A Memory Interference Model for Regularly Patterned Multiple Stream Vector Accesses

Qing Yang and Tao Yang

ABSTRACT

Most existing analytical models for memory interference generally assume random bank selection for each memory access. In vector computers, however, memory accesses are typically regularly patterned with a number of data items being accessed concurrently from different banks. Very little is known about the queueing behavior of memory interferences in multiple stream vector accesses. This paper presents an analytical model for memory interferences due to vector accesses in multiple vector processor systems. The model captures the effects of both bank conflicts among elements within one vector access stream and conflicts among multiple vector access streams on system performance. The model is based on a closed queueing network assuming an ideal interconnection network. An approximation technique is proposed to solve the memory queueing system that serves customers in a complicated way (non-FIFO). We also carry out extensive simulation experiments to study memory interference and validate our analytical model. Simulation results and analytical results are in a very good agreement, indicating that the model is very accurate. We further validate our analysis by comparing the numerical results obtained from our analytical model with those measurement results that were published by other researchers. Based on our analytical model and simulations, we carry out performance evaluation of the multiple vector processor systems. Our numerical results show that memory access conflicts pose a severe limitation on the number of useful processors in the system, implying that memory system design is essential to high performance computing.

1 Introduction

Multiple vector processor systems with common shared memory are the architecture of choice of today's most prominent supercomputers. Many such computers are already commercially available such as Cray computers, NEC SX-3, Convex (C-2), and Alliant FX/8, to mention only a few. Such vector computer systems require high memory bandwidth that is traditionally achieved by interleaved memory consisting of many independent memory modules commonly called banks. Typically, each vector access stream consists of a sequence of *element requests* that may go to a number of memory banks that can be accessed parallelly. Addresses associated with consecutive element requests of a vector access differ by a constant number which is known as *stride* of the vector access. Memory interference can occur in two different forms: *self-interference*, interference between element requests of the same vector access, and *cross-interference*, interference between different vector accesses. A key issue in such a system is the extent to which contention for shared memory banks causes performance degradation. A number of performance studies for memory interference in vector computers has been reported in the literature [1, 2, 3, 4, 5, 6, 7]. A good summary of the existing analytical models can be found in [3]. Until now, however, no one has derived an analytical memory interference model that contains all of the following four important and realistic properties:

1. all vector accesses are regularly patterned as they are in practice,
2. both self-interference and cross-interference due to multiple vector access streams are considered,
3. unsatisfied memory requests are buffered or queued at memory banks,
4. all possible values of the stride of vector access are considered,

We present here an analytical model that contains all these four properties for multiple vector processor systems. The model is based on a closed chain queueing network consisting of processors and a memory system queue assuming an ideal interconnection network. Customers or jobs that circulate in the queueing network are *vector access requests* each of which consists of a fixed number of element requests. When a vector access request that is generated from a vector processor joins the memory queue, the element requests of the vector access are placed in memory banks according to the access stride and the starting element address. An element request may see in the addressed memory bank other element requests from the same vector access (*self-interference*) or element requests from other vector accesses (*cross-interference*). Each memory bank serves element requests in the buffer on first-in-first-out (FIFO) basis. A vector access is said to be complete only when all

the element requests belonging to that vector have finished their memory accesses. As a result, if one considers each vector access request as a queue customer and looks at the memory system queue as a whole, the queueing discipline is not necessarily first-in-first-out any more because some banks may be more heavily loaded than others. This phenomenon makes the analytical model nontrivial. We have devised an approximation technique to handle this kind of irregular queueing discipline assuming infinite buffer size at the memory queue.

It is known that both self-interference and cross-interference have a lot to do with the vector access stride. While the unit stride occurs very frequently in applications, many vector accesses have nonunit stride [8, 9]. One example is accessing a row vector of a matrix stored in column-major. In this case, the stride is equal to the column length which is dependent on the problem size that varies widely. In our analysis, we assume that the stride of accessing a vector is constant throughout the vector access. This constant is chosen, for each vector access, from a random number with a uniform distribution between 1 and the number of memory banks except for a certain specified large probability of stride being 1. With this assumption, we precisely model the average number of bank conflicts and are able to give an efficient Mean Value Analysis (MVA) algorithm to solve the queueing model. The analytical model is validated through extensive simulation experiments to show they are in a very good agreement. We also validate our analysis by comparing the numerical results of our analytical model with the numerical results of previously published measurements on real machines [7]. It is shown that our analytical model can give a quick and fairly accurate performance estimate of multiple vector processor systems.

Using our analytical model, we carry out quantitative evaluation of the memory performance of multiple vector processor systems. It is observed that the performance improvement by adding more vector processors is not linearly related to the number of processors in the system due to memory contention. Our numerical results show that memory access conflicts pose a severe limitation on the number of useful processors in the system as concluded previously by other investigators [7, 3]. Therefore, memory system design is essential to high performance computing.

The paper is organized as follows. Section 2 presents problem formulation and assumptions used in our analysis. The analytical performance model for the multiple vector processor systems is presented in Section 3. We carry out simulation study and validate our analytical model in Section 4. Comparisons between our analysis and measurements are also carried out in this section. In Section 5, we present performance evaluations for the multiple vector processor system using the analytical model developed. Section 6 concludes the paper.

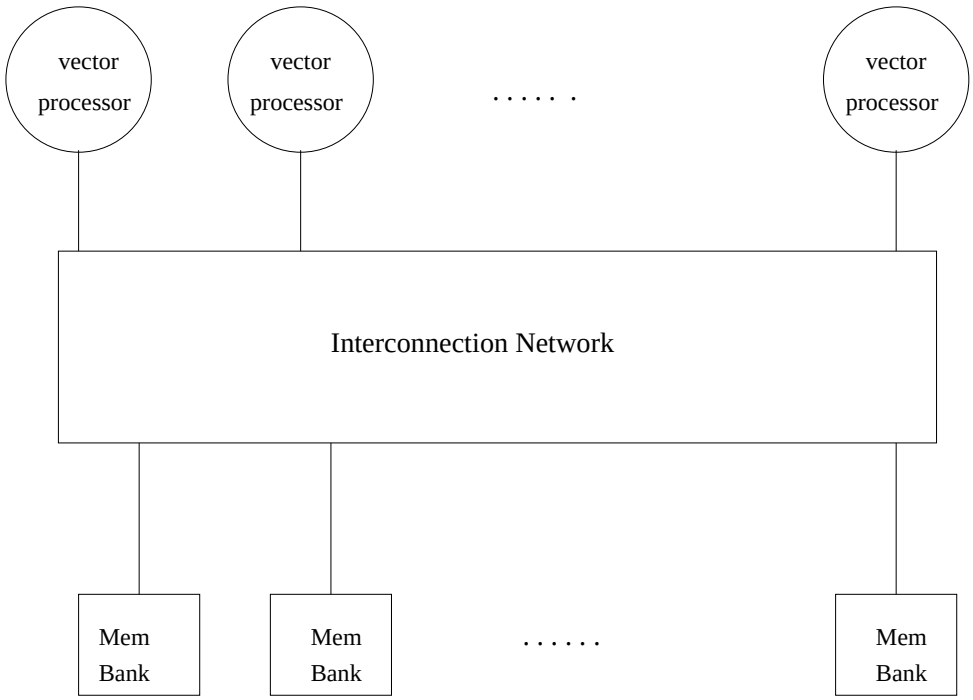

Figure 1: Schematic diagram for a multiple vector processor system.

2 Problem Formulation and Assumptions

Figure 1 shows the schematic diagram of the system considered in this paper. There are N_P memory access ports each of which attempts to access the memory system at regular intervals. For simplicity, we assume that each memory port corresponds to a vector processor. A vector access request generated from a port consists of a fixed number of element (word) requests. This fixed number of element requests is known as maximum vector register length of a given vector computer, denoted by V_L. Each vector access is associated with an access stride which is the address distance in terms of bank number between two consecutive vector elements. After a processor issues a vector access request, it waits in an idle state until all the V_L vector elements are successfully loaded into a vector register. For the purpose of simplicity, we make the following assumptions in our analysis.

1. The memory system has $M = 2^m$ low-order bit interleaved memory banks with a bank cycle time fixed at t_m. At any given time, only one memory access can be in progress in a bank. A vector element with address $addr$ is located in memory bank number $addr \mod M$.

2. There are N_P independent processors (memory ports) that generate vector access requests to the memory system. The workload is characterized by the interval between the completion of a vector access and the generation of the next vector access request. This interval is called think time as the processor is busy doing computation. The think times of the processors are

identical, independent and generally distributed random variables with mean Z.

