu_{r,j,k}$ = mean number of visits to server j by a class-k customer of chain r,
$\tau_{r,j,k}$ = mean service time of a class-k, chain-r customer at server j,
$\tau^*_{r,j,k}$ = inflated mean service time of a closed, class-k, chain-r customer at server j,
$R_{r,j,k}$ = mean response time of a class-k chain-r customer at server j,
$\lambda_{r,j,k}$ = arrival rate of an open class-k chain-r customer at server j,
$L_{r,j,k}$ = dummy arrival rate of an open class-k chain-r customer at server j,
Q_j = mean queue size of server j,
TX_r = total throughput of chain r,
C = the set of closed-class customers,
O = the set of open-class customers,
U_j = mean utilization of server j,
$U_{j,O}$ = mean utilization of open class customers at server j,
$\boldsymbol{e}_r$ = a unit vector of dimension R_c with the rth element being 1,
δ = a convergence threshold.

As a mixed product form QN is evaluated in each iteration, we extend the existing MVA algorithm for mixed QN [12] to our multi-chain and multi-class case. It is quite intuitive that the mean utilization of a server by open-class customers is

$$U_{j,O} = \sum_{r=1}^{R_c} \sum_{k \in O} \lambda_{r,j,k} \times \nu_{r,j,k} \times \tau_{r,j,k} \quad \text{for } j = 1, \ldots, J. \tag{4.1}$$

The mean capacity of a server left for processing closed-class customers is $1 - U_{j,O}$. So, the closed-class customers are equivalently going through a closed QN with an inflated mean service time

$$\tau^*_{r,j,k} = \frac{\tau_{r,j,k}}{1 - U_{j,O}} \quad \text{for } r = 1,\ldots,R_c,\ j = 1,\ldots,J \text{ and } k \in C. \tag{4.2}$$

The mixed QN problem is now reduced to a closed QN problem.

Three key equations constitute the backbone of MVA for a closed QN, which relate among performance measures such as mean queue length, mean response time and throughput of each server.

$$R_{r,j,k}(\boldsymbol{n}) = \tau_{r,j,k} \times \left[1 + Q_j(\boldsymbol{n} - \boldsymbol{e}_r)\right] \quad \text{for } r = 1,\ldots,R_c,\ j = 1,\ldots,J,\ k \in C, \tag{4.3}$$

$$TX_r = \frac{n_r}{\sum_j \sum_k \nu_{r,j,k} \times R_{r,j,k}(\boldsymbol{n})}, \tag{4.4}$$

$$Q_j(\boldsymbol{n}) = \sum_{k \in C} \sum_{r=1}^{R_c} TX_r \times \nu_{r,j,k} \times R_{r,j,k}. \tag{4.5}$$

Equation (4.3) is obtained from the arrival instant distribution theorem [18] and Eqs. (4.4) and (4.5) are based on Little's Law of queueing theory.

Since the arrival rates of open-class customers should actually be equal to the throughputs of their respectively associated closed-class customers at the forking servers, the open-class arrival rates are updated for the next iteration by

$$\lambda_{r,j,k} = TX_r \times \nu_{r,j,k} \quad \text{for } r = 1,\ldots,R_c,\ j = 1,\ldots,J, \text{ and } k \in O. \tag{4.6}$$

The complete iterative algorithm is summarized as follows.

MVAF algorithm

Step 1. Initialize the mean queue length of servers
$Q_j(\mathbf{0}) = 0$, for $j = 1,\ldots,J$;

Step 2. Loop over number of customers

MAIN LOOPS

for $n_{R_c} = 0,\ldots,N_{R_c}$,

...

or $n_1 = 0,\ldots,N_1$;

Step 3. Initialize arrival rates of open-class customers
$\lambda_{r,j,k} = 0$, for $r = 1,\ldots,R_c$, $j = 1,\ldots,J$, and $k \in O$.

Step 4. Calculate the inflated mean service time of closed-class customers according to Eqs. (4.1) and (4.2)

Step 5. Calculate mean response times of closed-class customers by using Eq. (4.3)

Step 6. Calculate the mean throughput of each chain according to Eq. (4.4)

Step 7. Store the newly calculated open-class arrival rates
$L_{r,j,k} = TX_r \times \nu_{r,j,k}$; for $r = 1,\ldots,R_c$, $j = 1,\ldots,J$, and $k \in O$.

Step 8. Check the convergence
If $|L_{r,j,k} - \lambda_{r,j,k}| < \delta$, for $r = 1,\ldots,R_c$, $j = 1,\ldots,J$, and $k \in O$, then the algorithm converges for population distribution $\boldsymbol{n}$, go to Step 10.

Step 9. Update arrival rates of open-class customers
$\lambda_{r,j,k} = L_{r,j,k}$, for $r = 1,\ldots,R_c$, $j = 1,\ldots,J$, and $k \in O$.
Go to Step 4.

Step 10. Compute the queue length of each server
$Q_j(\mathbf{n}) = \sum_{k \in C} \sum_{r=1}^{R_c} TX_r \times \nu_{r,j,k} \times R_{r,j,k}$, for $j = 1,\ldots,J$.

End of MAIN LOOPS.

Step 11. Calculate server utilizations
$U_j = \sum_k \sum_{r=1}^{R_c} TX_r \times \nu_{r,j,k} \times \tau_{r,j,k}$, for $j = 1,\ldots,J$.

Step 12. OUTPUT and STOP.

Remark. The computational complexity of each iteration from Step 4 to Step 9 is dominated by $R_c \times J \times K$. Since the number of chains, the number of server nodes and the number of classes all grow proportionally to the number of processors in the system, i.e., N^2, $O(N^6)$ is a simple estimate of upper bound to the computational complexity of Steps 4–9.

