Comparison of analytic performance models using closed mean-value analysis versus open-queuing theory for estimating cycles per instruction of memory hierarchies

R. E. Matick

Analytic models provide a simple but approximate method for predicting the performance of complex processing systems early in the design cycle. Over the years, extensive use has been made of various queuing models to analyze the memory hierarchies of multiprocessor systems in order to estimate the finite cache penalty and resulting system performance measured in cycles per instruction executed. Two general modeling techniques widely used for such performance evaluation are the open-system and closed-system queuing theories. Closed-queuing models can be solved by various methods, but mean value analysis is the most common for closed systems of the type considered here. The basic differences between these two approaches have been somewhat obscure, making them difficult to compare. This work explores some fundamental issues from a practical engineering viewpoint with the intention of illuminating the essential differences in the general techniques at the very basic level. In addition, the results of a detailed study comparing the two in a moderately complex multiprocessor memory hierarchy are presented.

Introduction

In its most simple form, any computer system can be considered to be composed of a number of servers, each with some given average service time. A server is any shared resource with a queue, such as a bus or memory array. These servers can be arranged in various series (tandem) and parallel combinations, and inevitably give rise to queues at each server as a result of contention from multiple requests. In a real system, queues can arise only at places where registers or other methods are provided to maintain a queue ahead of a shared resource. Such places will thus produce a server with a queue. We assume that these locations are known and constitute the system to be modeled.

In a uniprocessor or multiprocessor configuration, any given processor can typically process instructions at some idealized performance rate, expressed in cycles per instruction (CPI) executed with infinite cache, called $CPI[\infty]$ [1]. This performance rate includes accesses to the first-level cache (L1), but not to any other cache or memory subsystem. The implicit assumption is that there are no misses in L1 (i.e., infinite L1 capacity) or that any miss requires zero time to reload. If there is no L1 cache, $CPI[\infty]$ is the performance, assuming zero time for any memory accesses. In either case, this is the maximum performance that can ideally be obtained. The performance is proportional to $1/CPI$, so a smaller CPI gives faster performance. Because all real systems have

0018-8646/03/$5.00 © 2003 IBM

one or more finite-speed memory subsystems attached, there will be some additional cycles per instruction as a result of this memory access delay for L1 misses. In a system with a simple memory hierarchy—an L1 and only one main memory attached—every L1 miss incurs a fixed delay. In a system with a multiple-level cache hierarchy, each memory access can incur a different delay, and the same access at different times can incur different delays, determined by which level of the hierarchy contains the desired access (*access hit*) at the moment. An L1 miss will first access the fastest, and therefore smallest, cache, typically L2. If a hit occurs, a block (i.e., *cache line*) or several bytes are returned to the processor. If this level of cache does not contain the request, a miss occurs, and the request propagates to the next level of cache. A hit at this cache level returns the data with a longer delay than would have occurred if the request had been in the first level. If a miss also occurs here, the request propagates to the next level, etc., until a level is reached that returns only hits. This additional-delay adder, measured in CPI, is called the *finite cache penalty* (FCP). This quantity is evaluated in both open and closed queues to give the final system performance as

$$CPI = CPI[\infty] + FCP. \quad (1)$$

The delay path for processing any memory request is the sum of all sequential queuing times (line plus service times)[1] along its path from input to output. Different requests can traverse different paths and thus can incur different delays. An average delay is determined on the basis of some given request rates such as miss rates or number of customers in the system, discussed below. Such behavior can be observed in the pipeline of the processor itself, the memory hierarchy, or a combination. This work is focused primarily on queuing analysis of memory hierarchies, although the fundamentals are applicable in general.

The major problem in all analytical queuing analysis is to determine the queue delays (i.e., average residence time) at each server. Early queuing models were based mainly on open-network analysis because closed queuing techniques prior to 1980 were extremely difficult and cumbersome, requiring excessive computation time. Open queues require that the number of customers (i.e., reload requests) cannot be fixed at any point in the system. In fact, all queues must theoretically be capable of unlimited length, although the actual value is usually quite small. However, in an actual multiprocessor system, each processor typically permits only some fixed number of outstanding misses to exist within the memory hierarchy at any time. The reason is that unless there are special provisions to ensure correct ordering of all operations, a memory access miss stalls the central processing unit (CPU), which cannot continue processing until a miss-reload request is fulfilled (often referred to as a *blocking cache*). The number of outstanding misses per CPU is usually one miss total, or one instruction-cache miss and one data-cache miss in systems that have two such caches. Newer systems are beginning to allow more simultaneous outstanding misses (i.e., *nonblocking cache*), but still permit only a small, fixed number per CPU. Some systems allow additional misses for prefetches, which are not real misses in the sense that they need not halt the CPU, but they do add to the queues. In any case, the number of outstanding misses is fixed. This means that if we were to look at all queues within the memory hierarchy at any instant of time, the maximum total requests for reloading misses would be this fixed number. Of course, there could be fewer at any instant; only the maximum is fixed.

Mean value analysis (MVA) is an analytical technique that allows the total number of customers (n) in the full system to be arbitrarily fixed at a constant value. The theoretical aspects of MVA that permit practical analysis were first developed in 1980 [2]. Since then, numerous approaches have been used to apply fundamental MVA concepts to practical cases [2–5]. However, the internal queue lengths at individual servers cannot be specified; rather, only the time-averaged sum of all must equal the allowed n. The direct effect of this difference in n on the results of a closed compared with an open model is extremely difficult to gauge for memory hierarchy networks of the type considered here. For instance, while an open-queue analysis must theoretically allow unlimited queue length, the actual queues are inherently small and self-limited by an internal negative feedback process present in a memory hierarchy. This negative feedback in an open-queue model occurs as described at the end of the section on open-queue calculation. A similar feedback occurs in MVA, but affects only the distribution of customers among the various queues, not the total number of customers in the queues.

It is not clear that fixing the total number of customers in the total system, as is done in MVA, is fundamentally a more accurate representation of the actual system than the open model. (A brief discussion of this issue is presented in Appendix C.) Unfortunately, there are no simple, direct methods of comparing open and closed systems, so we must rely on comparing the final results of these two techniques for cases in which multiple factors affect the final results, as will be seen.

Performance calculations

As previously indicated, it is desirable to calculate the FCP measured in CPI, which is the average extra delay

[1] Standard queuing theory nomenclature defines a queue as consisting of a line plus the server, so the total queue time is the time in line plus time to be serviced. We follow this standard queuing theory nomenclature. Unfortunately, common usage generally refers to the line alone as the queue, and papers and books sometimes mix the definitions, leading to confusion.

per instruction introduced by the finite delays of the memory system. Fundamentally, the calculation is simple: For an average memory access, calculate the average delay in terms of CPI encountered in the full memory hierarchy. When the memory hierarchy consists of several layers of caches, an average access to the memory will experience a certain percentage of hits and misses at each level. The FCP delay is simply the weighted sum of hits at each level multiplied by the delay per hit at that level. At some level, misses no longer occur, so the summation terminates. Details of FCP calculations for an open queue are given in [6], and the equation for FCP of a k-level memory hierarchy is shown to be[2]

$$FCP = \sum_{k=2}^{z} (mr_{k-1} - mr_k)T_{rk} = \sum_{k=2}^{z} hr_k T_{rk}\,. \tag{2}$$

Assuming no misses in main memory, a hierarchy having four levels below main memory would have

$$\begin{aligned} FCP &= (mr_1 - mr_2)T_{r2} + (mr_2 - mr_3)T_{r3} + (mr_3 - mr_4)T_{r4} \\ &\quad + mr_4 T_{rmain} \\ &= mr_1\left[\left(1 - \frac{mr_2}{mr_1}\right)T_{r2} + \left(\frac{mr_2 - mr_3}{mr_1}\right)T_{r3}\right. \\ &\quad \left. + \left(\frac{mr_3 - mr_4}{mr_1}\right)T_{r4} + \left(\frac{mr_4}{mr_1}\right)T_{rmain}\right], \end{aligned} \tag{3}$$

where

mr_1, mr_2 = miss rate of cache L1, L2, etc. in misses per instruction executed;
$hr_2 = (mr_1 - mr_2)$, $hr_3 = (mr_2 - mr_3)$, $hr_4 = (mr_3 - mr_4)$, $hr_{main} = mr_4$ are respectively the hit rates of levels 2, 3, 4, and main;
$T_{r2} = T_{L2}/T_{cpu}$, $T_{r3} = T_{L3}/T_{cpu}$, $T_{r4} = T_{L4}/T_{cpu}$, etc. are respectively the average residency or hit access delays in processor cycles per hit of level 2, 3, 4, etc., including queues and all other delays, with
T_{L2}, T_{L3}, T_{L4}, $\cdots$ = average reload delay for L2, L3, etc. in seconds per hit (or seconds per request); and
T_{cpu} = processor cycle time in seconds per cycle.

These delay terms must also include a probability of being visited, as is discussed in detail below.

The L1 cache access time for hits, whether one, two, or more cycles, is assumed to be included in the infinite-cache performance, $CPI[\infty]$. Equation (2) merely sums the percentage of all other hits at each level below L1, multiplied by the average total delay per hit at each level. The same expression is used for both open- and closed-queue models. However, the manner of calculating the queues and residency delay terms (T_{r2}, T_{r3}, T_{r4}, etc.) is quite different for the two cases, as will be shown.

Open-queue calculation

For a simple open-queue system, the queue at an individual server is determined by the utilization (U), which itself depends on the request rate (R_R) and service time (S_t) of each server. These two parameters alone allow us to determine the individual server utilization, a dimensionless fraction, from

$$U = R_R S_t\,, \tag{4}$$

where R_R is in units of requests per second and S_t is in units of seconds per request. This is the usual expression given in textbooks, but it has an implicit assumption, namely that the visitation factor is unity. A modification required for a memory hierarchy is considered shortly. For now, assume that the visitation factor is unity and Equation (4) is valid. The queue length and delay can be determined directly from U for various assumed server types (exponential, constant, general service time) as given respectively in rows one and three of **Table 1**. The individual queue delays can then be used for the delay terms (T_{r2}, T_{r3}, T_{r4}, etc.) in the average FCP delay of Equation (3), provided the utilizations are all much less than about 60% for constant service or general service time cases, while an exponential service time model is correct for all U. The sum of these open queues cannot be specified ahead of time and hence is unknown until calculated. This is unlike a closed model, for which the total sum of all queues must be some specified number.[3]

In an open-queue model, the memory hierarchy input request rate emanating from each L1 cache, per processor, is given by

$$R_R = \frac{mr_1}{CPI\, T_{cpu}} \text{ requests per second,} \tag{5}$$

where mr_1 is the L1 cache miss rate in misses per instruction executed,[4] CPI is the total processor cycles per instruction executed, including the degrading FCP effect of the memory hierarchy, and T_{cpu} is the cycle time of the processor in seconds per cycle.