3. The interconnection network between the processors and the memory banks is assumed to have sufficient bandwidth.

4. The length of a vector to be accessed each time from the memory system is fixed at V_L which is assumed to be a power of 2, denoted by $V_L = 2^v$. We further assume that V_L is always less than or equal to the number of memory banks, M. This assumption is not a severe restriction since most existing supercomputers have more memory banks than the maximum length of vector registers [7].

5. The starting bank number for a vector access is assumed to be random with uniform distribution between 1 and M, but thereafter the bank number advances with some constant stride through the duration of this vector access. This constant stride is chosen to be 1 with probability P_{str1} and it takes any other integer between 2 and M equally likely if it is not 1.

6. Each memory bank has sufficient buffers to hold unsatisfied memory requests. This assumption is unrealistic because buffers are limited in real machines. However, we model the system using closed queueing network. The average queue length is less than one and there is also an upper limit for maximum buffer sizes because the queue population is fixed.

7. Only vector-vector interference is considered. The analysis for scalar-vector interference can be found in [1].

8. In solving the queueing network model, we assume that once a processor generates a vector request, all element requests associated with the vector access are placed in addressed memory banks at the same time.

The last assumption is unrealistic. In many existing vector computers, element requests of a vector access arrive at individual memory banks in a sequence of V_L clock cycles. Therefore, this assumption only approximates the actual behavior of the vector computer system. However, this assumption provides a very good approximation as evidenced later in this paper when we compare our analysis with real measurements. The reason is that we take into account the effect of request sequencing when we compute the total execution time of a vector operation. This is done by accumulating V_L response times of individual element requests to derive the execution time of one vector operation.

As mentioned in the introduction, memory delays can result from two types of bank conflicts: *self-interference* and *cross-interference*. Consider one vector access stream. If the access stride is one, all the V_L element requests go to different memory banks resulting in no conflict. If the stride

is not unity, however, depending on the relative value of the stride and the total bank number, the vector access may visit a number of banks less than V_L. We call the number of banks visited in one vector access stream a *return number* [10], denoted by N_r. This return number characterizes the degree of self-interference. In addition to the self-interference, when multiple vector access streams arrive at the memory system, element requests belonging to different vectors may visit the same bank resulting in cross-interference. The degree of cross-interference depends on the number of vector access requests, the return numbers of the vector accesses involved, and the number of banks etc.. The purpose of our analytical model is to model these two types of interferences based on the above assumptions.

3 Queueing Model

Let us first consider the self-interference of accessing one vector of data with stride s that plays a major role in deciding the addressing of each element request. Assuming the first element is addressed to bank f, then the second element is addressed to bank $(f+s) \mod M$, the third to bank $(f+2s) \mod M$ and so on. Based on the assumptions given in the previous section, the distribution of the stride of a vector access is easily obtained as

$$P\{s=i\} = \begin{cases} P_{str1}, & i=1, \\ \frac{1-P_{str1}}{M-1}, & i=2,\ 3, \cdots, M. \end{cases} \tag{1}$$

When a vector access stream traverses across all the memory banks (one sweep), the number of memory banks visited by the access stream, the return number (N_r), is given by $M/\gcd(M,s)$, where $\gcd(M,s)$ is the greatest common divisor of M and s. If $V_L > M/\gcd(M,s)$, memory conflicts occur within the vector access stream. For a special case where $\gcd(M,s) = 2^m = M$, all element requests go to a single memory bank. Since M is a power of 2, $\gcd(M,s)$ is in the form of 2^i for some $i = 0, \cdots m-1$. For each 2^i, there may be a number of values of s within M such that $\gcd(M,s) = 2^i$. In order to find the distribution of N_r, we need to find out the number of integers within M that share the same $\gcd(M,s) = 2^i$. This number is clearly the same as the number of values of s such that $\gcd(\frac{M}{2^i}, \frac{s}{2^i}) = 1$, which is Euler's function [11] given by

$$\phi(\frac{M}{2^i}) = \frac{M}{2^{i+1}} = 2^{m-i-1}, \quad for\ i < m. \tag{2}$$

For each s that satisfies $\gcd(M,s) = 2^i$, the return number is $N_r = 2^{m-i}$. Therefore, summarizing (1) and (2), we have

$$P\{N_r = 2^i\} = \begin{cases} \frac{1-P_{str1}}{M-1}, & i=0, \\ (1-P_{str1})\frac{2^{i-1}}{M-1}, & i=1,\ 2,\ \cdots v-1, \\ 1-\frac{V_L(1-P_{str1})}{2(M-1)}, & i=v. \end{cases} \tag{3}$$

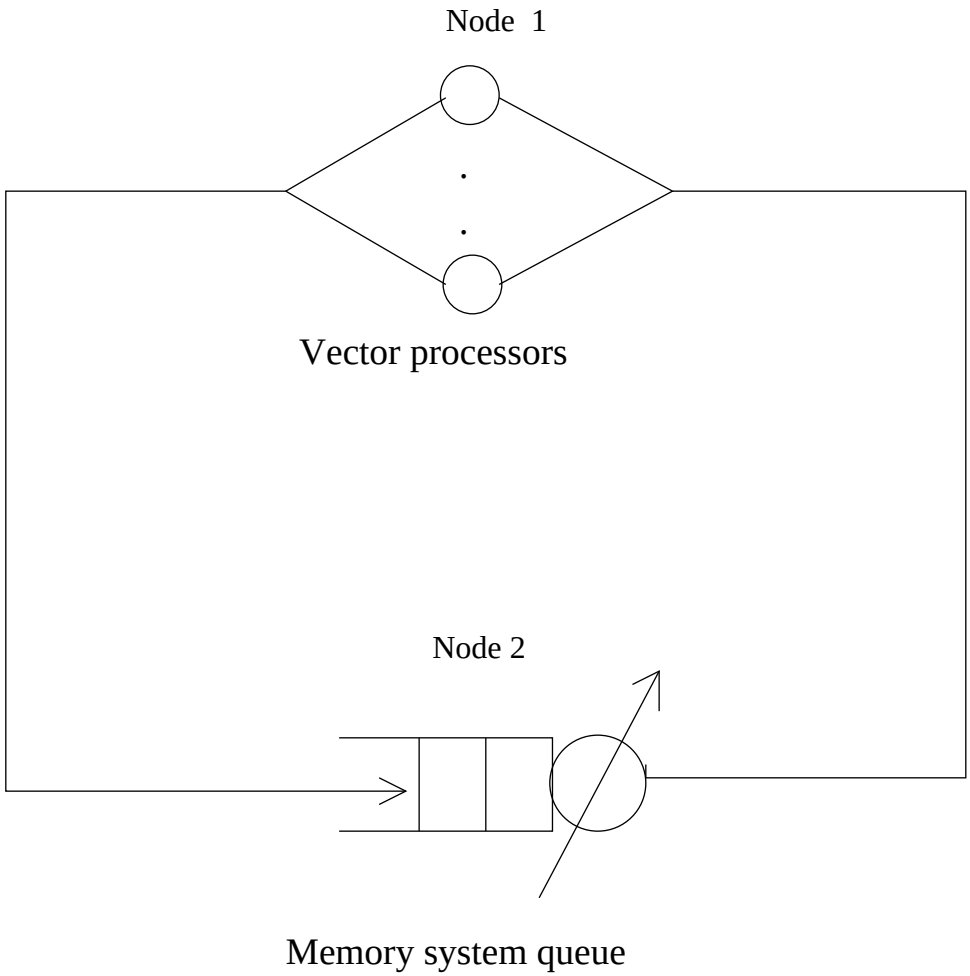

Figure 2: Queueing network representing the vector processor system.

From the above, we can see that self-interference occurs whenever $V_L > N_r$. In this case, element requests from the same vector request contend for one memory bank. Cross-interference occurs whenever element requests from two or more different vector requests contend for one memory bank. In order to investigate the influence of both self-interference and cross-interference, we model the above system as a two-node closed queueing network as shown in Figure 2. There is a total of N_P *jobs* or *customers* circulating in the network. Each of the N_P *jobs* represents a *vector access request* issued by a memory port. The first node is a conventional infinite-server queue. The service times are identical, independent and generally distributed random variables with a common mean Z. The event that job i is in the first node implies that memory port i is in a thinking process.