5. Convergence analysis

The MVAF algorithm developed in the previous section is essentially a Jacobi type of fixed point iteration algorithm. Let mapping A_i represent the algorithm of step i, $i = 4,\ldots,9$ in the MVAF algorithm and define

$$A \equiv A_9 \circ A_8 \circ A_7 \circ A_6 \circ A_5 \circ A_4, \tag{5.1}$$

where $A_{i+1} \circ A_i(\cdot) \equiv A_{i+1}(A_i(\cdot))$ for $i = 4,\ldots,8$. The iterative procedure of MVAF under a given population distribution $\boldsymbol{n}$ of closed-class customers can be abstracted as

$$\boldsymbol{\lambda}^{i+1} = A(\boldsymbol{\lambda}^i), \quad i = 0, 1, 2,\ldots, \quad \text{with } \boldsymbol{\lambda}^0 = \mathbf{0}, \tag{5.2}$$

where $\boldsymbol{\lambda}$ is a vector consisting of all the open-class arrival rates $\lambda_{r,j,k}$, for $r = 1,\ldots,R_c$, $j = 1,\ldots,J$, and $k \in O$.

This section shows that MVAF converges to a unique fixed point by proving that A of (5.1) is a contraction mapping [4].

Definition. A mapping A: $\mathbb{R}^n \to \mathbb{R}^n$ is monotonically decreasing if $A(\boldsymbol{w}) < A(\boldsymbol{u})$ for any pair of $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{R}^n$ with $\boldsymbol{u} < \boldsymbol{w}$.

A sufficient condition for a continuously differentiable mapping to be monotonically decreasing is given in Lemma 1.

Lemma 1. *Let* $[a, b] \in \mathbb{R}$, $a < b$, $D = [a, b]^n$, $x_i \in D$ *for* $i = 1,\ldots,n$, *and* $X = [x_1,\ldots,x_n]$. *Also let* f_i:$D \to [a, b]$ *for* $i = 1,\ldots,n$, $f_i \in C^1$ *and a mapping* A:$D \to D$, *where* $A(X) = [f_1(X),\ldots,f_n(X)]$. *If* $\partial f_j(X)/\partial x_i < 0$ *for* $1 \le i,j \le n$ *and* $\forall X \in D$, *then* A *is monotonically decreasing.*

The proof of Lemma 1 is described in the Appendix.

The following theorem then gives a set of conditions under which a continuous and monotonically decreasing mapping is a contraction mapping with a unique fixed point.

Theorem 1. *Let $D \subset \mathbb{R}^n$ be a closed set. A mapping $A: D \to D$ is monotonically decreasing and continuous. Suppose $A(\mathbf{0}) = \mathbf{z}$ and $\mathbf{0} < A(\mathbf{z}) < \mathbf{z}$. Let $H = \{\mathbf{x} \mid \mathbf{0} \le \mathbf{x} \le \mathbf{z}\} \subset D$, and $\mathbf{x}^{i+1} = A(\mathbf{x}^i)$ with $\mathbf{x}^0 = \mathbf{0}$, $i = 0, \ldots, \infty$. Then A is a contraction mapping and the sequence $\{\mathbf{x}^i\}$ converges to a unique $\mathbf{x}^* \in H$.*

Interested readers may refer to the Appendix for the details of the proof.

Finally, Theorem 2 concludes the convergence of algorithm A.

Theorem 2 (Convergence of the MVAF algorithm). *Algorithm A as defined in* (5.1) *for one iteration of MVAF under a given closed-class population distribution $\boldsymbol{n}$ is a continuously differentiable contraction mapping that $\boldsymbol{\lambda}^{i+1} = A(\boldsymbol{\lambda}^i)$, $i = 0, 1, 2, \ldots, n, \ldots$, with $\boldsymbol{\lambda}^0 = \mathbf{0}$, converges to a unique fixed point $\boldsymbol{\lambda}^* \in [0, \boldsymbol{\lambda}_{\max}]$, where $\boldsymbol{\lambda}_{\max}$ is the vector of maximum arrival rates of open-class customers.*

Proof. The mapping $A = A_9 \circ A_8 \circ A_7 \circ A_6 \circ A_5 \circ A_4$ is continuously differentiable due to the facts that $A_j \in C^1$, $j = 4, \ldots, 9$ and that a composite mapping of mappings in C^1 is also in C^1. Let $\lambda^i_{r,j,k}$ be a component of $\boldsymbol{\lambda}^i$. Then

$$\frac{\mathrm{d}\boldsymbol{\lambda}^{i+1}}{\mathrm{d}\boldsymbol{\lambda}^i} = \left[\frac{\mathrm{d}\lambda^{i+1}_{r,j,k}}{\mathrm{d}\lambda^i_{r',j',k'}}\right],$$

where $r, r' \in \{1, \ldots, R_c\}$, $j, j' \in \{1, \ldots, J\}$, $k, k' \in O$. By chain rule, we have

$$\begin{aligned}
&\frac{\mathrm{d}\lambda^{i+1}_{r,j,k}}{\mathrm{d}\lambda^i_{r',j',k'}} \\
&= \frac{\mathrm{d}A_9}{\mathrm{d}A_8} \times \frac{\mathrm{d}A_8}{\mathrm{d}A_7} \times \frac{\mathrm{d}A_7}{\mathrm{d}A_6} \times \frac{\mathrm{d}A_6}{\mathrm{d}A_5} \times \frac{\mathrm{d}A_5}{\mathrm{d}A_4} \times \frac{\mathrm{d}A_4}{\mathrm{d}\lambda^i_{r',j',k'}} \\
&= -\nu_{r,j,kO} \times \frac{\sum_{k \in C} n_r \times \nu_{r,j,k}}{\left(\sum_j \sum_{k \in C} \nu_{r,j,k} \times R_{r,j,k}\right)^2} \times \left[1 + Q_j(\boldsymbol{n} - \boldsymbol{e}_r)\right] \times \frac{\tau_{r,j,k} \times \tau_{r',j',kO}}{(1 - U_{j,O})^2} \\
&< 0, \quad \text{for } kO \in O \text{ and } k \in C.
\end{aligned}$$

So, A is monotonically decreasing in $[0, \boldsymbol{\lambda}_{\max}]$ according to Lemma 1.