For other cache levels of a general memory hierarchy, the request rates will have similar expressions, but the miss rates to the various levels will be different. For a

[2] The L1 cache is typically not included in the memory hierarchy analysis of a multiprocessor system because it is considered to be part of the processor. Its misses, expressed as the miss rate, mr_1, are the input process to the memory hierarchy of the processor.

[3] A caveat: It is not quite so simple for a closed model in which a delay center such as a CPU is necessary (see the section on the processor CPI in a closed MVA model).

[4] See [6] for a detailed discussion of open-model queuing analysis of both simple and complex memory hierarchies.

Table 1 Open-queue equations for delays and queue lengths

	M/M/1 *Poisson (exp) input* *Exponential service time*	*M/C/1 or M/D/1* *Poisson (exp) input* *Constant service time*	*M/E/1* *Poisson (exp) input* *Erlang service time*
Total, mean no. in Q (line + server)	$Q_M = \frac{U}{1-U}$	$Q_C = \frac{U - \frac{U^2}{2}}{1-U} = \frac{1}{2}\left(\frac{U}{1-U} + U\right)$ $= \frac{1}{2}(Q_M + U)$ $\approx \frac{U}{1-U}$ for small U	$Q_E = \frac{U}{1-U}\left[1 - \frac{U}{2}(1 - C_s^2)\right]$
Mean no. in line (line only)	$L_M = \frac{U^2}{1-U} = \frac{U}{1-U} - U$ $= Q_M - U$	$L_C = \frac{1}{2}\frac{U^2}{1-U} = (Q_C - U)$ $= \frac{1}{2}(Q_M - U)$	
Mean no. at server	$\frac{U}{1-U} - \frac{U^2}{1-U} = U$	U	
	Note: mean waiting line is shorter than total Q length by U, the number at server.		
Total queue delay (line + server) = mean residence time T_r	$T_{rM} = \frac{U}{1-U}\frac{S_t}{U}$ $= S_t\left(1 + \frac{U}{1-U}\right)$ $= S_t(1 + Q_M)$ $= S_t + T_{LM}$	$T_{rC} = \frac{\left(U - \frac{U^2}{2}\right)}{1-U}\frac{S_t}{U} = \frac{1}{2}\frac{US_t}{1-U} + S_t$ $= S_t\left(1 + \frac{1}{2}\frac{U}{1-U}\right)$ $= S_t + T_{LC}$	$\frac{S_t}{1-U}\left[1 - \frac{U}{2}(1 - C_s^2)\right]$ $C_s^2 = \frac{\sigma^2}{S_t^2}$ = 1 for exponential service time = 0 for constant service time σ^2 = variance of interarrival times
	In general, T [total] = T [line] + T [service].		
Delay, Q line only, T_L	$T_{LM} = \frac{U^2 S_t/U}{1-U} = S_t\frac{U}{1-U}$ $= S_t Q_M = T_{rM} - S_t$	$T_{LC} = \frac{1}{2}\frac{U^2 S_t/U}{1-U} = \frac{1}{2}\frac{US_t}{1-U}\left(\neq S_t \times Q_C \text{ except for } \frac{U^2}{2} \ll U\right)$ $= T_{rC} - S^t$	
Delay, server only	$\frac{S_t}{1-U} - \frac{US_t}{1-U}$ $= S_t = T_{rM} - T_{LM}$	$\frac{\left(1 - \frac{U}{2}\right)S_t}{1-U} - \frac{1}{2}\frac{US_t}{1-U}$ $= S_t = T_{rC} - T_{LC}$	
Utilization	$U = R_R S_t = \lambda S_t$ for all, where $\lambda = R_R$ = request rate for service (requests/time) and S_t = service time (time/request).		

Note: For exponential or constant service time, delay of waiting line only = (*line length*) × S_t/U = (*line length*)/R_R.
Delay of total queue (line + server) = (*total Q size*) × S_t/U = (*total Q size*)/R_R, where $S_t/U = 1/R_R$.
In all cases with Poisson input, U fixes size and delay of Q.

multiprocessor system with many shared resources, the utilization of each server can be composed of many different types of accesses and is most conveniently broken into the various components and summed, as detailed in [6].

From Table 1, for an open queue, the total queue delay times (residence) for service centers with exponential service time per request (M/M/1 queue) and constant service time per request (M/D/1 queue) are given respectively by

$$\begin{aligned} &\textit{Total delay (line + server) per request} \\ &= T_{\mathrm{rM}} = \frac{S_{\mathrm{t}}}{1-U} = S_{\mathrm{t}}\left(1+\frac{U}{1-U}\right) \\ &= S_{\mathrm{t}}(1+Q_{\mathrm{M}}) \qquad \text{exponential } S_{\mathrm{t}}, \end{aligned} \tag{6}$$

where $Q_{\mathrm{M}} = \dfrac{U}{1-U}$ (dimensionless)

and

$$\begin{aligned} &\textit{Total delay (line + server) per request} \\ &= T_{\mathrm{rC}} = \frac{\left(1-\frac{U}{2}\right)S_{\mathrm{t}}}{1-U} = \frac{1}{2}\frac{US_{\mathrm{t}}}{1-U} + S_{\mathrm{t}} \\ &= S_{\mathrm{t}}\left(1+\frac{1}{2}\frac{U}{1-U}\right) \qquad \text{constant } S_{\mathrm{t}} \end{aligned} \tag{7}$$

in units of seconds per request, where U is given by Equation (4) and R_{R} is given by Equation (5). Thus, for open-queue models we can calculate the delay of each server on the basis of given input parameters. While the calculation for each server in a complex multiprocessor system can be rather tedious [6], the fundamentals are quite trivial for a simple system. For instance, consider a uniprocessor system consisting of one CPU, an L1 cache, and a main memory. All misses in L1 are reloaded from main memory. Assume that the L1 miss rate is mr_1 misses per instruction executed, the CPU execution rate with infinite cache (i.e., no misses) is $CPI[\infty]$, processor cycle time is T_{cpu} seconds per cycle, and the average reload (service) time from main back to L1 is T_{m} seconds per request or $T_{\mathrm{m}}/T_{\mathrm{cpu}}$ processor cycles per request. For this case, the server service time is $S_{\mathrm{t}} = T_{\mathrm{m}}$ seconds per request. The memory utilization, from Equation (4), is

$$U = R_{\mathrm{R}}S_{\mathrm{t}} = \frac{mr_1}{CPI\,T_{\mathrm{cpu}}}T_{\mathrm{m}}. \tag{8}$$

The above utilization expression is valid for the simple case of all L1 reloads satisfied in main memory, because all L1 requests propagate to main memory with a probability of 1. In a memory hierarchy in which misses occur at various levels, the probability of requests reaching any given server will vary as determined by the given miss rates for each component. The utilization must include these probabilities because if the probability of all requests reaching any given server is 0, that server obviously has an average utilization of 0, regardless of all other parameters. Each traffic component that accesses a server must include the probability of each request visiting (i.e., hitting) each of the downstream caches. This is often expressed as the visitation probability, which is simply the hit probability per L1 miss request. For the case of a uniprocessor with tandem caches, this visitation factor at each downstream level can easily be expressed as

$$\begin{aligned} &\textit{Visitation probability}[k]\; V_k = (L_k \textit{ hits}/I)/(\textit{L1 misses}/I) \\ &= \frac{mr_{k-1} - mr_k}{mr_1} \end{aligned} \tag{9}$$

in units of fractional percent (dimensionless).

The denominator is always mr_1 because the probability is measured per L1 miss. Note that the numerator is always the hit rate at level k, and this numerator term occurs repeatedly in Equations (2) and (3) for calculating the FCP. The visitation probability at each level is used as a multiplying or weighting factor for the utilization of each level in open queues or the residency time in MVA, as discussed later in this paper. For open queues, the visitation probability is easily included in the utilization by modifying Equation (4) to become

$$U = V_k R_{\mathrm{R}} S_{tk} = V_k \frac{mr_1}{CPI\,T_{\mathrm{cpu}}} S_{tk}, \tag{10}$$

where S_{tk} is the service time of server k in seconds per request.

The total queue delay time for level k with constant service time is obtained by substituting Equation (10) into Equation (7) to obtain

$$\begin{aligned} T_{rk} &= S_{tk} + \frac{1}{2}S_{tk}\frac{U}{1-U} \\ &= \frac{1}{2}S_{tk}^2\frac{V_k\,mr_1}{CPI\,T_{\mathrm{cpu}} - V_k\,mr_1 S_{tk}} + S_{tk} \text{ seconds per access.} \end{aligned} \tag{11}$$

Once the various total queue delay times are determined for each level, the final FCP, as in Equation (3), is given by

$$\begin{aligned} FCP = mr_1(&T_{\mathrm{r1}} + T_{\mathrm{r2}} + \cdots \\ &+ T_{rk\max})/T_{\mathrm{cpu}} \text{ cycles per instruction.} \end{aligned} \tag{12}$$

The given value of $CPI[\infty]$ is added to the FCP above to obtain the final system CPI as in Equation (1).

For our previous simple system with L1 misses reloaded directly from main memory with a service time of T_{m}

seconds per request, there is only one delay term, and the visitation probability is 1. Equation (1) can easily be solved directly by substituting Equations (11) and (12) with $S_{tk} = T_m$ into Equation (1) to obtain

$$CPI + \frac{mr_1}{T_{cpu}}\left(0.5T_m^2 \frac{mr_1}{CPI\,T_{cpu} - mr_1 T_m} + T_m\right), \quad (13)$$

where $CPI[\infty] = CPI[\infty]$ is the given value of CPI at infinite L1 cache (i.e., no L1 misses), and CPI is to be calculated. All other parameters are given. Obviously, we could solve for CPI directly by several methods. For instance, we could obtain a quadratic equation in CPI by multiplying through by the denominator term above, and collect terms giving

$$CPI^2 - CPI\left(2mr_1 \frac{T_m}{T_{cpu}} + CPI[\infty]\right) + \left(mr_1 \frac{T_m}{T_{cpu}} CPI[\infty] + 0.5mr_1^2 \frac{T_m^2}{T_{cpu}^2}\right) = 0, \quad (14)$$

which has the standard quadratic form $ax^2 + bx + c = 0$, with $x = CPI$.

For this case, a relatively simple quadratic equation represents the solution. In general, the CPI equation will be the sum of all queue delay terms in the network. There will be as many such delays as there are queues in the system, with each queue adding another term to Equation (13) and each having a denominator term involving CPI and its service time, similar to that above. A single polynomial can be obtained by multiplying through by all of the denominator terms. As a result, the order of the single CPI equation to be solved is equal to the number of servers plus 1. A system with eight servers would produce a polynomial of order 9, 20 servers produces a polynomial of order 21, etc.