The second node consists of M parallel single-server queues, each of which represents a memory bank and the associated queue. Upon arrival at the second node, a job (a vector access consisting of $V_L = 2^v$ element requests) will split into 2^i groups of 2^{v-i} *subjobs* each with probability $P\{N_r = 2^i\}$ given in Equation (3). A *subjob* represents an *element request* of a vector access giving rise to

V_L $(= 2^v)$ subjobs in each job. The 2^i groups will join corresponding 2^i different queues. All subjobs are identical in the sense that they request exactly the same amount of memory service time t_m. In each queue, subjobs from the same job are served in a random sequence (Note that the order in which subjobs of the same job are served is insignificant) while subjobs from different jobs are served on the first-in-first-out basis. A job remains in the second node until all its subjobs are served.

An exact analytical solution to the above queueing network seems extremely difficult (if not impossible) to obtain due to the complicated way in which a job is processed at the second node. For example, although all the subjobs in a memory bank queue are processed in the order of arrival, the same does not necessarily apply to jobs. If all the subjob groups of a job happen to go to some empty memory bank queues (no cross-interference), that job may leave the second node before some other jobs that arrived earlier. Another complication factor is that the rate at which the second node processes jobs is not constant. In fact, it is directly related to the number of active memory banks (queues with subjobs) which in turn depends on the total number of jobs in the second node. Therefore, we can consider the second node as having a state-dependent service rate.

To attack such a complicated queueing network, we propose an approximation method which is based on the Mean Value Analysis (MVA) technique [12]. The key is to approximate the second node by a simple single-server queue with state-dependent service rate.

Now consider a job arriving at the second node. If the return number, N_r, is less than the vector length, V_L, N_r memory banks receive more than one subjob while other memory banks do not receive any subjob. The number of subjobs received by any one of these N_r memory banks is clearly a random variable. Let h denote this random variable. The distribution of h is given by

$$P\{h = 2^i\} = P\{N_r = 2^{v-i}\}, \qquad i = 0, 1, \cdots, v. \tag{4}$$

The expected value of h is given by

$$E[h] = \sum_{n=0}^{v} 2^n \cdot P\{N_r = 2^{v-n}\}.$$

Notice that the number of subjobs received by any memory bank must be a power of 2 since both M and V_L are power of 2. Substituting the probability distribution of N_r, we have

$$E[h] = 1 + \frac{vV_L(1 - P_{str1})}{2(M-1)}.$$

Let $\mu(k)$ be the average service rate at the second node when there are k jobs in the node. To find $\mu(k)$ we need to determine the number of active memory banks given that there are k jobs. For this purpose, we denote by X the number of subjobs received by an arbitrary memory bank due to

an arrival of a job at the second node. Since the starting vector element is uniformly distributed over the M memory banks, we have

$$P\{X = 2^n | N_r = 2^{v-n}\} = \frac{2^{v-n}}{M}, \quad n = 0, 1, \cdots, v.$$

Therefore, the distribution of X is given by

$$P\{X = 2^n\} = \frac{2^{v-n}}{M} P\{N_r = 2^{v-n}\}, \quad n = 0, 1, \cdots, v \tag{5}$$

and

$$P\{X = 0\} = 1 - \sum_{n=0}^{v} \frac{2^{v-n}}{M} P\{N_r = 2^{v-n}\}.$$

Simple algebraic manipulations lead to

$$P\{X = 0\} = 1 - (\frac{V_L}{M} + \frac{(1 - P_{str1})(1 - V_L^2)}{3M(M-1)}). \tag{6}$$

Next, let us consider that there are k jobs in the second node. Define $Y(k)$ to be the number of subjobs received by an arbitrary memory bank due to arrivals of k jobs. Clearly, $Y(k)$ is a random variable given by

$$Y(k) = X_1 + X_2 + \cdots + X_k.$$

The subscripts of X's in the above equation are insignificant since they have an identical distribution. Based on assumption (2), we know that X_i's are also independent. Therefore, the distribution of $Y(k)$ should not be difficult to obtain. In particular, we have

$$P\{Y(k) = 0\} = [P\{X = 0\}]^k = [1 - (\frac{V_L}{M} + \frac{(1 - P_{str1})(1 - V_L^2)}{3M(M-1)})]^k, \tag{7}$$

which is the probability that a memory bank is not active when there are k jobs in the second node.

We are now in the position to determine $\mu(k)$, the average service rate at the second node given that there are k jobs in the node. Since each job has V_L subjobs and each memory bank processes exactly one subjob in t_m time units, $\mu(k)$ should be equal to the average number of active memory banks divided by $V_L t_m$, i.e.

$$\mu(k) = \frac{M}{V_L t_m}[1 - P\{Y(k) = 0\}] = \frac{M}{V_L t_m} - \frac{M}{V_L t_m}[1 - \frac{V_L}{M} - \frac{(1 - P_{str1})(1 - V_L^2)}{3M(M-1)}]^k. \tag{8}$$

Assuming that the variation of $\mu(k)$ is small during the time interval between k jobs remaining and $k-1$ jobs remaining in the second node, $\frac{1}{2\mu(k)}$ can be used to approximate the backward recurrence time of the first of the k jobs leaving the node. We must point out here that $\frac{1}{2\mu(k)}$, instead of $E[ht_m]/2$, is used for the backward recurrence time because the use of the former captures

the fact that jobs at the second node are not first-in-first-out while the latter implies that jobs are served in the sequence of arrival.

Having analyzed the throughput characteristics of the memory system queue with k customers, we are now ready to solve the overall queueing network. If we use the mean value analysis (MVA) algorithm [12] to solve the queueing network, then the mean response time seen by a job during the nth iteration of the MVA algorithm is given by

$$R(n) = E[h]t_m + \sum_{j=1}^{n} t_m E[Y_{\max}(j-1)]P_q\{j-1|n-1\} - \sum_{j=2}^{n} \frac{1}{2\mu(j-1)} P_q\{j-1|n-1\}, \quad (9)$$

where $P_q\{j-1|n-1\}$ is the probability that there are $j-1$ jobs in the second node given the population size of $n-1$ and $E[Y_{\max}(j-1)]$ is the average waiting time (excluding its service time) of a job given that there are $j-1$ fresh jobs already in the node (the word "fresh" means that none of the $j-1$ jobs has started its service). Unlike the traditional MVA algorithm, we have included the backward recurrence time in the above formula in order to catch the effects of nonexponential service times for jobs at the second node. The validity and correctness of this technique have been studied in [13, 14].

To evaluate $E[Y_{\max}(k)]$, we note that the random variable $Y_{\max}(k)$ depends on the value of N_r associated with the $(k+1)$st job. Without loss of generality, let us assume that the $(k+1)$st job uses memory banks $1,\ 2, \cdots,$ and N_r. Let $Y_i(k)$ be the number of subjobs received by bank $i,\ i = 1, 2, \cdots,\ N_r$, due to the arrival of k jobs. The memory access delay of the $(k+1)$st job is clearly determined by the subjob that joins the memory bank having the longest waiting queue. Thus,

$$Y_{\max}(k) = \max[Y_i(k), i \in (1, \cdots, N_r)].$$

Clearly, $Y_i(k)$'s have the same distribution as $Y(k)$ but are not independent. However, their mutual dependence is very weak as evidenced by our simulation experiments to be discussed shortly. Therefore, we assume that $Y_i(k)$'s are all independent and we have

$$P\{Y_{\max}(k) \le x\} = \sum_{n=0}^{v} (P\{Y_i(k) \le x\})^{2^n} P\{N_r = 2^n\},\ x = 0, 1, \cdots, k \cdot V_L.$$

From basic probability concept, we have

$$E[Y_{\max}(k)] = \sum_{x=0}^{kV_L} P\{Y_{\max}(k) > x\} = \sum_{x=0}^{kV_L} (1 - P\{Y_{\max}(k) \le x\}).$$

The queue length distribution, $P_q\{j-1|n-1\}$, is computed as follows. If there is only one job in the second node, the expected service time is clearly $t_m E[h]$. If there are j jobs, the second node finishes $\mu(j)$ jobs per unit time. However, this service rate changes as the number of jobs varies.