Now let $\boldsymbol{\lambda}^0 = \mathbf{0}$ (no open-class customers arriving) and $\boldsymbol{\lambda}^1 = \boldsymbol{\lambda}_{\max} = A(\mathbf{0})$. Note that upper bounds for throughputs of closed-class customers at each server under a given population distribution $\boldsymbol{n}$ can be obtained by assuming no open-class customers in the queueing network, which in turn yields upper bounds for $\boldsymbol{\lambda}$ since open-class customer arrival rates should actually be equal to their associated closed-class customer throughputs. Let $H \equiv [0, \boldsymbol{\lambda}_{\max}]$. As $\boldsymbol{\lambda}^1 = \boldsymbol{\lambda}_{\max}$,

A is monotonically decreasing in $[0, \boldsymbol{\lambda}_{\max}]$, and $\boldsymbol{\lambda}^2 = A(\boldsymbol{\lambda}) < \boldsymbol{\lambda}_{\max} = \boldsymbol{\lambda}^1$, it is clear that A satisfies all the conditions in Theorem 1, is a contraction mapping and $\{\boldsymbol{\lambda}^i\}$ converges to a unique fixed point $\boldsymbol{\lambda}^* \in H$. □

6. Performance evaluation

6.1. Validation of MVAF

To validate the MVAF algorithm for performance evaluation of the QN model developed in Section 3, a simulation environment is developed for the QN model using the Q^+ [16] simulation package. In the simulation environment, an $N \times N$ system is assumed where a modified data block requested by one processor due to a cache miss is possessed by one of the other $(N^2 - 1)$ processors with a uniform probability distribution and so is a shared block by one of the N main memory modules. It is also assumed that row bus and column bus speeds are equal, the time unit is one bus cycle, and bus access times are deterministic depending on whether the access is an address request or a cache block transfer. Although the bus access time of a cache block is determined by both the block size and bus bandwidth, these two parameters are reduced to a single parameter, the cache block transfer time in our model.

Key input parameters of our simulation model are listed as follows:

t_p = mean processing time at a processor,
p_s (p_x) = probability that a cache miss request is for a shared (modified) data block,
$P_{read\text{-}mod}$ (P_{read}) = probability that the cache miss is a write (read) brequest,
$t_{A,r}$ ($t_{A,c}$) = time for transmitting the block address on a row (column) bus,
$t_{D,r}$ ($t_{D,c}$) = time for transmitting the data on a row (column) bus,
$t_{WB,c}$ = time for transmitting a data block to the main memory module through a column bus,
$t_{I,r}$ = time for transmitting an invalidation signal on a row bus,
d_{mem} = main memory latency,
d_{cache} = cache memory latency,
N = number of processors per row (or column).

Baseline parameter values used in our simulation are $t_{A,r} = t_{A,c} = 2$ cycles, $t_{I,r} = 1$ cycle, $t_{D,r} = t_{D,c}$ = the number of cycles of the quoted cache block size, $t_{WB,c} = t_{D,c}$, and $d_{cache} = d_{mem} = 15$ bus cycles. Given a cache miss rate, a memory reference frequency and a bus cycle time, the average processing time of a processor server t_p can then be calculated as

$$t_p = 1/(\text{bus cycle time} \times \text{memory reference frequency} \times \text{cache miss rate}). \tag{6.1}$$

Mean processor and bus utilizations are the performance measures considered, since the design of a parallel computer is desirable to fully exploit the power of its processors. It is assumed that each processor is constantly busy processing tasks except when waiting for data due to memory latency or communication. The mean processor utilization U_p is defined as

$$U_p = \frac{\text{mean processing time}}{\text{mean waiting time} + \text{mean processing time}}$$

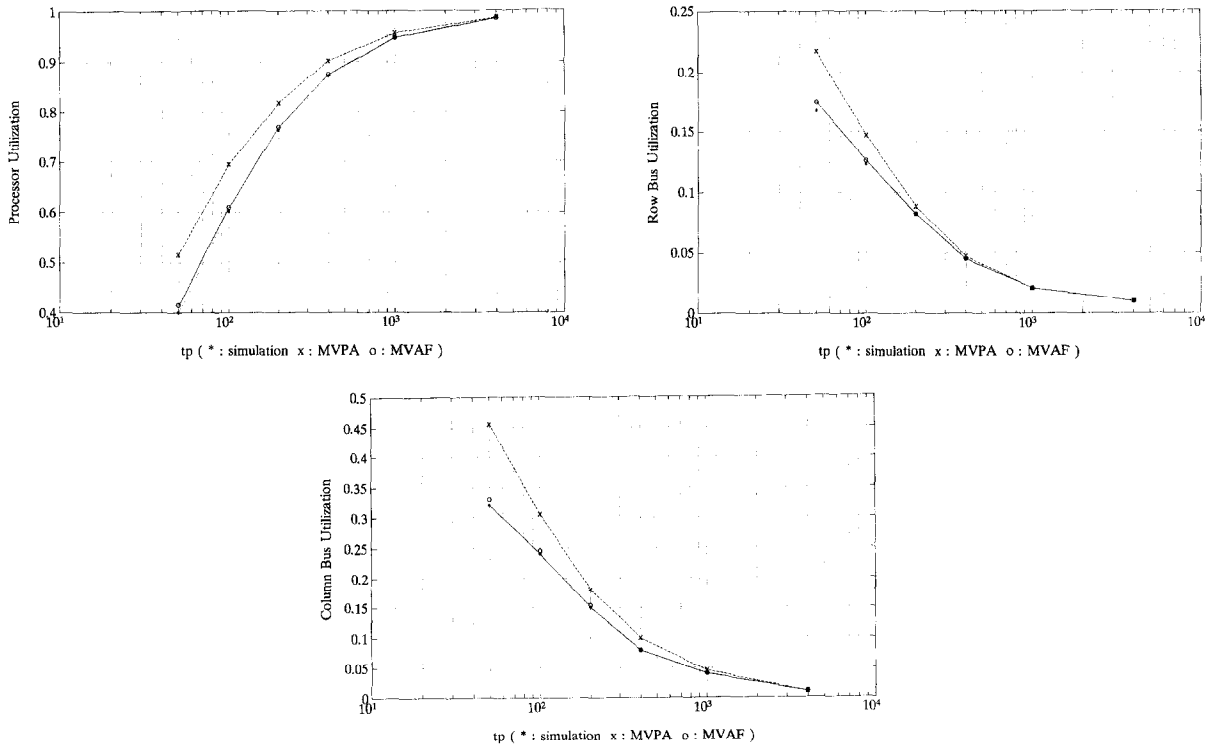


Fig. 4. (a) Mean processor utilization versus mean processing time; (b) Row bus utilization versus mean processing time; (c) Column bus utilization versus mean processing time for $N = 2$ and cache block size = 16 under even load distribution.