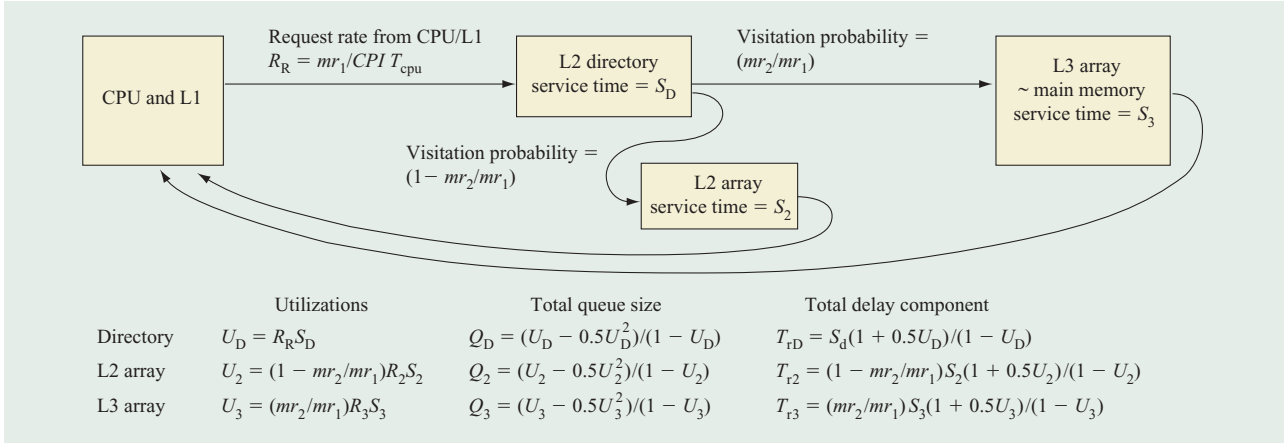
For complex systems with many servers and queues, the determination of all utilization components and final queues from a direct solution to such a polynomial can be quite messy, tedious, and prone to error. While a more direct solution to such a large polynomial is possible, it is expedient to leave all of the individual equations for queues and other delays expressed in a spreadsheet or other convenient equation solver. These equations can then easily be solved by using the recalculation or equation-solving capabilities of the methodology. Any recalculation method, such as those provided in spreadsheets, that allows all component equations to be expressed in natural form is very useful. In any case, the solution for an open system follows a simple algorithm in principle.

Note that R_R in Equation (5) is proportional to 1 over the total CPI [$= CPI[\infty] + FCP$], which makes the analysis nonlinear. It is also the connection that provides the negative feedback to keep the queues from becoming large. This feedback occurs as follows.

All queue delays increase as the request rates for reloads (i.e., misses per second) increase. The request rates, in turn, are inversely proportional to CPI—the smaller the CPI, the faster instructions are processed, thereby generating more memory misses and therefore more reload requests per second. But any increase in memory requests per second (due to a decrease in CPI) creates larger queue delays, which then increase FCP, which increases CPI, which makes the queue delays smaller, which decreases CPI, etc. This then reduces the number of requests per second (number of customers in the system), and a balance is eventually achieved. In other words, any decrease in CPI will increase the queue delays, but any increase in queue delay will increase the CPI, which then decreases the queue delay, until a balance is reached. Thus, there is an inherent self-limit to the total number of customers in the system, but we cannot specify it; we can only determine its value after the final CPI has been determined.

Open-queue example of memory hierarchy with L2 directory, L2 array, and L3 array

The calculation of the utilizations, queue delays, FCP, and final CPI of an open-queue memory-hierarchy network is relatively straightforward using the above relationships. Memory hierarchies typically contain translation directories, but they may or may not have to be included in the delay path, depending on the organization of the cache level. A late-select organization in which the cache directory is accessed at the same time in parallel with the cache array will see only the delay of the slowest path, typically the array. However, a sequential organization, in which the directory is accessed first before accessing any array, will see the additional directory delay component. Our model assumes a sequential organization for the L2 level and ignores any L3 directory delay components. This is shown in **Figure 1**, which consists of a second-level cache, L2 with its associated directory, and a third-level cache array, L3. It is assumed that there are no misses in L3, only mr_2 misses per instruction in L2 and, of course, mr_1 misses per instruction in L1. With a sequential L2 directory organization, all miss requests arriving from L1 first access the L2 directory. Subsequently, only L2 directory hits—namely $mr_1(1 - mr_2/mr_1) = (mr_1 - mr_2)$ hits per instruction—access the L2 array and have a return path to the input, as indicated. All L2 directory misses, namely mr_2 misses per instruction, access the L3 array. All L3 accesses are hits, and hence have a return path to the input, as shown. Thus, the respective visitation probabilities for the directory, L2 array, and L3 array from Equation (9) are



	Utilizations	Total queue size	Total delay component
Directory	$U_D = R_R S_D$	$Q_D = (U_D - 0.5U_D^2)/(1 - U_D)$	$T_{rD} = S_d(1 + 0.5U_D)/(1 - U_D)$
L2 array	$U_2 = (1 - mr_2/mr_1)R_2S_2$	$Q_2 = (U_2 - 0.5U_2^2)/(1 - U_2)$	$T_{r2} = (1 - mr_2/mr_1)S_2(1 + 0.5U_2)/(1 - U_2)$
L3 array	$U_3 = (mr_2/mr_1)R_3S_3$	$Q_3 = (U_3 - 0.5U_3^2)/(1 - U_3)$	$T_{r3} = (mr_2/mr_1)S_3(1 + 0.5U_3)/(1 - U_3)$

Figure 1

Open-queue example of memory hierarchy with L2 directory, L2 array, and L3 array.

$$V_d = 1, \quad V_2 = \left(1 - \frac{mr_2}{mr_1}\right), \quad \text{and} \quad V_3 = \frac{mr_2}{mr_1}. \tag{15}$$

Clearly, if mr_2 is 0, the L2 visitation probability, V_2, is unity, the L3 visitation probability, V_3, is 0, and all accesses are directed to the L2 array as if L3 were not present. Similarly, if $mr_2 = mr_1$, all accesses are directed to L3 as if L2 were not present.

The calculation of the utilization of each server using the above probabilities, the total queue and queue delay for each, the final FCP, and CPI is illustrated in detail in **Figure 2** (in Appendix D) for the case of

Service times: L2 directory, $S_t = 2$ cycles per access; L2 array, $S_2 = 12$ cycles per access; and L3 array, $S_3 = 140$ cycles per access;

$mr_1 = 0.01$, $mr_2 = 0.001$ misses per instruction;

$CPI[\infty] = 1.2$ cycles per instruction;

and

$T_{cpu} = 1$ ns per cycle.

For these parameters, the sum of all queues, shown in column K of Figure 2, is 0.56 customers. This number, of course, increases as the miss rates increase, but cannot exceed 1. The FCP in column L is 1.53 and is larger than the assumed CPI at infinite cache of 1.2. Thus, this design with the given parameters loses more than 50% of the ideal processor computing power. The utilizations (columns B, C, and D) for the three servers show that the L3 is most heavily used, and the total delay times (columns H, I, and J) similarly show that L3 has the largest delay component. Thus, improvements to the system would concentrate first on reducing L3 access time or improving the L2 miss rate via a larger L2. Of course, a larger L1 with a correspondingly smaller miss rate would also improve the FCP, but this usually is not an option. Nevertheless, the queuing analysis can reveal potential bottlenecks and show where improvements are possible.

Closed-queue network (MVA)

For a closed-queue model (MVA) the FCP is evaluated similarly to that for an open system, but the queue delays or, more commonly, the *server residency time*, must be evaluated for the boundary condition that the total time-averaged number of customers (n) in the system is fixed. This is quite different from an open queue where we do not, and cannot, know the number of customers in the system until the full set of queue equations are solved—and then we can find the number of customers only by adding all of the queues in the system, as was done in the previous, open-model example. For a closed queue, it should be apparent that we must express the request rate in terms of n. This can be done by using the following relationships, which represent the essential equations of MVA:

1. Forced-flow law applied to the entire system, input to output,

   $$R_R = n/\Sigma T_{rk} \text{ requests per second.} \tag{16}$$

2. Little's law applied to individual servers in tandem,

   $$Q_k = R_R T_{rk}. \tag{17}$$

3. Total residency time in queue (line + server),[5]

[5] From Lazowska et al. [4, p. 112], Q_k is the average number of customers seen at the server node k at the instant a new customer arrives.

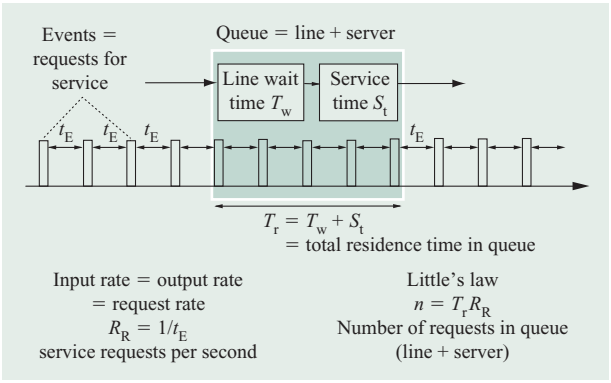


Figure 3

Graphical representation of Little's law.

$$T_{rk} = S_t[1 + Q_k(n - 1)] \text{ (queuing centers)}, \quad (18)$$

and

$$T_{rk} = S_t \text{ (delay centers, i.e., no queue)}, \quad (19)$$

where

Q_k, $Q_k(n - 1)$ = the total queue length at server k, including the queue line and server, when there are n and $n - 1$ customers, respectively, in the system;
T_{rk} = total residency time per request in server k for both queue line and service (in seconds per request);
R_R = throughput = input event request rate = output rate at steady state in requests per second;
S_t = service time per request of the server alone, not including the queue (in seconds per request); and
n = total number of customers (requests) in the system.

Note that when we solve Equations (16) and (17) for n, we obtain

$$\Sigma Q_k = n, \quad (20)$$

which is basically an input specification for a closed network, namely that the sum of queues must be fixed and specified. However, the actual value of any individual queue cannot be specified; only the sum may be specified, as indicated.

Equation (17) for Little's law can be understood conceptually from **Figure 3**.[6] On average, events or requests for service occur at the rate of R_R requests per second as indicated. Within any window of width T_r, there must be $T_r R_R$ such events, which is the total queue as specified above. This relationship must clearly be true for any system in steady state, because otherwise the queues will grow to infinity or diminish to zero if the input rate is respectively greater than or less than the output rate.

The entire analysis is based on the assumption of forced flow. This simply means that in steady state, output rate equals input rate at all tandem queues in the system. However, instantaneously, we can have input rate different from output rate, so queues can increase or decrease until the steady-state time-averaged mean value is reached. The queue calculations are contingent on steady-state behavior. Thus, at steady state we have $R_R[\text{in}] = R_R[\text{out}] = R_R$ at each serial server.