On average, the second node would finish $\frac{1}{j}\sum_{k=1}^{j}\mu(k)$ jobs per time unit provided that there were j jobs to begin with. Let $\hat{\mu}(j) = \frac{1}{j}\sum_{k=1}^{j}\mu(k)$. We have

$$P_q\{j|n\} = \begin{cases} \mathrm{Thr}(n)\cdot E[h]\cdot t_m \cdot P_q\{j-1|n-1\}, & j=1 \\ \mathrm{Thr}(n)\cdot (\frac{1}{j}E[h]\cdot t_m + \frac{j-1}{j}\frac{1}{\hat{\mu}(j-1)})\cdot P_q\{j-1|n-1\}, & j=2,\cdots,n \\ 1-\sum_{i=1}^{n} P_q\{i|n\} & j=0, \end{cases} \tag{10}$$

where $\mathrm{Thr}(n)$ is the system throughput at the nth iteration of the MVA algorithm. By Little's formula, we have

$$\mathrm{Thr}(n) = \frac{n}{Z+R(n)}.$$

We now present the MVA algorithm that is needed for computing $R(n)$, the average delay of a vector access at the memory system, and $\mathrm{Thr}(n)$, the system throughput:

Algorithm MVA

1. Compute distribution of $Y(k)$ by convolving k distributions given in Equations (5) and (6) for $k = 1,\ 2,\cdots,N_P$;
2. Initialize queue length distribution, $P_q\{0|0\} = 1$, and $P_q\{j|n\} = 0$ for all j and n;
3. (Main Loop) For $n = 1,\cdots,N_P$ perform 4 to 6;
4. Compute $R(n)$ using Equation (9);
5. Compute throughput: $\mathrm{Thr}(n) = \frac{n}{Z+R(n)}$;
6. Compute queue length distribution using Equation (10);

We have written a program in PASCAL to solve the queueing network of Figure 2 based on the above analysis. The complexity of the algorithm is $O(N_P^2 * V_L)$. One can obtain performance estimates of a system with $N_P = 64$ and $M = 256$, for example, within a second on a PC or workstation.

4 Model Validations

In order to verify the correctness of our analytical model developed in the previous section, we have carried out experimental studies to validate the model. In this section, we compare the numerical results obtained using the analytical model to those obtained through simulation experiments and to those from real measurements reported by independent researchers.

Selecting proper input parameters for the queueing model is essential to the output of the model. Therefore, care must be taken in determining these input parameters. We try to choose the system parameters as close to those of an existing vector machine (the Cray Y-MP) as possible. We assume in the analytical model that there are 128 or 256 interleaved memory banks each of which has the memory cycle time of 5 processor cycles, i.e. $t_m = 5$. The vector register length is assumed to be 64 elements. Other system parameters will be shown in the corresponding tables and figures presented below.

4.1 Performance Metric

The commonly used performance parameter for vector computers is the *total execution time* of a sequence of operations on a vector of length B, T_B. This quantity depends on a number of factors including the overhead for computing the starting addresses and setting up vector controls, the overhead for executing scalar code for strip-mining, and the maximum vector register length, V_L. Bucher and Simmons have developed a model to calculate vector execution time [7]. Based on their model, the total execution time of a vector operation is given by

$$T_B = T_o + \lceil \frac{B}{V_L} \rceil \cdot T_{start} + B \cdot T_{elemt}, \tag{11}$$

where

T_o is the startup time for the outer loop,
T_{start} the start-up time for each inner most loop,
B the vector length,
V_L the vector register length,
T_{elemt} the time for processing one element of the vector ignoring the startup time.

In the above formula, we incorporate memory delays caused by interferences into T_{elemt}. The analytical model developed in the previous section is used to evaluate the memory delays. In particular, the response time, $R(N_P)$, is the average memory delay faced by a processor for a vector access in an N_P-processor system. In practice, a machine with chaining facility can start processing a vector element as soon as the element arrives at the CPU. Nevertheless, the completion time of processing a whole vector is still dependent on the latest element to arrive at the processor. We shall use $R(N_P)$ divided by V_L as the average memory stalls for each vector element. This quantity and the arithmetic operation time for processing an element result (1 cycle) constitute our T_{elemt}, which is given by $T_{elemt} = 1 + R(N_P)/V_L$. Once T_{elemt} is determined, the total execution time can be calculated using Equation (11).

4.2 Comparison with Simulations

For the purpose of verifying our analytical model, we have written a simulator. The simulator is synchronous (Time-driven Simulation) in the sense that all the system activities occur at discrete time intervals which are processor cycles. Each processor in the simulator can be in one of two states: *active* doing useful computations or *idle* waiting for a memory access. At the beginning of each simulation cycle, we check all the active processors to see if they generate vector access requests using a random number generated based on a uniform distribution. If it turns out that a vector access request is generated from a processor, the processor is made idle. The starting memory bank number of the vector access is selected from the M memory banks equally likely. At the mean time the access stride is also determined based on the random number following the assumed distribution. Once the starting address and the stride are determined, V_L element requests are then appended to the corresponding memory bank queues. Every two consecutive element requests of the vector are separated in terms of memory bank number by the chosen constant stride. Before the cycle ends, we update all the memory queues, decrease the remaining memory time of the request at the head of a queue, and return the request to the requesting processor if the remaining time is 0. If an idle processor has collected all V_L element requests, it resumes active for the following cycle.

The statistics that we collected from the simulator is the proportion of time that a processor is active. This proportion is known as processor utilization denoted by U_p. It is obtained by accumulating active cycles of a processor and dividing the sum by the total run time. The final result is the average over all N_P processors in the system. We run the simulator for 1,000,000 cycles in each simulation run. All the simulation results reported in this paper are the average of 10 such runs. Using Little's law, we can easily derive the average response time of a vector access request which is given by $Z/U_p - Z$[12]. Assuming that the vector length of application programs is 256 ($B = 256$), the total execution time, T_B, is readily derived using formula (11). In all the following tables, we assume T_o to be 4 cycles and T_{start} to be the think time Z.

Table 1. Comparison between analysis and simulation in terms of vector execution time for data size $B = 256$.

(M=128, V_L=64, $N_P = 1$, $t_m = 5$)

P_{str1}	Z =1		Z =5	
	Simulation	Analysis	Simulation	Analysis
0.1	311.22	311.22	327.80	327.34
0.2	308.37	308.19	324.94	324.31
0.4	302.06	302.15	316.18	318.14
0.8	289.85	290.08	305.25	306.08

We first verify our analysis by varying the fraction of unit stride, P_{str1}. Table 1 shows the comparison between the simulation results and the analytical results with one memory port ($N_P = 1$). It is shown in the table that the two results are in an excellent agreement for all considered values of P_{str1}. With one processor, only self-interferences exist. Therefore, this table indicates that our analysis models self-interferences fairly accurately. Next, we increase the number of processors to 8 in order to check the accuracy of the model for both self-interferences and cross-interferences. Table 2 shows the execution times for 2 different think times. Note that all 8 processors carry out the same vector operations on vectors of length $B = 256$. Similar agreement is also observed for the system with 8 processors as shown in Table 2. In all cases, the maximum difference between the simulation and the analysis is about 6%, indicating a high accuracy of our analytical model.

Table 2. Comparison between analysis and simulation in terms of vector execution time for data size $B = 256$.

(M=128, V_L=64, $N_P = 8$, $t_m = 5$)

P_{str1}	Z =1		Z =5	
	Simulation	Analysis	Simulation	Analysis
0.1	467.25	461.01	480.99	470.53
0.2	458.02	453.24	461.41	461.21
0.4	446.05	436.21	446.92	443.49
0.8	397.93	396.99	398.89	403.88

In the above comparisons, we deliberately make the workload very high: $Z = 1$ and 5. We did so because of the following two reasons. First of all, high workload creates more memory activities resulting in shorter simulation time than low workload for the same accuracy. Secondly, analytical models are generally good for low to medium loads [14] whereas they are not as good at high system load. Thus, validating the analytical model at high workload implies a conservative conclusion. However, in order to make our validation more convincing, we have run the simulator at low workload: $Z = 25, 50$, and 100. To achieve reasonable accuracy, we increased the run time by 10 times, which takes several days. Table 3 lists the numerical results under relatively low workload for different number of processors. Again, the execution times are defined as the average execution time for a processor to finish vector operations on a vector of length B=256 assuming that all processors perform the same operations. As expected, the analytical results match well with the simulation results.

Table 3. Comparison between analysis and simulation in terms of vector execution time for data size $B = 256$.