Similarly the bus utilization can be defined as

$$U_b = \frac{\text{mean data transmission time}}{\text{mean data transmission time} + \text{mean bus holding time}}$$

where b is a bus index and a bus is held during a memory access.

The validation is performed on a 2×2 and a 3×3 2-D Multicube system models with the baseline parameters. Both the Q^+ simulation and the MVAF algorithm are run on a SUN/Sparc-IPC workstation. It takes in average 2–3 hours of CPU time for one simulation run and less than 0.1 s for getting a MVAF solution.

6.1.1. Homogeneous workload scenario

In this scenario, routing probabilities and the mean processing time distributions are the same among processor modules. Figures 4(a)–(c) give the processor utilization, row and column bus utilization versus the mean processing time t_p in a 2×2 Multicube respectively. The variation of t_p reflects the variation of bus communication load; the shorter the t_p, the higher the load. Figures 5(a)–(c) are for a 3×3 Multicube. Very good agreement between the MVAF and simulation results are observed from all these figures, where the maximum relative difference that happens in the high load range (short t_p) is less than 5% for the 2×2 case, and

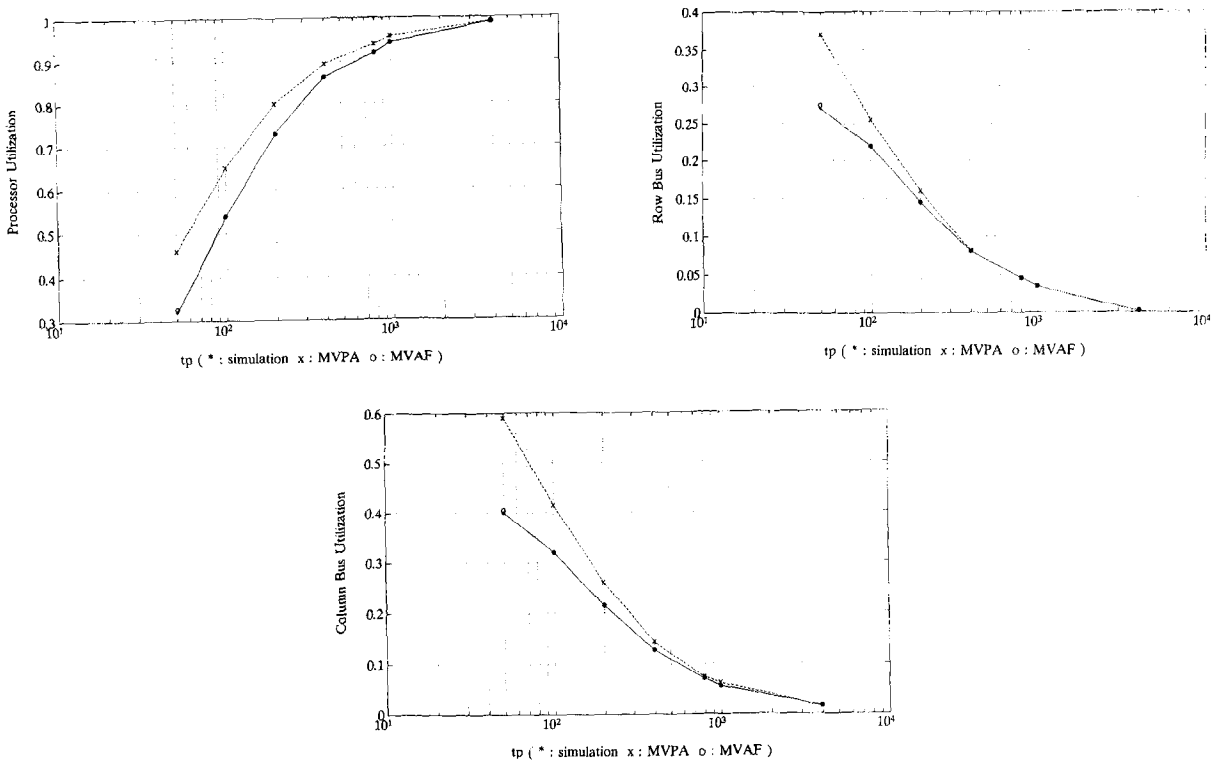


Fig. 5. (a) Mean processor utilization versus mean processing time; (b) Row bus utilization versus mean processing time; (c) Column bus utilization versus mean processing time for $N = 3$ and cache block size = 64 under even load distribution.

there is almost no difference between the two for the 3×3 case. Intuitively, blocking occurs more significantly at short t_p, i.e., high communication load. We therefore conjecture that the minor overestimate of utilization at the high load range is caused by the infinite buffer (or non-blocking) assumption of MVAF.

Performance measures obtained by using the Mean Value Performance Analysis (MVPA) algorithm of [19] are also plotted in Figs. 4 and 5. It is obvious that the MVPA overestimates the utilizations; the shorter the t_p, the higher the overestimation; the overestimation is more severe in the 3×3 case than in the 2×2 case. The cause of such difference will be discussed in the next subsection.