From Table 1, column 1, the total queue delay for an open, M/M/1 queue (exponential service time) is given by

$$T_k = S_t[1 + Q_k(n)]. \quad (21)$$

This expression indicates that *for an open system, the time-averaged queue lengths $Q_k(n)$ can be used for the arrival instant queue length* in evaluating residency time. A similar expression can be used for a closed queue, provided we use the queue length seen at arrival when there are a total of n customers in the system. The following argument is used in MVA closed queues to achieve this, and is crucial. Because there are always n customers in the system, when a new customer arrives, one must leave at the same instant, on average. Thus, the queue length seen by the arriving customer will be the same as the time-averaged queue that would exit there with *one less customer in the network*; i.e., *the incoming customer sees a mean delay of only $n - 1$ customers, not a mean delay of n.* Thus, the equation for T_{rk} *of each queue* must use the Q_k length evaluated with $n - 1$ customers in the system, as expressed in Equation (18). This assertion relates strictly to steady-state time-averaged performance. (On an instantaneous basis, there can be a transient buildup of queues.) The resulting throughput or request rate from substituting Equation (18) into Equation (16) is thus

$$R_R[n] = n/\Sigma S_t(1 + Q[n - 1]). \quad (22)$$

In a closed MVA system, the total number of customers in the system is fixed, so the value of n is given. It is necessary to calculate the value of the input request rate (R_R) which will give this number of customers (n). This can be obtained from Equation (16), but requires the sum of all individual residency times (T_{rk}). The latter can be obtained from Equation (18) if we know the individual queue values with $n - 1$ customers in the total network. But we do not know the value of each queue for $n - 1$ customers in the system. However, we can start an *iterative solution* from $n = 1$, where all queues at $n - 1$, namely $Q[n - 1]$, are known and equal to 0, which allows us to calculate R_R at $n = 1$, then queues $Q[n]$ for $n = 1$ which are valid for finding R_R at $n = 2$, etc., up to $n =$ the

[6] This is not a proof; it is only a conceptual way to understand and remember the relationship. The proof must include the dynamics of the queues changing as new events occur [4, p. 42].

maximum number of customers in the system. An example is given in the following section.

MVA solution

***Exact solution: Iterations on* n**

For a given system of servers, each having a mean service delay time of S_t and a total of n customers in the system, the throughput (the value of R_R) can be found by an iterative process as follows. The problem is to determine the value of residency time (T_{rk}) for all servers with all customers present from which R_R can be found using Equation (16). However, we need each queue length (Q_k) to find T_{rk}; see Equation (18). But we do not know the value of Q_k even at $n - 1$. Thus, we start at a point where we do know $Q_k[n - 1]$, namely, one customer in the system, which gives the first value of T_{rk}. We then increment to the solution from there by continually adding one more customer, as follows.

- *Step A*: Start with $n = 1$, or $n - 1 = 0$, so there are no customers in the system when the first one arrives. For such a case, all queue lengths ($Q[n - 1]$) must obviously be 0, so we can specify the first iteration values for each residence time (T_k). It is obvious from Equation (18) that

  $$T_{rk} = S_{tk} \tag{23}$$

  for each server, k, at $n = 1$. These values of T_{rk} are used in Equation (16) to obtain the first iteration of request rate as

  $$R_R = 1/\Sigma S_{tk} = 1/(S_{t1} + S_{t2} + S_{t3} + \cdots). \tag{24}$$

  Now that we have a first value for R_R and T_{rk}, these can be used to evaluate the first iteration of each $Q[n]$ from Equation (17) to obtain

  $$Q_1 = T_{r1}/(S_{t1} + S_{t2} + S_{t3}),$$
  $$Q_2 = T_{r2}/(S_{t1} + S_{t2} + S_{t3}), \tag{25}$$
  $$Q_3 = T_{r3}/(S_{t1} + S_{t2} + S_{t3}), \text{ etc.,}$$

  where $T_{r1} = S_{t1}$, $T_{r2} = S_{t2}$, and $T_{r3} = S_{t3}$ (for this iteration only).

- *Step B:* Next, increment to $n + 1 = 2$ customers in the system and reevaluate all T_{rk} using Equation (18), noting that the queue lengths ($Q_k[n - 1]$) for this iteration are those evaluated in Step A above at $n = 1$. These new values for T_{rk} are used in Equation (16) to find R_R for $n = 2$. New values for all queue lengths are also evaluated using these values of T_{rk} and R_R in Equation (17). These Q_k evaluated at $n = 2$ will be the lengths used in Equation (18) to evaluate T_{rk} at the next iteration of n.
- *Step C*: Increment to $n + 1 = 3$ and reevaluate all T_{rk}, R_R, and Q_k as above.

This process is repeated until $n =$ the desired number of customers, giving the final value of R_R as the desired throughput.

An example from a spreadsheet evaluation is illustrated in **Figure 4** (see Appendix D) for four servers with service times of $S_{t1} = 2$ s, $S_{t2} = 4$ s, $S_{t3} = 8$ s, and $S_{t4} = 20$ s.

The progression of the iteration can easily be followed with hand calculations, and a spreadsheet allows many configurations to be quickly analyzed. It can be seen in column K that the sum of the four individual queues always equals n, the number of customers in the system (column A), as it should. The lengths of the shorter queues, Q1, Q2, and Q3, reach a constant, steady-state value very quickly, at $n \approx 4$–6, respectively, and additional customers are added to the end of the longest queue, Q4, as n increases further.

***Approximate solution: No iterations on* n**

The above MVA process of starting from $n = 1$ and incrementing up to large n can be avoided by use of the Schweitzer approximation [7], which is

$$Q[n - 1] = [(n - 1)/n]\,Q[n], \tag{26}$$

with accuracy increasing as n increases. The resulting nonlinear problem still requires a recalculation method similar to that required for open queues, but is computationally simpler in large networks and can be solved for any one given n.

The above analysis is sufficient for many types of models and represents the general procedure given in standard books on MVA. In dealing with memory hierarchies, the queue analysis requires additional analytical constructs, presented below.

Processor CPI in closed MVA model: CPU as delay center

As seen above for the simple MVA examples, n customers visit k servers, each with a queue. The request rate was the same for all, and we easily calculated the queue for each server. In a memory hierarchy model, the situation is somewhat different. The processor is not a server in the usual sense and does not have a queue for memory requests. The essential question is, *How do we include the processor CPI and simultaneously the miss rates of the various cache levels?* In an open queue, we saw that the CPU/L1 was just a requestor, issuing requests to the hierarchy at a rate proportional to mr_1/CPI as in Equation (5). The miss rates for L2, L3, etc. determine the hits for

each delay component of the FCP as in Equation (3). In a closed-queue model, the miss rates for the various cache levels are still introduced as in Equation (3), but the request rate must be calculated as in Equation (16). This is accomplished by treating the CPU not as a direct requestor, but as a delay center [4, p. 115]. The n customers (i.e., reload requests) visit the various cache components, and reloads are sent back to the CPU. The CPU subsequently sends these reload requests back to the memory hierarchy after being delayed, with the delay given by

$$CPU\,[delay] = T_C = CPI\,T_{cpu}/mr_1 \text{ seconds per miss request.} \tag{27}$$

This is the CPU residency time, and it provides the feedback coupling between the cache queue delays and FCP. This delay time is the CPU delay component in calculating the request rate as in Equation (16), but it does not appear directly in the FCP calculation because the CPU does not have a queue and does not contribute a delay component to the FCP. Only the L2, L3, etc. caches have queues as part of their residency times, and only the cache delays contribute to the FCP. In essence, the delay time of the CPU is the time the CPU must spend executing other, non-memory instructions until the next miss or reload request is encountered. Obviously, the CPU delay time should depend on the L1 miss rate in such a manner that, as mr_1 goes to 0, the delay goes to infinity; i.e., there is an infinite time interval until the next reload request is issued. This is seen to be the case in Equation (27). The use of the CPU as a delay center avoids the need to include the processing of non-memory instructions and introduces the CPI and mr_1 as desired.

It is interesting to note that the CPU delay as a delay center in the MVA model (and in the subsequent model) is exactly the reciprocal of the request rate used in the open model, Equation (5). The only difference between these two models is the method of calculating the queues and residency time delays. Even the visitation probabilities are identical, as discussed next.

Residency time and visitation probability

MVA requires the determination of the residency time as given by Equation (18). This residency time is essentially the mean time spent in the queue per request. In a memory hierarchy, any given reload request from the CPU/L1 can result in a request to any of the downstream caches with a certain probability given by the various miss rates. The probability of this request visiting (i.e., hitting) each of the downstream caches must be included, as is done in calculating the utilization factor for servers in open-queue models. This is the same visitation probability as given by Equation (9) and is used as a multiplying or weighting factor for the residency time of each level. Thus, we can express the MVA residency time of Equation (18) more generally as

$$T_{rk} = V_k S_{tk}(1 + Q_k[n-1]) \text{ seconds per request.} \tag{28}$$

Once the residency time of each level has been determined from the above, the final FCP is calculated identically to that for an open queue as in Equation (3). As previously, the given value of $CPI[\infty]$ is added to the FCP to obtain the final system CPI, as in Equation (1).

MVA example with CPU/L1, L2 directory and array, and L3 array

This example is the memory hierarchy model shown in Figure 1, which was analyzed as an open-queue network. The same model is analyzed using the MVA techniques described above. The MVA model is shown in **Figure 5**, where it is again assumed that all miss requests arriving from L1 first access the L2 directory. Subsequently, only L2 directory hits access the L2 array and have a return path to the input as indicated. All L2 directory misses, namely mr_2 misses per instruction, access the L3 array, and all are hits with a return path as indicated. The CPU does not have a queue, but rather is a delay center, as discussed above.

The request rate according to Little's law [Equation (16)] is similar to the previous MVA case of simple servers without a CPU and misses, except that the total delay (denominator) is the sum of the CPU delay plus the L2 (directory and array) and L3 residence times (line + server). The L2 and L3 residence times are determined as in Equation (18), where the queuing time is once again that seen with $n - 1$ customers in the system. However, the CPU residency delay time is given by Equation (19) for a delay center, which is equal to Equation (27).