(M=256, V_L=64, $P_{str1} = 0.2$, $t_m = 5$)

N_P	Z	=100	Z	=50	Z	=25
	Simulation	Analysis	Simulation	Analysis	Simulation	Analysis
1	690.57	691.97	492.56	492.02	391.75	392.10
2	693.37	693.84	497.25	495.85	399.28	398.50
4	701.01	698.60	506.61	504.80	416.25	415.52
8	713.51	709.94	526.67	526.67	444.50	447.97
16	739.62	735.62	570.56	568.17	499.23	496.41
32	794.05	785.62	648.35	632.44	591.13	569.60

4.3 Comparison with Measurements

In this subsection, we compare the performance results obtained from our analytical model with those of measurements independently carried out by other researchers. The measurement data are taken from previously published results by Bucher and Simmons [7]. In [7], performance measurements were performed on four vector machines: the Cray X-MP, Cray Y-MP, Cray 2 with static memory, and Cray 2 with dynamic memory. Among these four machines, the Cray Y-MP provides an independent data path for each processor to each memory bank whereas the other three are prone to interconnection network conflicts. Since our analytical model assumes ideal interconnection network, we will only compare our results with that of the Cray Y-MP.

The compiler running on the Cray Y-MP practices vector loop unrolling for performance optimization. As a result, Equation (11) in the previous subsection needs to be modified to take into account this effect. The level of unrolling for vector loops is two implying that the compiler processes two strips of length V_L in a single pass through the unrolled loop. Therefore, the execution time can be expressed as [7]

$$T_B = T_o + \lfloor \frac{B-1}{2 \cdot V_L} \rfloor \cdot T_{start} + \lceil \frac{B \mod (2 \cdot V_L)}{V_L} \rceil \cdot T_{start} + B \cdot T_{elemt}. \quad (12)$$

In the above expression, the second term gives the total unrolled loop startup times and the third term gives the startup cost of remaining single strips. Because the startup times for a strip and an unrolled loop are approximately the same, we use T_{start} for both cases. As mentioned previously, T_{elemt} takes into account the memory access delays of individual vector elements which is the output of the queueing model presented in the previous section. Our first step is, therefore, to apply the queueing model to derive the average memory access time for each vector element ($R(N_P)/V_L$). We then substitute T_{elemt} ($= 1 + R(N_P)/V_L$) into Equation (12) to obtain the total execution time of

a vector operation.

In all the comparisons presented below, we set the probability of unit stride vector access, P_{str1}, to 1, i.e. all vector accesses are stride 1 accesses. In determining the startup cost, we use the undisturbed vector execution time measured on the Cray Y-MP (Figure 1 in [7]) as the basis. The difference between the actual execution time and the ideal execution time (one result per cycle) implies the startup cost. Since the measured execution time for a 64 element vector is 68 and the time for a 1024 element vector is 1145, we approximate the outer loop startup time T_o to be 4 cycles and the strip as well as the unrolled loop startup times, T_{start}, to be 8. Recall that the processor think time is the interval between the completion of a vector access and the generation of the next vector access request. The measurements reported in [7] assume repeated vector operation: $V = V + V$, the addition of a contiguous vector to another contiguous vector on each processor. In this situation, the processor think time can be considered as the time taken for a processor to handle the loop overheads between two consecutive vector operations. It is, therefore, reasonable to assume that the think time is the same as the startup cost which is 8 cycles and is used in our model.

Figure 3 shows the comparison between our analysis and the measurements reported in [7]. The measurement data are the execution times of the test process on the Cray Y-MP with one up to eight processors presented in Figure 4a of [7]. We plotted separately the execution times corresponding to 1, 2, 4 and 8 processors in Figures 3a, 3b, 3c and 3d, respectively. It is shown in these figures that the execution times calculated through our analytical model show the same trend as that of the measurements as the vector length changes. Moreover, the two curves in each figure are very close, indicating the high accuracy of the analysis.

As mentioned in Section 2, we made an unrealistic assumption that all element requests arrive at addressed memory banks at the same time. In reality, the V_L element requests associated with a vector access request should arrive at addressed memory banks in a sequence of V_L cycles. Even with this assumption, our analysis still models closely the real execution behavior of the vector computer. The reason is that this assumption is only used in solving the queueing network model to derive the average memory delay of each element request. This memory delay is measured from the arrival of an element request at the memory bank until the return of the accessed data to the CPU. Statistically, the average memory delay for individual memory requests should be approximately the same whether the V_L element requests arrive in sequence or at once since we are interested in the interval between the arrival and the departure of an element request at a memory bank. On average, this memory delay depends only on the relative timing of arrivals of element requests at a memory bank. For cross interferences, this relative timing solely depends on the time instances at which vector access requests are generated. Therefore, the queueing delay of each

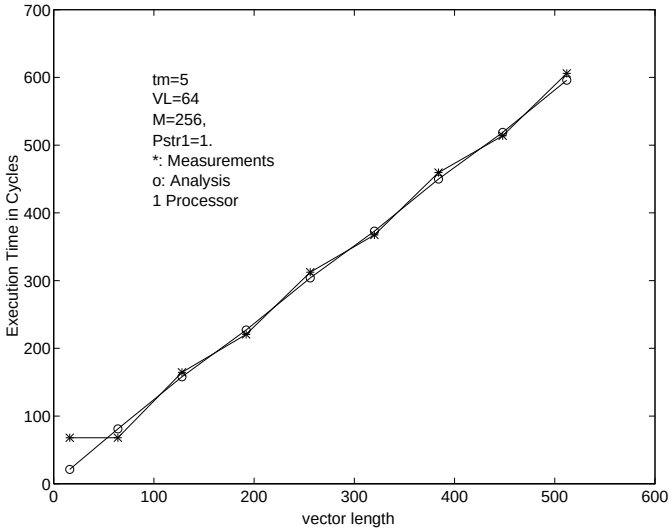

Figure 3a. Comparison with measurement data.

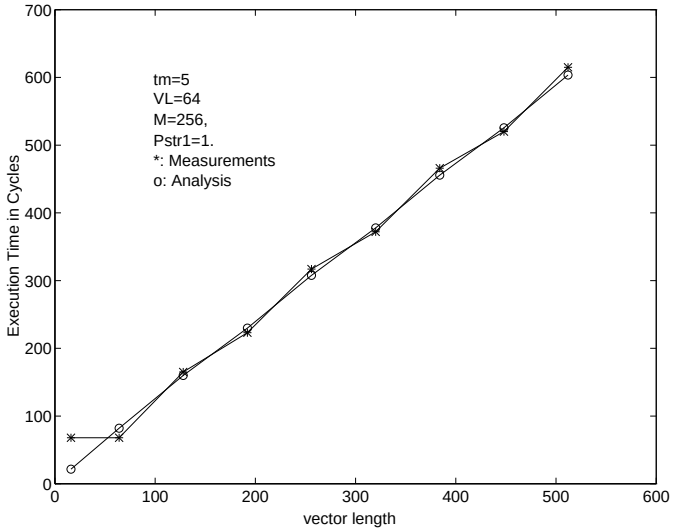

Figure 3b. Execution time of 2 processor system.

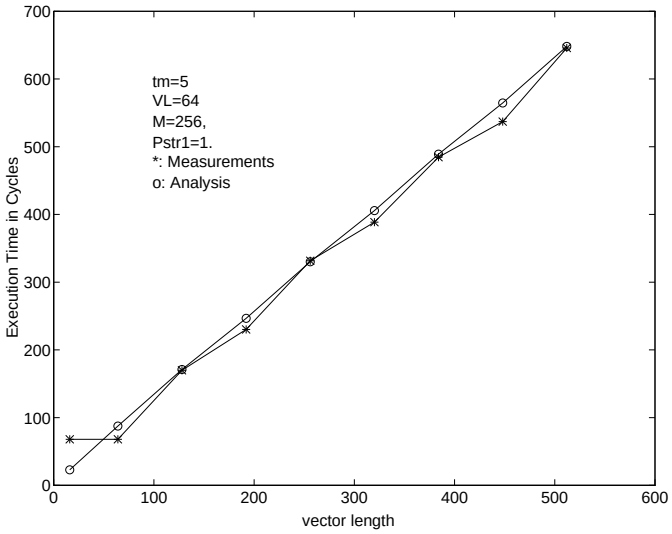

Figure 3c. Execution time of 4 processor system.