6.1.2. *Heterogeneous workload scenario*

Inter-cache miss time t_p for processor p11 in the 2×2 2-D multicube system is set 40% shorter than that of the other three processors, i.e., p11 has a higher cache miss rate. Figures 6(a)–(c) again demonstrate excellent agreements between results of the MVAF algorithm and the simulation. The minor differences also occur at the high load range as in (a). The MVPA

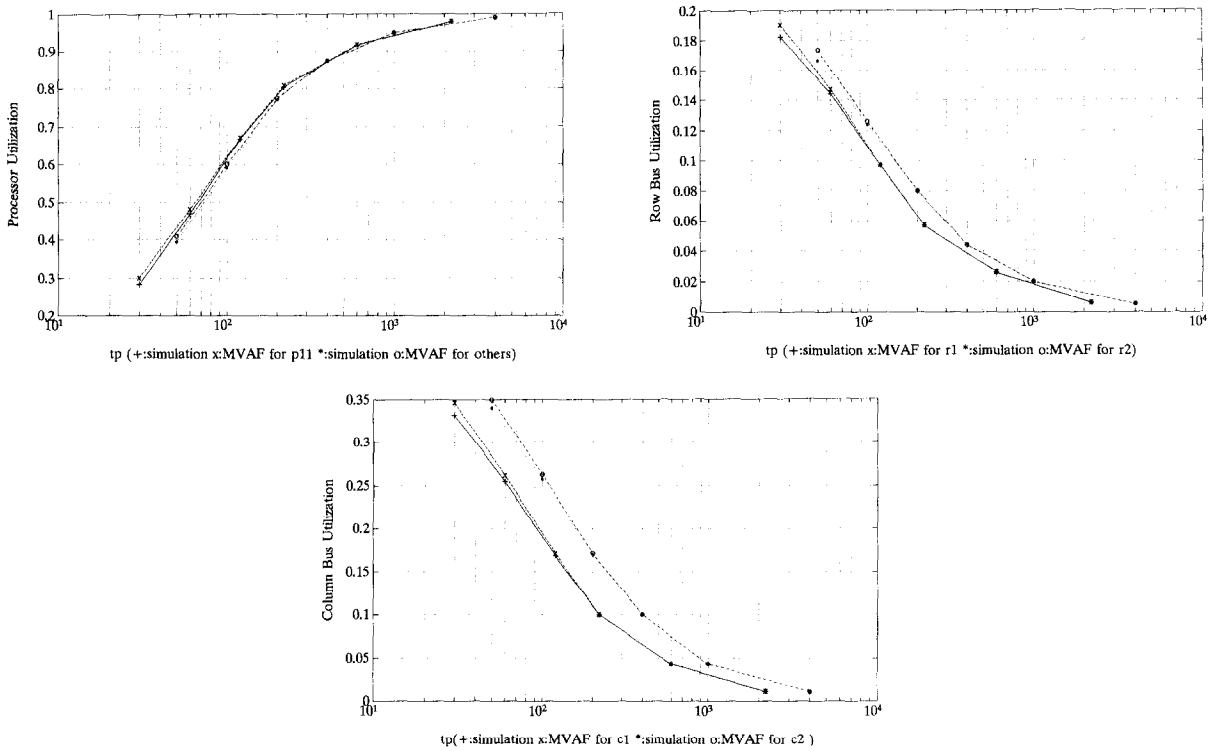


Fig. 6. (a) Mean processor utilization versus mean processing time; (b) Row bus utilization versus mean processing time; (c) Column bus utilization versus mean processing time for $N = 2$ and cache block size = 16 under uneven load distribution.

algorithm cannot be applied to cases of heterogeneous load and is therefore not compared with.

Simulations of larger scale systems would provide further validations of the MVAF algorithm. However, the construction of routing path and transaction paths and simulating a large-scale system are very time consuming. As we have shown the method of doing so and the excellent validation results for small-scale systems, we claim that the validity of the MVAF algorithm for evaluating the performance of the QN model of a 2-D Multicube system has been preliminary justified.

6.2. Performance evaluation by MVAF

The MVAF algorithm is now applied to evaluate the performance of large-scale 2-D Multicube system. The same set of problems as those of [14] are studied, with parameter values adopted in Subsection 6.1 as the baseline parameters. Study issues include the effect of system scale, cache miss rate and cache block size on processor and bus utilizations. Results are then compared with those obtained by the MVPA algorithm in [14].

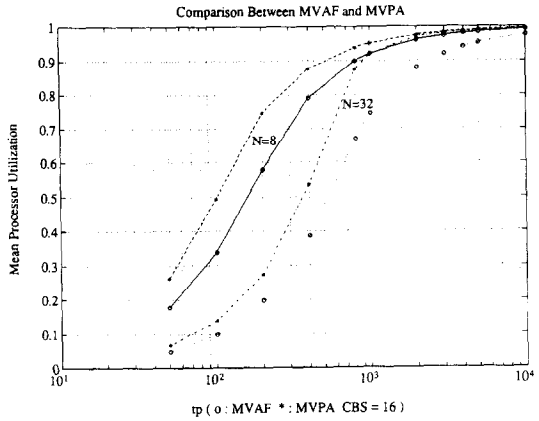


Fig. 7. Mean processor utilization for cache block size = 16.

6.2.1. Processor utilization

Figures 7 and 8 illustrate how the mean processor utilization is varied with respect to system size (N) for the cases of data block size 16 and 64 respectively. Mean processor utilization decreases as N increases because for the 2-D Multicube architecture, the number of buses increases linearly with N while the number of processor modules and the total system workload increase quadratically. The load per processor under which an 8×8 Multicube can sustain a utilization over 90% is about two times of that for a 32×32 system and utilization degradation due to scale increase turns out to be graceful. Such an observation holds for both MVAF and MVPA results. Although MVAF and MVPA show similar trends in their results, MVPA generates much more optimistic utilizations that does MVAF (may be up to 30% difference). The difference reduces as data block size grows larger.