At steady state, the input request rate, R_R, must equal the output service rate as shown. The servers L2 and L3 are in parallel because only one or the other is accessed on any request from the L2 directory. Thus, the visitation probabilities for these components, from Equation (9), are identical to those for the open queue model:

$$V_d = 1, \quad V_2 = \left(1 - \frac{mr_2}{mr_1}\right), \quad V_3 = \frac{mr_2}{mr_1}. \tag{29}$$

The residency times for the L2 directory, L2 array, and L3 array of a server are calculated in a manner identical to that done previously, with the iteration starting at $n = 1$, for which all queues at $(n - 1)$ are 0, etc., using the relations

$$T_{rd} = V_d S_{td}(1 + Q_d[n-1]) = S_{td}(1 + Q_d[n-1]), \tag{30}$$

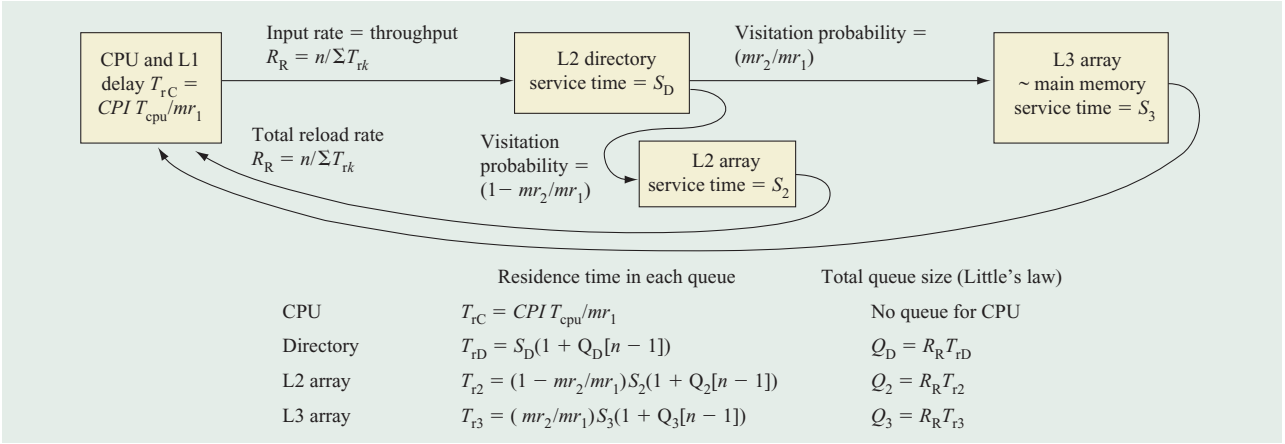


Figure 5

MVA model of memory hierarchy with L2 directory, L2 array, and L3 array, with CPU as delay center.

$$T_{r2} = V_2 S_{t2}(1 + Q_2[n - 1]) = (1 - mr_2/mr_1)S_{t2}(1 + Q_2[n - 1]), \quad (31)$$

and

$$T_{r3} = V_3 S_{t3}(1 + Q_3[n - 1]) = mr_2/mr_1 S_{t3}(1 + Q_3[n - 1]). \quad (32)$$

The CPU delay time is calculated as given previously by Equation (27), namely

$$T_C = CPI\ T_{cpu}/mr_1, \quad (33)$$

$$R_R = n/\Sigma(T_{rk} + T_C) = n/(T_{rd} + T_{r2} + T_{r3} + T_C), \quad (34)$$

$$FCP = mr_1(T_{rd} + T_{r2} + T_{r3}), \quad (35)$$

$$FCP = S_{td}(1 + Q_d[n - 1]) + (mr_1 - mr_2)S_{t2}(1 + Q_2[n - 1]) + mr_2 S_{t3}(1 + Q_3[n - 1]), \quad (36)$$

and

$$CPI = CPI[\infty] + FCP. \quad (37)$$

The new individual queue lengths are determined as before from Equation (17) as

$$Q_d = T_{rd}R_R, \quad Q_2 = T_{r2}R_R, \quad Q_3 = T_{r3}R_R. \quad (38)$$

These queues are then used to evaluate the individual residency times on the next iteration of n, next R_R, FCP, new Q values, etc., until the maximum value of n is reached.

A spreadsheet example of this iterative calculation is given in **Figure 6** (see Appendix D) for assumed parameters (same as the previous open-queue example):

Service times: L2 directory, $S_t = 2$ cycles per access; L2 array, $S_2 = 12$ cycles per access; and L3 array, $S_3 = 140$ cycles per access;

$mr_1 = 0.01$, $mr_2 = 0.001$ misses per instruction;

$CPI[\infty] = 1.2$ cycles per instruction;

and

$T_{cpu} = 1$ ns per cycle.

It can be seen by comparing columns A and N that the number of customers in the memory hierarchy queues is smaller than n and smaller than the number of customers delayed in the CPU, as given in column J. Comparison of this MVA model with the open model of Figure 2 shows that the FCP of the open model, namely 1.53, is nearly the same as the MVA value of 1.54 for $n = 2$. However, the sum of customers in the memory queues is 0.56 for the open model and 0.72 for MVA. If we linearly interpolate the MVA value to a number of customers in memory queues equal to 0.56, as for the open model, the MVA value of FCP becomes 1.45, which is about 5% smaller. Thus, they give nearly the same performance estimate. The MVA model, of course, shows how the FCP will increase as the number of customers increases. However, this is not particularly meaningful because we cannot get large numbers of customers in a blocking-cache model, only in a nonblocking case. For a multiprocessor system or for a multi-issue processor with nonblocking memory, the average FCP should initially decrease rather than increase as n increases because some of the memory access times are overlapped with continued processing time. This model cannot handle such nonblocking systems (see Appendix C). Various other types of MVA calculations can also be performed, including coherency protocol and other aspects of the system [8–10].

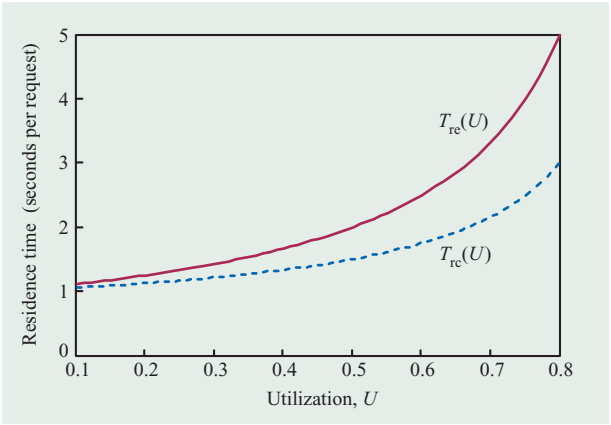


Figure 7

Total queue residency time, T_{re} and T_{rc}, in seconds per request for single M/M/1 (exponential service time) and M/C/1 (constant service time) servers as a function of utilization, U.

Open- and closed-queue approximations

Some assumptions were inherently necessary in order to be able to represent the queue calculations above in simple analytical form; these should be understood, because they may not be valid for a given real system. Some of these basic assumptions are as follows.

An open-queuing network—the only known system for which useful queuing equations can be obtained—requires the assumption of a Poisson customer arrival process for the miss requests. For tandem queues, this requirement further implies that the servers must also have exponential service time, because this is the only service time that gives a Poisson process output to the next server in tandem. These assumptions simply require that the arrival process and the server be memoryless in the statistical sense (for an explanation of the term *memoryless*, see Appendix B). This is referred to as an M/M/1 queue, meaning *memoryless input process/memoryless service time/one server*. The queues and delays for such a case can be expressed as shown in the first column of Table 1.

A single open queue with constant service time can also be treated analytically. Such a server is typically referred to as M/C/1 or M/D/1 and, assuming a Poisson input process, it obeys the queue and delay expressions given in the second column of Table 1. However, for a series of tandem queues with constant service times, with the original input equal to a Poisson process, the separate inputs at each individual server will not be a Poisson process. For a network of tandem queues, this requirement can be satisfied at any given server only if the preceding server has exponential service time. For an open network of k servers in tandem, only the first $k - 1$ must have exponential service time, and the last server can be general service time. For such a case, the first $k - 1$ servers have Poisson departures and feed into the subsequent queue so that the input conditions are satisfied. At the last queue, which has Poisson arrivals, it does not matter what the departure process is because its output is not used.

For closed queues, it is necessary that all servers have exponential service times. This is inherent in the product-form solution, which is the starting point for MVA and requires the assumption of exponential servers for an exact solution. In a closed system of tandem queues, if all servers have exponential service times, all inputs will be a Poisson process and each server can be treated as an M/M/1 type. Thus, the situation with service time is similar to that for an open queue.

While memory hierarchies can have an input miss stream that is approximately a Poisson process,[7] service times for the various memory levels are typically constant, not exponential. The question arises, *How good is the use of exponential service time approximation for MVA and constant service time with Poisson input (input from exponential service time server) for open-queue delays?*

Servers with exponential service time and constant service time have a total residency time given respectively by the first and second column, fourth row of Table 1, or

$$T_{re} = S_t\left(1 + \frac{U}{1-U}\right) \quad \text{and} \quad T_{rc} = S_t\left(1 + \frac{1}{2}\frac{U}{1-U}\right). \qquad (39)$$

These two equations are plotted in **Figure 7** as a function of utilization, U. From extensive experience, it is well known that systems for which the utilization of any server is much above 40–50% is a marginal design; i.e., any server with high utilization is a major bottleneck in system performance. Thus, server utilizations should always be maintained below these values. It can be seen from Figure 7 that for small utilizations [less than about 0.4 (40%)], the residency time for constant service time is about 20% smaller than that for exponential service time. This is intuitively correct because, for small U, the additional queuing delays are negligible in all cases. Hence, equations for exponential service times can be used as reasonable engineering approximations for constant service times, provided U is small. As U increases above 40%, the error increases substantially, as can be seen. The U values for each server must include the visitation probability and, for MVA, can be determined for each value of n only after the request rate has been calculated. The validity of the approximation can then be

[7] This has been explored with a number of traces from SPEC** benchmarks. Some have been found to have approximately Poisson (exponential) interarrival time distributions and some are far from a true exponential, but all have some gross similarity to a $1 - e^{-x}$ function.

	A	B	C	D	E	F	G	H
1		L2 miss ratio =						
2		V3 = m2/mr1 =	0.1			Utilizations:		
3						with Vn probability		
4	# in system	FCP=	CPI =	Sum of Qs		Utilization	Utilization	Utilization
5						of L2 dir	of L2	of L3
6	n	mr1(TrD+Tr2+Tr3)	FCP + CPI[inf}	QD+Q2+Q3		RR2Sd	V2*RR*S2	V3*RR*S3
7								
8	1	1.34	2.54	0.35		0.026	0.139	0.180
9	2	1.54	2.74	0.72		0.047	0.252	0.326
10	3	1.77	2.97	1.12		0.063	0.342	0.443
11	4	2.01	3.21	1.54		0.077	0.413	0.536
12	5	2.27	3.47	1.98		0.087	0.470	0.609
13	6	2.55	3.75	2.43		0.095	0.515	0.667
14	7	2.83	4.03	2.89		0.102	0.550	0.714
15	8	3.13	4.33	3.36		0.107	0.580	0.751
16	9	3.43	4.63	3.83		0.112	0.603	0.782
17	10	3.73	4.93	4.31		0.115	0.623	0.807
18	11	4.05	5.25	4.79		0.118	0.639	0.829
19	12	4.36	5.56	5.27		0.121	0.653	0.846
20	13	4.68	5.88	5.76		0.123	0.664	0.861
21	14	5.01	6.21	6.25		0.125	0.674	0.874
22	15	5.33	6.53	6.74		0.126	0.683	0.885
23	16	5.66	6.86	7.23		0.128	0.690	0.895
24	17	5.99	7.19	7.73		0.129	0.697	0.903
25	18	6.32	7.52	8.22		0.130	0.702	0.910
26	19	6.65	7.85	8.71		0.131	0.707	0.917
27	20	6.99	8.19	9.21		0.132	0.712	0.922

Figure 8

Server utilizations for previous MVA memory hierarchy with L2 directory and array, L3 array, and CPU as a delay center.

determined. For most cases of interest, the utilizations are reasonable, and the approximation is acceptable. For instance, the utilizations for the three servers of the previous open model of Figure 1—L2 directory, L2 array, and L3 array—are shown in columns B, C, and D of Figure 2. It can be seen that they are quite reasonably small, with the maximum of 0.256 for L3.