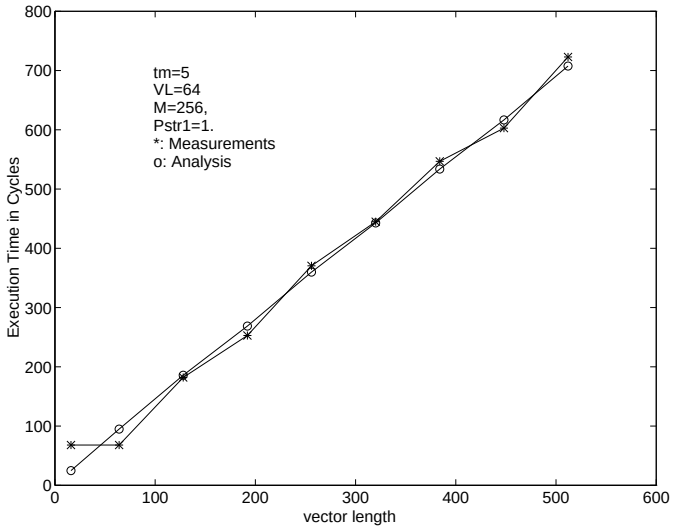

Figure 3d. Execution time of 8 processor system.

Figure 3: Validation of the model with measurement data

individual element request calculated using this assumption is fairly accurate. The effect of request sequencing is taken care of by Equation (12) when we compute the total execution time of a vector operation.

5 Numerical Results and Discussions

We are now ready to use our analytical model to carry out performance evaluations of the vector computers. As in the previous subsection, we assume 2 levels of loop unrolling in our following discussions. Therefore, Equation (12) will be used to calculate the execution time of a vector operation.

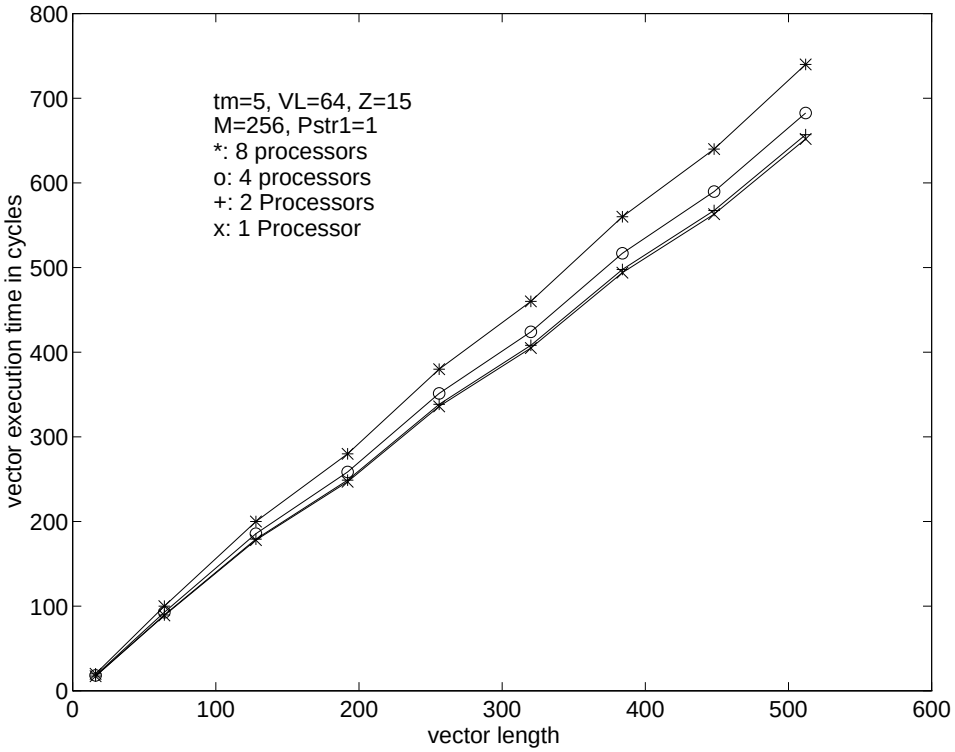

Figure 4: Vector execution time as a function of vector data length.

We first plotted the vector execution time as a function of vector length, B, as shown in Figure 4. Four curves are drawn for the number of processors in the system being 1, 2, 4, and 8, respectively. Since the probability of stride being 1 is assumed to be 1 in this figure, there is no self-interference for vector accesses. However, cross-interferences among multiple vector access streams increase as the number of processors in the system increases. As a result, the vector execution time increases with the increase of the number of processors in the system. The more processors there are in the system, the more memory contentions there will be due to cross-interferences.

In order to take into account both the effect of memory delay and the effect of system size in terms of the number of processors in a system, we will use the *average number of element results per cycle* processed by the whole system as the performance measure in our following discussions. The *average number of element results per cycle* is defined as

$$\frac{Total\ data\ size\ in\ terms\ of\ No.\ of\ vector\ elements}{Total\ execution\ time\ on\ the\ data} \cdot Number\ of\ Processors$$

$$= \frac{B}{T_B} \cdot N_P.$$

In an ideal situation, we expect that a single processor can process one vector element per cycle and an N_P-processor system should produce N_P results per cycle. Unless otherwise specified, the probability that a vector is accessed with stride 1, P_{str1}, is assumed to be 0.7 which is the average

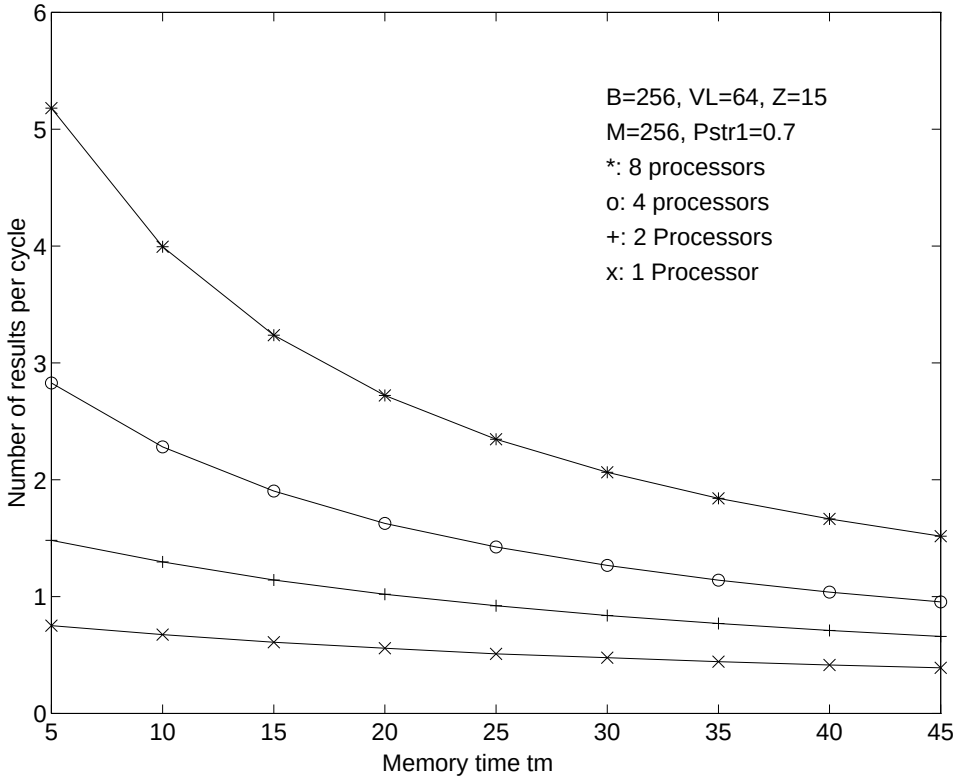

Figure 5: Performance vs memory reservation time t_m.

value of proportion of stride one accesses in a typical vector code.

The trend of the computer technology is that the speed gap between processor and memory is getting larger. To see the effects of this technology trend on performance of vector computers, we plotted the number of results per cycle as a function of memory cycle time or memory reservation time, as shown in Figure 5. Similar to Figure 4, four curves are plotted for four different system sizes. While the performance degradation due to large memory cycle time in a single processor system is tolerable, the performance of large systems (more processors) drops rapidly as the memory cycle time increases. In a single vector processor system, proper pipelining of memory accesses can well overcome the problem of memory latency for slow memories. However, as more vector processors are added to the multiple vector processor system, memory latency affects system performance significantly and becomes a system bottleneck. Other researchers have drawn similar conclusions [9, 7]. As shown in Figure 5, the performance of the 8-processor system drops by a factor of 3 as the memory cycle time increases from 5 to 45 CPU cycles. For large systems, pipelining itself may not be sufficient to attain high performance. Large register file, excessive number of memory banks or some kind of memory hierarchy designs [15, 16, 17, 18] would be necessary to maintain high vector performance for systems with a large number of processors.

The effects of system size in terms of number of processors are further illustrated in Figure 6.