Figure 9 shows that a smaller block size gives a more graceful degradation in mean processor utilization. Note that in the one hand, the larger the cache block size, the more communication

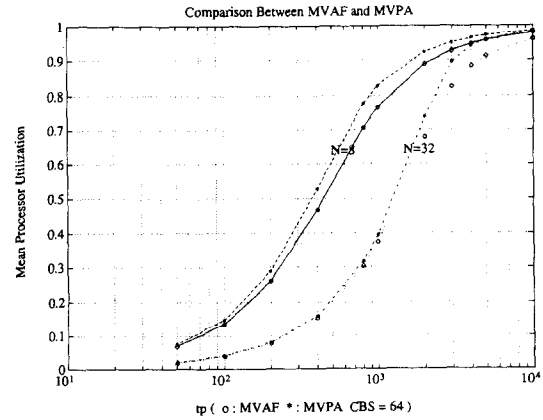


Fig. 8. Mean processor utilization for cache block size = 64.

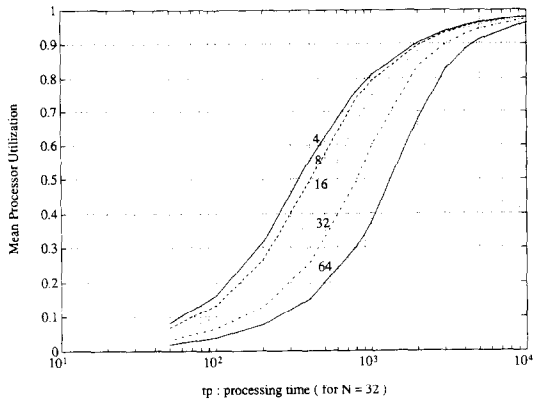


Fig. 9. The effect of block size for a 32×32 2-D Multicube.

time needed in one transmission and the higher probabilities of bus contention for a given cache miss rate; on the other hand, due to the property of the locality of reference [11], the cache miss rate may increase if the block size is smaller. There is therefore a trade-off to determine the optimal block size for the system. Comparing the processor utilization curves of the same system size in Figs. 7 and 8, we also observe that the trend of degradation predicted by MVPA is more sensitive to the block size than that predicted by MVAF.

6.2.2. Bus utilizations

Bus utilizations with respect to different performance factors are expected to have a reverse trend as compared to that of processor utilization because the utilization of buses intuitively implies the idleness of some processors. Figures 10(a)–(d) indeed confirm this expectation. In a 2-D Multicube system with a homogeneous workload, the cache coherent protocol basically distribute the bandwidth demands equally to row and column buses with the exceptions of invalidation and write back transactions. The amount of invalidation traffic on the row buses versus write-back traffic on the column buses determines which type of buses is more heavily utilized.

Figures 10(a)–(d) give the row and column bus utilizations for systems with block sizes of 16 and 64, respectively. Figures 10(a) and (b) show that for a cache block size of 16, the row bus is much more utilized than the column bus when cache miss rate is above one per 1000 cycles. Figure 10(c) and (d) show that for a block size of 64, the column bus is now more heavily utilized than the row bus. This phenomenon can be explained by the fact that although an invalidation request generates concurrent traffic to all row buses, it requires a shorter time on each row bus than transmitting a data block on a column bus (i.e., $t_{I,r} < t_{WB,c}$). When the block size is large enough, the utilization of one column bus due to a write-back request becomes higher than the total row bus utilization caused by an invalidation signal.

Both MVAF and MVPA results reflect the above features, but there are significant differences (up to 40% in Fig. 10(b)) of predicted utilizations in the high load range. Such differences are mainly due to the difference in modeling. In MVPA, a bus is not concurrently held when accessing a memory or a cache module and may serve other requests. In MVAF, a

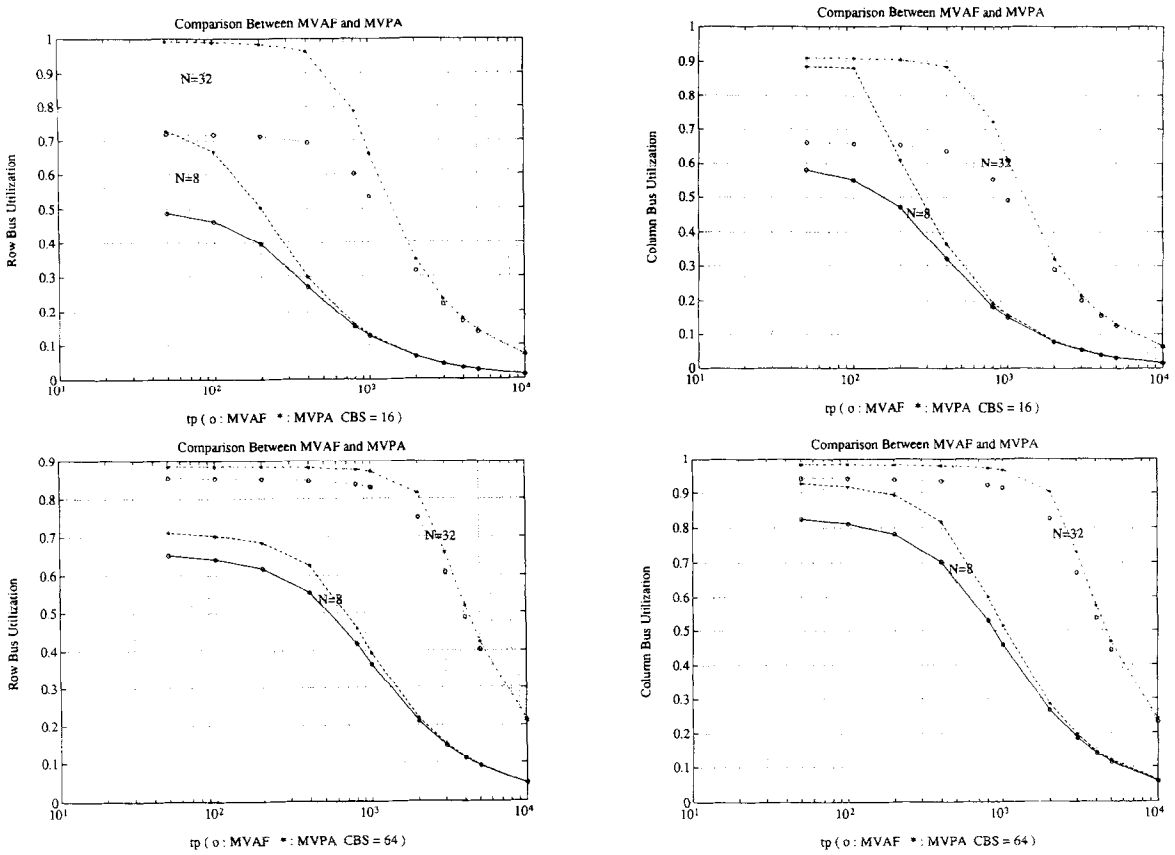


Fig. 10. (a) Row bus utilization for cache block size = 16; (b) Column bus utilization for cache block size = 16; (c) Row bus utilization for cache block size = 64; (d) Column bus utilization for cache block size = 64.

bus is tied up (but considered idle) till the completion of a memory or a cache access. The former therefore has a higher utilization than the latter, especially in the high load range.