Similarly, the utilizations for the same three servers of the previous MVA model of Figure 5 are shown in **Figure 8**. For small values of n (≤ 3), the utilizations are indeed small. Values of n greater than 3 for a single processor are not typical, so the approximation is acceptable. However, as n increases above 3, the utilization of L3 and, to a lesser extent, that of the L2 array increase substantially above 0.4, so the approximation becomes less accurate. There is no simple rigorous method for calculating such cases. For constant-service-time servers, one heuristic approximation for the mean residual service time (the remaining service time of a customer currently in service when a new customer arrives) is

$$\textit{Mean residual service time} = S_t/2, \tag{40}$$

where S_t = mean service time [11]. This approximation requires random arrival time, but the arrival time in a closed network will not be random if a customer spends little time (relative to the constant service time) before coming back to the server. In such cases, the mean residual service time is closer to the service time S_t.[8] Intuitively, this seems correct because, for constant service time, the maximum mean residual service time is S_t. For all cases studied here, the utilizations are small enough that the exponential server approximations are acceptably accurate for our purposes. The approximations are that the equations for a single server with constant service time can be used for open queues in tandem, and queue residency delay for a single server with exponential service time can be used for MVA tandem queues as long as all server utilizations—and hence the queues—are small.

[8] From the work of Men Chow Chiang, IBM Austin, private communication, memo June 18, 2001.

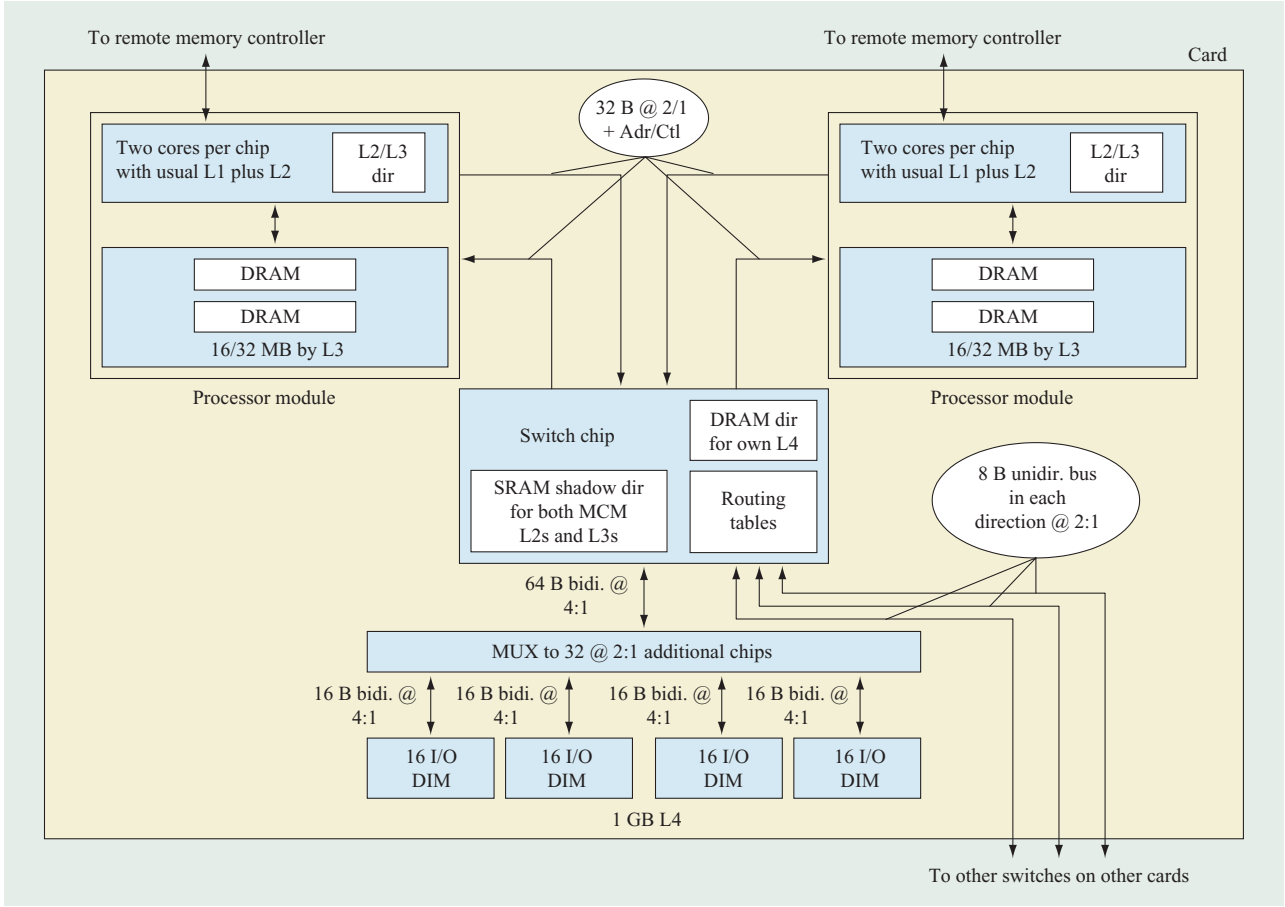


Figure 9

Organization of basic four-way card having large L3 on the processor module and large L4 on card.

Comparison of MVA closed- and standard open-queue models of a moderately complex 16-way multiprocessor memory hierarchy

To obtain a definitive comparison of closed- and open-queue model performance results, a moderately complex 16-way multiprocessor system that had previously been modeled as a closed MVA system was chosen. The model is based on the general MVA constructs described above and was constructed using an IBM modeling system known as EZMVA.[9] For comparison, an open model using a standard spreadsheet[10] as the vehicle was constructed in detail by the author, with very particular attention to using the same parameters. A 16-way system was chosen that was constructed from four cards. Each card, as shown in **Figure 9**, contains two processor modules, a switch-chip module, and an L4. Each processor module contains one processor chip and an on-module L3, as indicated. Each processor chip contains two processors, each with its own L1, plus one on-chip L2. The two processor modules are connected to the switch chip by means of unidirectional buses. Interconnections to the on-card L4 are provided by bidirectional buses that connect through the switch chip as shown. Thus, each card contains four processors, two private L2s, two private L3s, and one private L4. In principle, cross-interrogates should be performed on all L2, L3, and L4 misses. However, cross-interrogates at the L3 and L4 level are too small to have any significant impact on performance and are neglected. Only cross-interrogates for L2 misses are included in the various utilization components.[11]

[9] A description of this system and the model would require several separate papers and is not attempted. This system has been used and verified extensively.

[10] Detailed analysis is described in [6]. The model example in [6] is very similar but not identical to that used for comparison here.

[11] See [6, Section 5] for definition and inclusion of cross-interrogates in open-queue analysis.

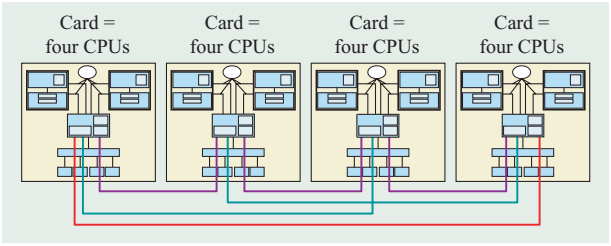


Figure 10

Configuration of 16-way multiprocessor on four cards with direct-connected buses between each card.

A miss in L4 that necessarily incurs a reload from main is loaded through to L3, L2, and, of course, to L1. More generally, a hit at any given level is reloaded to all upstream levels. All main memory and I/O are address-sliced shared, and there is one shared slice per processor chip (for details on the address-sliced sharing process, see [6]). Each of these slices (memory and I/O) is connected via a separate bus to a memory controller chip, and the latter is connected to a processor chip via one bidirectional bus. Thus, each processor chip contains one slice of main memory and one slice of I/O.

A 16-way multiprocessor configuration is obtained by interconnecting four cards via buses that all pass through the switch chip, as shown in **Figure 10**. Each of these buses is assumed to be two unidirectional buses. There are four CPUs per card, and four separate cards with direct simple interconnections between the cards, which results in simple bus delays and utilization calculations for off-card communications.

Results of open- and closed-model comparison

The open model was not as detailed as the closed MVA model, but the neglected details were second-order degradation effects and were not important enough to justify extensive additional work. Because of this, it was anticipated that the open model would give slightly more optimistic results than the closed model. This was found to be true in all comparisons.

The first comparison is that between the CPI and the L2 and L3 miss rates for L2 cross-interrogates of 50% and 10%. The second comparison is the utilization of the major unidirectional buses compared with the L3 miss rate, namely CPU-to-switch and switch-to-CPU buses. Varying the miss rates is equivalent to varying the size of the caches, where we can assume to a first approximation that the miss rates vary as the inverse of the square root of cache-size ratios.

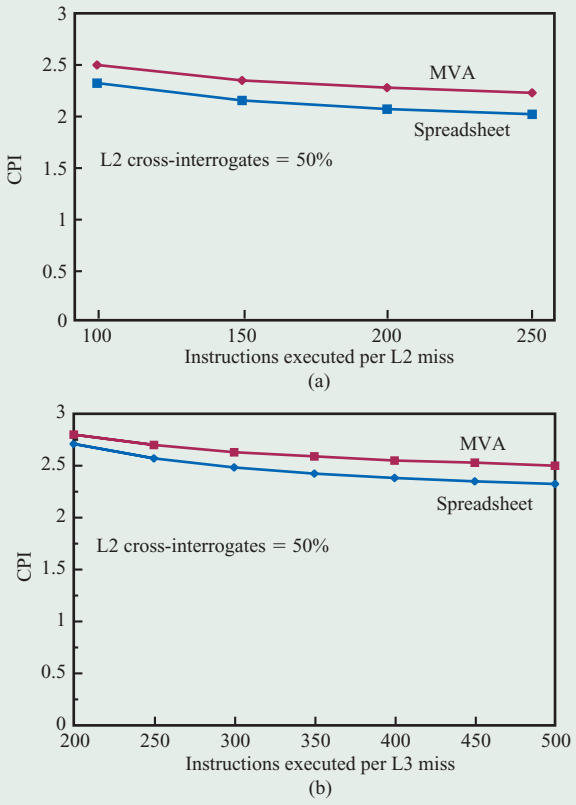


Figure 11

CPI as a function of (a) L2 and (b) L3 instructions per miss comparing spreadsheet (open) and MVA (closed) models. L2 cross-interrogates = 50%.