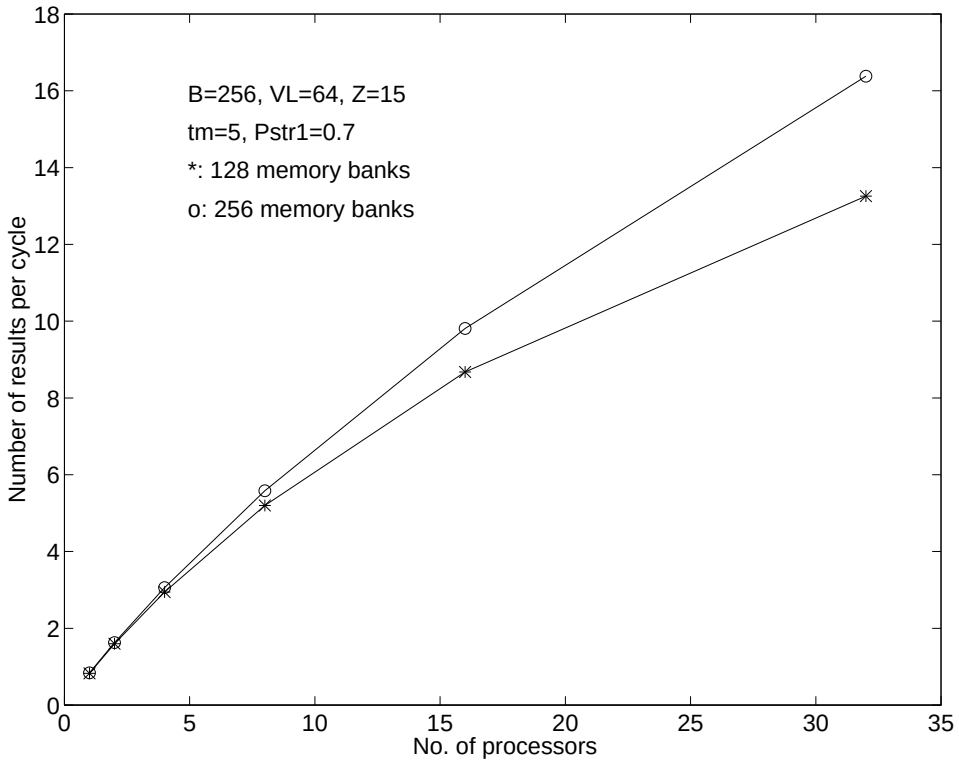

Figure 6: Performance vs number of processors.

The two curves in the figure correspond to two difference numbers of memory modules with the same memory cycle time $t_m = 5$. We notice in this figure that the system with one processor can produce close to one vector element result per cycle. However, as the number of processors in the system increases, the number of results produced per cycle is not linearly related to the number of processors. In other words, the performance gains are far less than linear with respect to the increase of the number of processors. Therefore, by just adding more processors to a system one may not obtain linear performance increase because of memory latency. Memory system plays an important role in a multiple vector processor system. It is therefore necessary to have a high performance memory system in order to achieve high parallelism in a multiple vector processor system. It is shown in Figure 6 that with 256 memory banks, the system performs better than that with 128 banks. A large number of memory banks will reduce memory bank interferences and increase parallelism of memory operations. This fact is further illustrated in Figure 7 where vector performances are plotted as functions of number of memory banks for different size systems.

Although high degree of memory interleaving reduces memory conflicts, a large number of memory modules necessitates complex interconnection network. Not only does the complex interconnection network increase the hardware cost but also enlarge the memory access time due to network delays. Another common approach to achieving high memory performance in vector computers is

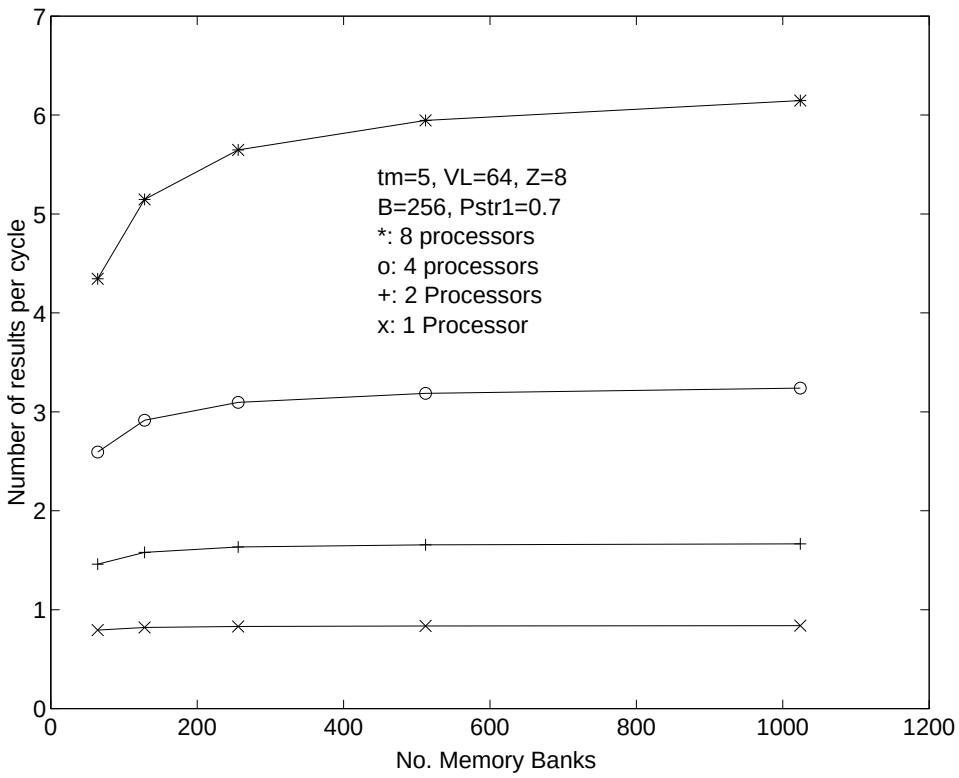

Figure 7: Performance vs number of memory banks.

to have large register files or vector cache memories [19]. The effect of having large register files and vector cache memories is the reduction of memory request rate. In our analytical model, reduced memory request rate implies increased processor think time. Figure 8 shows the effects of processor think time on vector performance for different numbers of vector processors in a system. We change the processor think time in the multiple of 64. Each increase of 64 cycles in processor think time indicates that one more vector operation of length V_L is performed within vector registers. For example, for processor think time being $Z = 64i + T_{start}$, $64i$ elements are processed within vector registers while the remaining $B - 64i$ elements are computed through the main memory. Since there is no memory access during the processor think time, one vector element result is computed every cycle not considering startup overhead. As a result, T_{elemt} in formula (12) is 1. Large processor think time results in better performance as shown in Figure 8. This observation becomes more evident if the ratio of memory cycle time and processor cycle time is large. In Figure 9, we show the performance versus processor think time for larger memory cycle time, $t_m = 31$, which is the average memory reservation time of Cray 2. In this case, the slope of the performance curves becomes much larger, which suggests that there is a great potential in gaining better performance by means of a proper design of memory hierarchy.

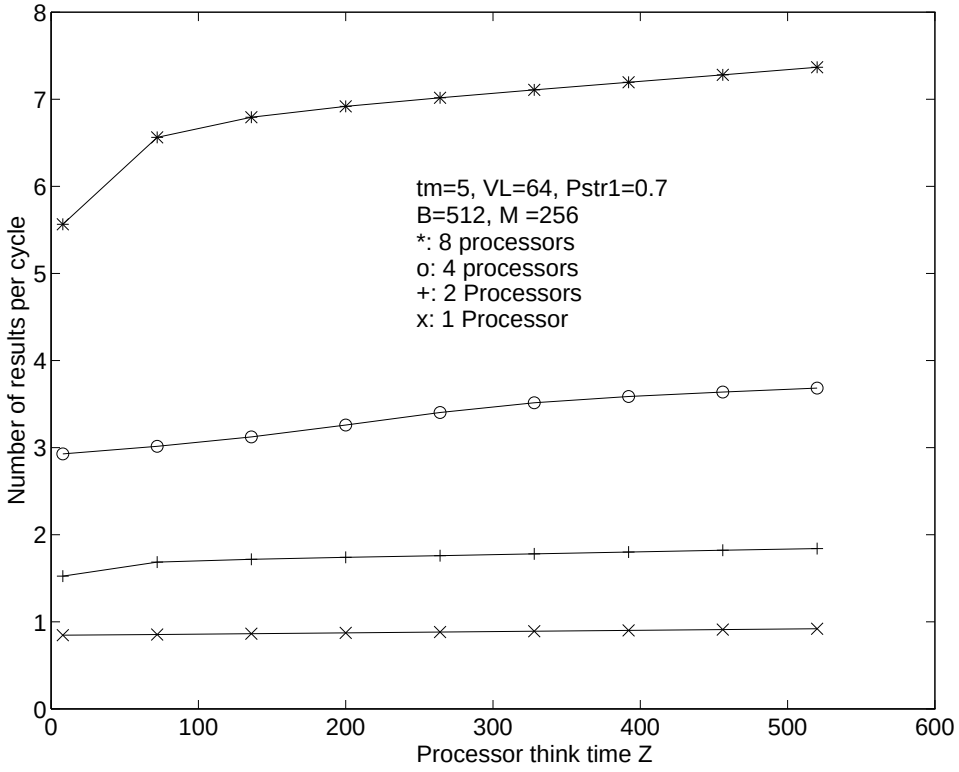

Figure 8: Performance vs processor think time Z.