7. Conclusions

In this paper we conducted the performance evaluation of a 2-D Multicube, cache coherent multiprocessor architecture. A multiclass, multichain queueing network (QN) model was first constructed for the multiprocessor, which had salient features of blocking and forking. Simulation results showed that the blocking effect was relatively minor and was neglected in our later study. To handle a QN with forking phenomena, we developed an iterative mean value analysis algorithm (MVAF), which first approximates a forking by the generation of an open-class customer and a closed-class customer and the Mean Value Analysis (MVA) is applied to the resultant mixed QN. The MVAF then iteratively applies both the approximation and the MVA till convergence. We proved that the MVAF is a contraction mapping and converges to the solution of the original QN with forking. Comparing the performance evaluation results by

MVAF for small-scale multicube multiprocessors to those by simulation, we observed less than 5% of differences in average processor and bus utilizations and the larger the system, the smaller the differences. We also compared MVAF with the mean value performance analysis (MVPA) on large-scale 2-D Multicube systems. Results indicated that the two approaches required the same order of computation times, generated performance measures of similar trends, but the difference in magnitude went up to 40%. Though our approach required some knowledge about QN as compared to MVPA, we believe that it is not only superior for evaluating the 2-D Multicube multiprocessor, but also has a good potential for further extensions and applications.

Acknowledgement

This work was supported in part by the National Science Council of Republic of China under Grant NSC-80-0404-E-002-12 and by the Electronics Research & Service Organization of Industrial Technology Research Institute.

Appendix

Proof of Lemma 1. Define $\delta x_i \equiv [0,\ldots,\delta x_i,\ldots,0]$ and $\delta X \equiv [\delta x_1,\ldots,\delta x_n]$, where $\delta x_i > 0$ for $i = 1,\ldots,n$. Since $\partial f_j(X)/\partial x_i < 0$ for all $X \in D$,

$$f_j(X+\delta x_i) < f_j(X) \quad \text{for all } X+\delta x_i \text{ and } X \in D, \tag{A.1}$$

which implies that all the f_js are monotonically decreasing in each coordinate.

Let $X_0 = X$ and $X_k = X + \sum_{i=1}^{k} \delta x_i \in D$, for $k = 1,\ldots,n$. As a result of (A.1),

$$f_j(X_k) > f_j(X_{k+1}) \quad \text{for } k = 0,\ldots,n-1,$$

i.e.,

$$f_j(X) > f_j(X+\delta X) \quad \text{for any } \delta X > 0 \text{ such that } X+\delta X \in D.$$

So, $f_j(X)$ is monotonically decreasing, $\forall j$.

It then easily follows that

$$[f_1(X),\ldots,f_n(X)] > [f_1(X+\delta X),\ldots,f_n(X+\delta X)]$$

or

$$A(X) > A(X+\delta X) \quad \text{for all } \delta X > 0 \text{ and } X+\delta X \in D,$$

So $A(X)$ is monotonically decreasing in D. □

Proof of Theorem 1. We need only show that the mapping A strictly contracts in the closed set H, i.e.,

$$\| A(x^{i+1}) - A(x^i) \| < \| x^{i+1} - x^i \|, \quad i \geq 0. \tag{A.2}$$

The existence and uniqueness of the fixed point then follows from Proposition 1.1 of [4, pp. 182–183]. We now show that (A.2) holds by mathematical induction:

(1) $i = 0$. Since $x^0 = \mathbf{0} \in H$, $x^1 = z \in H$, $\mathbf{0} < A(z) < z$ and $x^2 = A(z) \in H$ by assumption,

$$\| A(x^1) - A(x^0) \| = \| A(z) - z \| < \| z - \mathbf{0} \| = \| x^1 - x^0 \|.$$

(2) $i = 1$. As $\mathbf{0} = x^0 < x^2 < x^1$ and A is monotonically decreasing,

$$A(x^1) = x^2 < A(x^2) = x^3 < A(x^0) = x^1 = z,$$

So, $x^3 \in H$ and $\| A(x^2) - A(x^1) \| = \| x^3 - x^2 \| < \| x^2 - x^1 \|$.

(3) Assume that (A.2) holds for $1 \le i \le n$,

$$\begin{aligned} &\mathbf{0} < x^{i-1} \le x^i < x^{i-2} < z \quad \text{if } i \text{ is odd,} \\ &\mathbf{0} < x^{i-2} \le x^i < x^{i-1} < z \quad \text{if } i \text{ is even.} \end{aligned} \tag{A.3}$$

(4) $i = n + 1$. From (A.3) and the monotonically decreasing property of A,

$$\begin{aligned} &\mathbf{0} < A(x^{n-2}) = x^{n-1} < A(x^n) = x^{n+1} < A(x^{n-1}) = x^n < z \quad \text{if } n \text{ is odd,} \\ &\mathbf{0} < A(x^{n-1}) = x^n < A(x^n) = x^{n+1} < A(x^{n-2}) = x^{n-1} < z \quad \text{if } n \text{ is even,} \end{aligned}$$

i.e.,

$$\mathbf{0} < x^{(n+1)-1} < x^{n+1} < x^{(n+1)-2} < z \quad \text{if } n+1 \text{ is odd,} \tag{A.4a}$$

$$\mathbf{0} < x^{(n+1)-2} < x^{n+1} < x^{(n+1)-1} < z \quad \text{if } n+1 \text{ is even.} \tag{A.4b}$$

For both (A.4a) and (A.4b), we have $x^{n+1} \in H$ and

$$\| A(x^n) - A(x^{n-1}) \| = \| x^{n+1} - x^n \| < \| x^n - x^{n-1} \|.$$

By mathematical induction, we conclude from (1) through (4), that (A.2) holds for all $k \ge 1$, strictly contradicts on H and has a unique fixed point.