The CPI compared with L2 and L3 miss rates for L2 cross-interrogates of 50% are shown in **Figures 11(a)** and **11(b)**, respectively. As can be seen, the open model gives approximately an 8% (or less) more optimistic CPI than the closed MVA model for both the L2 and L3 varying in miss rates. However, the trends of the curves track each other quite accurately. It is the latter that is of most significance, because the absolute values are always in some doubt.

Similar comparisons of CPI as a function of L2 and L3 miss rates, but for L2 cross-interrogates of 10%, are shown in **Figures 12(a)** and **12(b)**. It can be seen that the curves are much the same as for the previous case, with cross-interrogates of 50%, except that the CPI is slightly smaller, as would be expected for a smaller cross-interrogate ratio.

Bus utilization is an important parameter in any multiprocessor configuration because it can have a significant effect on the finite cache penalty. One of the

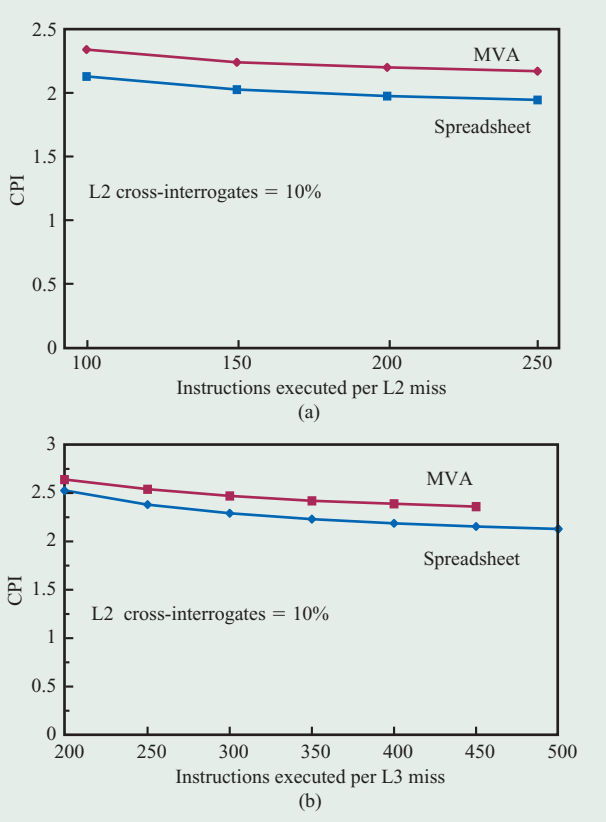


Figure 12

CPI as a function of (a) L2 and (b) L3 instructions per miss comparing spreadsheet (open) and MVA (closed) models. L2 cross-interrogates = 10%.

major buses in this model is that between the CPU and switch. These buses are unidirectional because of the high expected traffic, with the traffic on the bus from the CPU to the switch being somewhat different than that from switch to CPU, and are thus modeled separately. This traffic is summarized in **Figure 13**. The utilization of the switch-to-CPU bus as a function of the L3 miss rate is shown in **Figure 14(a)** and that of the bus in the opposite direction, from CPU to switch, in **Figure 14(b)**, both for an L2 cross-interrogation ratio of 50%. Once again, the open model is slightly more optimistic than the closed model, but both are within approximately 3% or better and have identical trends against the L3 miss rate.

Summary and conclusions

Comparisons of various mathematical formulations and general behavior of both open-queuing models and closed MVA queuing models of memory hierarchies indicate that the two methods are comparable in many respects. The detailed comparison of the behavior of one rather complex multiprocessor memory hierarchy using MVA with the more standard open-queue model gave similar results and distinctively identical trends in performance against L2 and L3 miss rates. This suggests that the two methods include the important performance-determining parameters in similar ways and gives some confidence that both are reasonable models. However, it should be noted that both cases were obtained for servers with relatively modest queues and therefore small queue delays. Thus, we would not expect large differences, although it is not obvious that the trend curves should be identical. Nevertheless, one major conclusion is that the choice between open- and closed-modeling methodology is of secondary importance, while ease of use and difficulty in constructing the model are major considerations. The skills and current knowledge of the user will determine the choice. The experience of the author is that if one has no knowledge of either method, the open model is easier to understand and construct.

Additional studies are required to compare various methods of analysis. Very few such comparisons have been done; rather, models are constructed on the basis of the particular knowledge and skills of the implementer, who typically is familiar with and uses only a single analytical methodology. The application of queuing models to memory hierarchies has potential for improvement and is an area with substantial payoff as the complexity and design time of systems continue to increase and require continued refinement and improvement.

Appendix A: Queue formulas: Some interesting observations and intuition

Table 1 shows various queue lengths and queue delays for exponential and constant service times (and a few for general Erlang service time). Several general and important observations concerning exponential and constant service times are as follows:

1. Total number of customers in queue (line + server) is smaller for constant service time by an amount

$$U^2 \frac{1}{2(1-U)}. \qquad \text{(A1)}$$

   For small U, this term becomes negligible, so the total queue lengths become the same for both exponential and constant service times.
2. The mean number of customers at the server is always U for both cases.
3. The mean number of customers in only the line is always $Q_x - U$, where Q_x is the total number in line + server for the respective service time. This follows from point 2

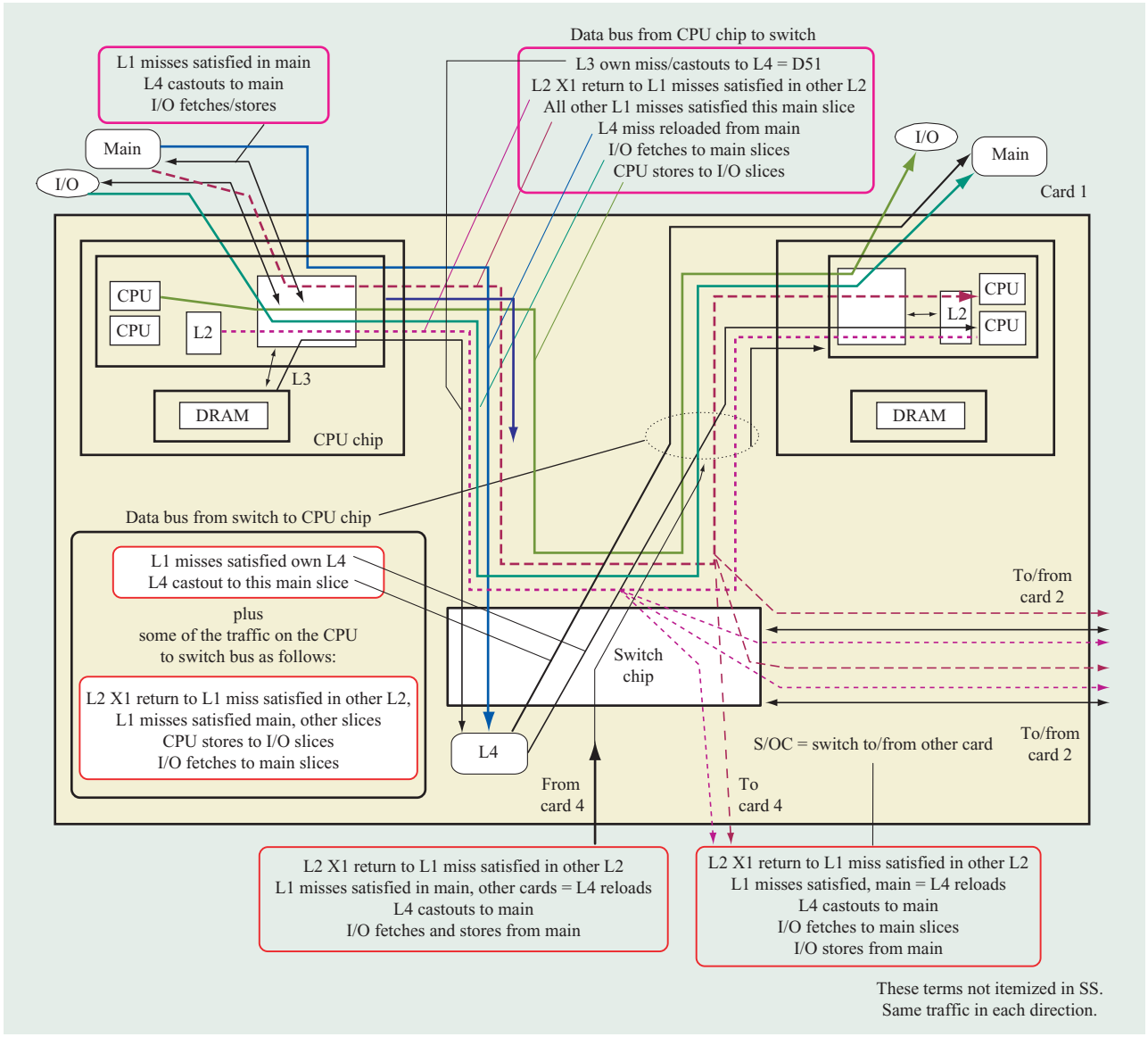


Figure 13

Data bus utilizations for CPU chip to and from switch.

above because the average number in the server is always U.

4. The total queue (line + server) delay, i.e., the mean residence time, is always

$$T_r = \frac{S_t}{U} Q_x, \tag{A2}$$

where Q_x is the respective total queue, as in Table 1.

5. Delay for the line alone with exponential service time is

$$T_{Le} = S_t \frac{U}{1-U}. \tag{A3}$$

For constant service time, the line delay is exactly one half of this, or

$$T_{Lc} = \frac{1}{2} S_t \frac{U}{1-U}. \tag{A4}$$

6. The average delay at the server is always the respective service time (S_t), where S_t is the *time constant* of the

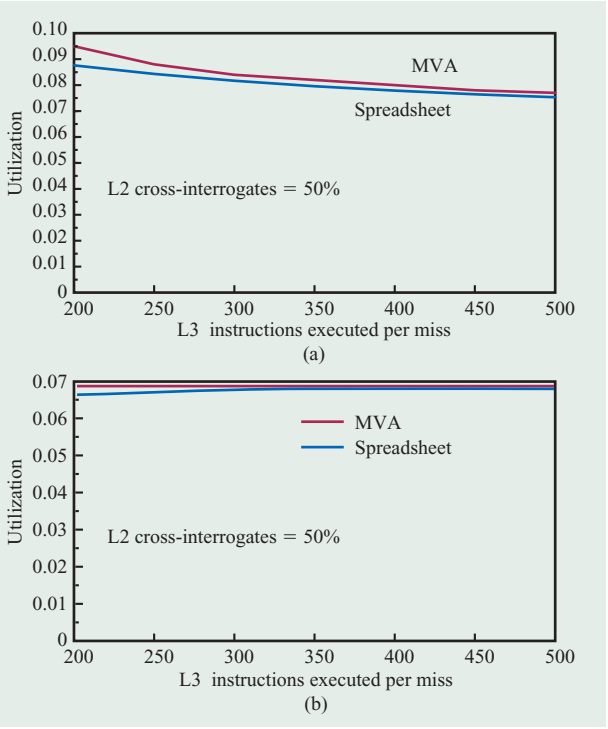


Figure 14

Utilization of (a) switch-to-CPU bus and (b) CPU-to-switch bus as a function of L3 instructions executed per miss, comparing spreadsheet (open) and MVA (closed) models.

exponential e^{-t/S_t} for exponential service time, and S_t is the given value for constant service time.