6 Conclusions

In this paper we have developed an analytical model for memory interferences due to vector accesses in multiple vector processor systems. Each vector access is assumed to be regularly patterned as it is in real vector computers. Both self-interference and cross-interference of multiple stream vector accesses are considered in the model assuming that the stride of each vector access is a random number with a uniform distribution except for a certain specified large probability of unit stride. We have proposed approximation techniques to analyze the complicated queueing behavior of the memory system that consists of a number of independent banks. The analytical model can be solved very efficiently using the given Mean Value Analysis algorithm.

Extensive simulation experiments have also been carried out to validate our analytical model. It has been shown that analytical results match very well with simulation results over all possible load conditions with the maximum error of less than 8%. We have also validated our analysis by comparing the numerical results calculated from our analytical model with previously published measurements on a real multiple vector processor machine. Using the analytical model, we have studied the effects of memory interferences on the performance of vector computers. Our numerical results show that memory system is the major bottleneck for large scale multiple vector processors.

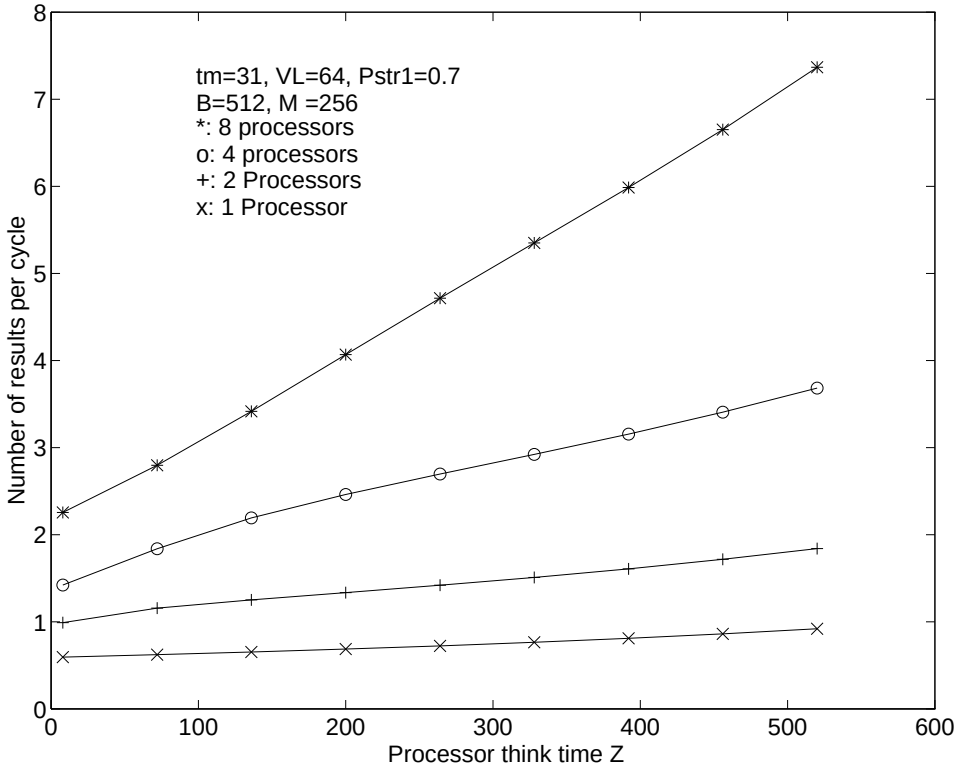

Figure 9: Performance vs processor think time Z.

Therefore, it is essential to have high performance memory system in order to achieve high vector performance in multiple vector processor systems.

Acknowledgements

The authors are very grateful to Dr. Ingrid Y. Bucher for providing us with the measurement data on Cray Y-MP, which helps us greatly in selecting system parameters and drawing Figure 3 of this paper. The authors would also like to thank the anonymous referees for their valuable comments that provide great help in improving the quality of the paper.

References

[1] R. Raghavan and J. P. Hayes, "Scalar-vector memory interference in vector computers," in *Proc. Int'l Conf. On Parallel Processing*, pp. I180–187, Aug. 1991.

[2] D. H. Bailey, "Vector computer memory bank contention," *IEEE Trans. on Computers*, vol. C-36, pp. 293–298, March 1987.

[3] I. Y. Bucher and D. A. Calahan, "Access conflicts in multiprocessor memories queueing models and simulation studies," in *Proc. 1990 Int'l Conf. on Supercomputing*, pp. 428–438, 1990.

[4] D. A. Calahan, "Some results in memory conflict analysis," in *Supercomputing'89*, pp. 775–778, 1991.

[5] P. Tang and R. H. Mendez, "Memory conflicts and machine performance," in *Proceedings of Supercomputing'89*, pp. 826–831, 1989.

[6] D. Y. Chang, D. J. Kuck, and D. H. Lawrie, "On the effective bandwidth of parallel memories," *IEEE Trans. on Computers*, vol. C-26, pp. 480–489, 1977.

[7] I. Y. Bucher and M. L. Simmons, "Measurement of memory access contentions in multiple vector processor systems," in *Supercomputing'91*, Nov. 1991.

[8] D. T. Harper III, "Block, multistride vector, and FFT accesses in parallel memory systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, pp. 43–51, Jan. 1991.

[9] D. T. Harper III and J. R. Jump, "Vector access performance in parallel memories using a skewed storage scheme," *IEEEE TRANS. on Comput.*, vol. C-36, pp. 1440–1449, 1987.

[10] W. Oed and O. Lange, "On the effective bandwidth of interleaved memories in vector processor systems," *IEEE Trans. on Computers*, vol. C-34, pp. 949–957, Oct. 1985.

[11] A. J. Pettofrezzo and D. R. Byrkit, *Elements of Number Theory.* Prentice-Hall, 1970.

[12] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance—Computer System Analysis Using Queueing Nework Models.* Englewood Cliffs: Prentice-Hall, 1984.

[13] Q. Yang and L. N. Bhuyan, "Analysis of packet-switched multiple-bus multiprocessor systems," *IEEE Transactions on Computers*, March 1991.

[14] Q. Yang, L. Bhuyan, and B.-C. Liu, "Analysis and comparison of cache coherence protocols for a packet-switched multiprocessor," *IEEE Trans. on Comput.*, vol. 38, pp. 1143–1153, August 1989. Special Issue on Distributed Computer Systems.

[15] Q. Yang and L. W. Yang, "A novel cache design for vector computers," *19th Ann. Int'l Symp. on Computer Architectures*, pp. 362–371, May 1992. Queensland, Australia.

[16] K. So and V. Zecca, "Cache performance of vector processors," in *Proc. 15th. Int'l Symp. on Comp. Arch.*, pp. 261–268, 1988.

[17] J. W. C. Fu and J. H. Patel, "Data prefetching in multiprocessor vector cache memories," in *Proc. 18th. Int'l Symp. on Comp. Arch.*, pp. 54–63, 1991.

[18] M. S. Lam, E. E. Rothberg, and M. E. Wolf, "The cache performance and optimizations of blocked algorithms," in *Proc. of Arch. Supp. for Prog. Lang. and Opr. Sys.*, pp. 63–74, April 1991.

[19] Q. Yang, "Introducing a new cache desgin into vector computers," *IEEE Trans. on Computers*, vol. 42, pp. 1411–1424, Dec. 1993.