References

[1] M. Ajmone Marsan, G. Balbo and G. Conte, *Performance Models of Multiprocessor Systems*, MIT Press, Cambridge, Mass. (1986).

[2] I.F. Akyildiz, Mean value analysis for blocking queueing networks, *IEEE Trans. Softw. Eng.* **14** (4) (1988) 418–427.

[3] J. Archibald and J.L. Baer, Cache coherence protocols: evaluation using a multiprocessor simulation model, *ACM Trans. Comput. Systems* **4** (1986).

[4] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation*, Prentice-Hall, Englewood Cliffs, N.J. (1989).

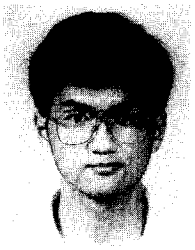
[5] K.M. Chandy, J.H. Howard Jr. and D.F. Towsley, Product form and local balance in queueing networks, *J. ACM* **24** (1977) 250–263.

[6] D.S. Chang and C.S. Tsou, An approximate algorithm for closed queueing networks of flexible manufacturing system, *Proc. 1st Int. Conf. on Automation Technology*, Taipei, July 1990, pp. 153–164.

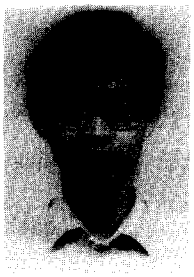
[7] J.R. Goodman, Coherency for multiprocessor virtual address caches, *Proc. 2nd Int. Conf. on Architectural Support for Programming Language and Operation Systems (ASPLOSII)*, October 1987, pp. 72–81.

[8] J.R. Goodman and P.J. Woest, The Wisconsin multicube: a new large-scale cache coherent multiprocessor, *Proc. 15th Int. Conf. Computer Architecture*, 1988, pp. 422–431.

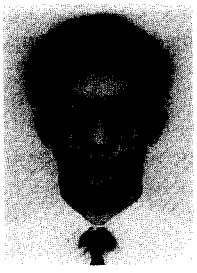
[9] P. Heidelberg and K.S. Trivedi, Queueing network models for parallel processing with asynchronous tasks, *IEEE Trans. Comput.* **31** (11) (1982) 1099–1109.
[10] M.A. Holiday and M.K. Vernon, Performance estimates for multiprocessor memory and bus interference, *IEEE Trans. Comput.* **36** (1987) 76–85.
[11] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York (1984).
[12] S.S. Lavenberg (Ed.), *Computer Performance Modeling Handbook*, Academic Press, Orlando, Fla. (1983).
[13] E.D. Lazowska et al., *Quantitative System Performance — Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Englewood Cliffs, N.J. (1984).
[14] S.T. Leutenegger and M.K. Vernon, A mean value performance analysis of a new multiprocessor architecture, *Proc. Conf. on Measurement and Modeling of Computer Systems, ACM SIGMETRICS* May 1988, pp. 167–176.
[15] K.R. Pattipati, M.M. Kostreva and J.L. Teele, Approximate mean value analysis algorithms for queueing networks: existence, uniqueness, and convergence results, *J. ACM* **37** (3) (1990) 643–673.
[16] The Q+ User's Manuals, AT & T Bell Labs., Homdel, N.J.
[17] M. Reiser and S.S. Lavenberg, Mean-value analysis of closed multichain queueing networks, *J. ACM* **27** (1980) 313–322.
[18] K.C. Sevick and I. Mitrani, The distribution of queueing network states at input and output instants, *J. ACM* **28** (2) (1981) 358–371.
[19] M.K. Vernon, R. Jog and G.S. Sohi, Performance analysis of hierarchical cache-coherent multiprocessor, *Perform. Eval.* **9** (1989) 287–302.
[20] Q. Yang, L. Bhuyan and B.C. Liu, Analysis and comparison of cache coherence protocols for a packet-switched multiprocessor, *IEEE Trans. Comput.* **38** (8) (1989) 1143–1153.



Chien-Yuan Huang received the B.S. degree in mechanical engineering in 1989 and the M.S. degree in electrical engineering in 1991 both from National Taiwan University.Since 1993, he has been working towards the Ph.D. degree in the Department at Electrical Engineering at the Pennsylvania State University. His major areas of interest are computer communication networks, parallel systems, and computer vision.



Shi-Chung Chang received his B.S.E.E. degree from National Taiwan University, Taiwan, Republic of China, in 1979, and his M.S. and Ph.D. degrees in electrical and systems engineering from the University of Connecticut, Storrs, in 1983 and 1986 respectively.
From 1979 to 1981 he served as an Ensign in the Chinese Navy, Taiwan. He worked as a technical intern at the Pacific Gas and Electric Co., San Francisco, in the summer of 1985. During 1987, he was a member of the Technical Staff, decision systems section, ALPHATECH, Inc., Burlington, MA. He is currently with the Electrical Engineering Department of National Taiwan University. His research interests include optimization theory and algorithms, operation scheduling and control of large-scale systems, parallel computing, high speed networks and distributed decision making.
Dr. Chang is a member of Eta Kappa Nu and Phi Kappa Phi.



Chern-Lin Chen was born in 1962 in Taipei, Taiwan. He received his B.S. and Ph.D. degrees in Electrical Engineering from the National Taiwan University in 1984 and 1987, respectively. Since then, he has been with the Department of Electrical Engineering at this university.His current research interests lie in the areas of analysis, design and application of power electronics converters and computer applications in power system engineering.