7. Utilization of the server always fixes the size and delay of the queue; this requires knowledge of the request rate and, of course, the service time. However, note that this is useful mainly for open queues. For closed-queue systems, a different method is required to determine average residence times and queue delays.

Appendix B: Meaning of memoryless process

In queuing theory, a *memoryless* process, distribution, etc. is often necessary and requires the use of an exponential function involving $e^{-\beta x}$. The actual simple meaning of this memoryless process is seldom clearly explained, but can easily be understood as follows.

Suppose we plot a curve of $1 - e^{-\beta x}$ as a function of x for $x = 0$ to infinity (large x). This gives the classical exponentially rising function shown in **Figure 15(a)**. Now plot a similar curve of $1 - e^{-\beta x}$, but start from any value of $x > 0$ to infinity (large x). This gives some upper portion of the first curve. The second curve is subsequently stretched in the vertical and horizontal directions to have the same physical size as the first curve, and results in a curve like the one shown in **Figure 15(b)** for x starting at 1. These two curves are identical in functional shape and cannot be distinguished from one another if superimposed. This is true for all such curves, no matter what initial value of x is chosen. In other words, the curves have the identical functional form and shape, no matter where they start. Hence, the *future* behaviors have no memory of what initial value of x was chosen (i.e., they are *memoryless*), and the future functional behavior is independent of past inputs. Unfortunately, only the exponential function is memoryless, which greatly limits the range of analysis for problems requiring memoryless behavior.

Appendix C: Some observations on the number of customers in the memory hierarchy and model complexity

MVA easily allows us to fix n as the number of requests or customers in the total network, which includes the CPU

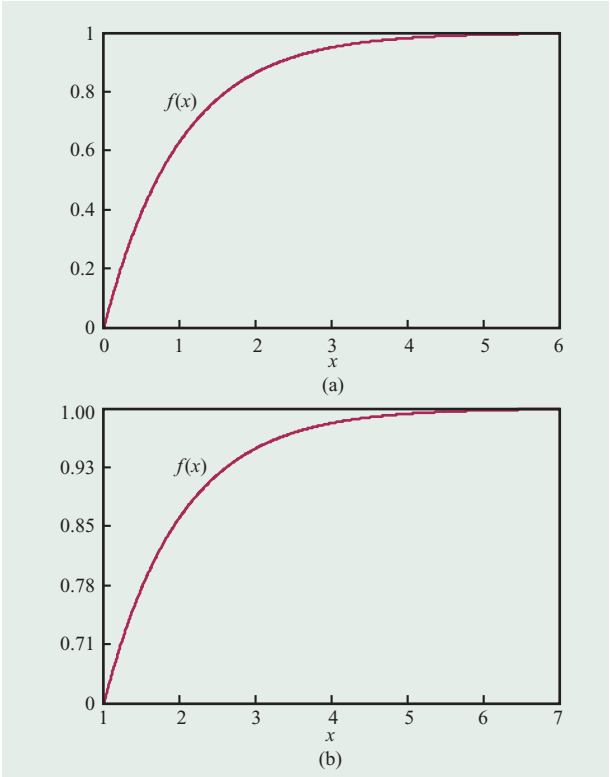


Figure 15

Exponential curves with different axes, illustrating meaning of *memoryless* assumption. Plot (a) is a standard exponential function, $1 - e^{-x}$ starting at $x = 0$ for $\beta = 1$. Plot (b) is an exponential function of (a), but starting at $x = 1$ rather than $x = 0$ for $\beta = 1$.

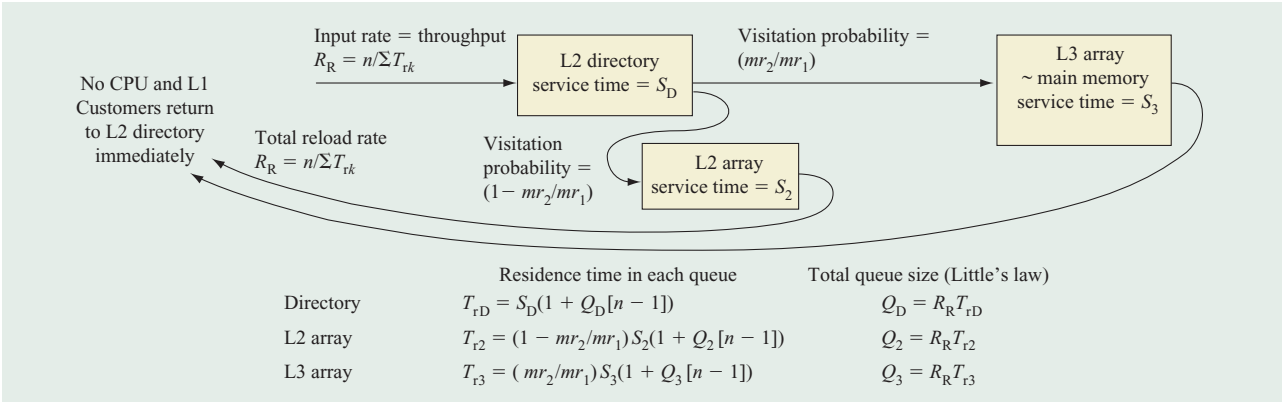


Figure 16

MVA model of memory hierarchy with L2 directory, L2 array, and L3 array, but no CPU delay center.

as a delay center. In such cases, we saw that the number of customers in the memory hierarchy itself was not, and could not, be fixed; i.e., we do not have this option. In an open queue, we could not specify any queues; rather, we could determine the number in each queue only after calculating all utilizations and then queues separately. The negative feedback mechanism via CPI limits open queues from becoming very large, but nevertheless they cannot be pre-specified. In reality, neither of these cases represents the actual working of a system with a memory hierarchy. In an actual system with multiple outstanding reloads (nonblocking caches), the average maximum number of allowed reloads, or at least a large fraction of these, ideally appear as customers in the memory hierarchy rather than the full system. In other words, the L1 miss rate is temporarily and partially decoupled from the system CPI during this time and, it is hoped, an improved CPI results. The FCP calculation would be valid only during periods when the CPU is actually stalled. An exact model of this will not be trivial, because more statistics will be required about the detailed behavior. Especially important will be some average time during which the CPU actually stalls because memory accesses can no longer be hidden. This will be very difficult to obtain and, in fact, is the essential parameter desired. A significantly more complex model will be required [9], and the model should be able to calculate this on the basis of other parameters.

A simple alternative model is an MVA network without the CPU as a delay center, so that the maximum number of customers is specified in the memory hierarchy itself. Such a network can be obtained from the previous model of Figure 5 simply by eliminating the CPU/L1 components, as shown in **Figure 16**. The CPU/L1 residency delay term is removed from the throughput R_R term, and all other components remain the same. The MVA calculations for such an example are shown in **Figure 17** (see Appendix D) for the same parameters as in Figure 6. *Interestingly, the calculated FCP of* 7.14 *in Figure* 17 *at* $n = 10$ is close to, but smaller in value than, the FCP of about 7.5 (extrapolation between 7.32 and 7.66) in Figure 6, which also has ten customers in the memory hierarchy queues (column N of Figure 6), requiring n to be between
21 and 22 total customers. The same is true for all other values of n; that is, the model without the CPU as a delay center always gives slightly improved (smaller) FCP than the model that includes the CPU as a delay center for the identical number of customers in the memory hierarchy queues. The deviation between the two models increases as n approaches 1. The problem with this model without a CPU is that the final CPI is not used anywhere in any of the queue nor FCP calculations, and must ultimately be included. However, the nonblocking nature of the cache decouples this dependency, at least temporarily, so that it has some elements of the real system. It is tempting to speculate that this improvement in FCP is related to the improvement from using a multi-issue, nonblocking cache, but this remains to be proven.

Appendix D: Spreadsheet figures

Spreadsheet figures 2, 4, 6, and 17 appear in Appendix D on pages 514–516.

	A	B	C	D	E	F	G	H	I	J	K	L	M
2		SD[L2dir}	S2[L2 array]	S3 [L3 array]						mr1 =	0.05	L2 Miss Ratio =	
3		2	12	140	Tcpu =	1		CPIinf =	1.2	mr2=	0.005	V3= m2/mr1 =	0.10
4								Total delay	Total delay	Total delay			
5	Req. Rate	Utilization with visitation probability						time in L2 Dir.	time in Q2	time in Q3	Sum of	FCP=	CPI =
6	RR =	UD of L2 Dir =	U2 of L2 array =	U3 of L3 array =	QD [dir line+server]	Q2 [line+server]	Q3 [line+server]	TrD	Tr2	Tr3	all Queues	mr1(TrD+Tr2+Tr3)	FCP + CPI[inf}
7	mr1/(CPI* Tcpu)	VD*RR*Sd	V2*RR*S2	V3*RR*S3	(UD-.5*UD^2)/(1-UD)	(U2-.5*UD^2)/(1-U2)	(U3-.5*UD^2)/(1-U3)	SD(1+ .5UD/(1-UD)	V2*S2(1+ .5U2/(1-U2)	V3*S3(1+ .5U3/(1-U3)	QD+Q2+Q3		
8													
9	0.018	0.037	0.198	0.256	0.0373	0.2223	0.3007	2.04	12.13	16.41	0.56	1.53	2.73
10													
11													
12		VD = 1 = probability of visiting Dir											
13													
14		V2 = (1 - mr2/mr1) = probability of visiting L2											
15													
16		V3 = mr2/mr1 = probability of visiting L3 {no misses in L3}											
17													
18													

Figure 2

Spreadsheet performance calculation of open-queue memory hierarchy with L2 directory and array, L3 array, and CPU/L1 as a requestor.

	A	B	C	D	E	F	G	H	I	J	K
2		St1 =	St2=	St3 =	St4 =						
3		2	4	8	20						
4		Residence	Residence	Residence	Residence						
5	# in system	time, in Q1	time in Q2	time, in Q3	time, in Q4	Req. Rate	Q length	Q length	Q length	Q length	Check if
6	n	T1	T2	T3	T4	RR	QL1	QL2	QL3	QL4	sum of QL = n
7		St1(1+ Q1[n-1])	St2(1+ Q2[n-1])	St3(1+ Q3[n-1])	St4(1+ Q4[n-1])	n/T1+T2+T3+T4	RR*T1	RR*T2	RR*T3	RR*T4	QL1+QL2+QL3+QL
8							0.00	0.00	0.00	0.00	
9	1	2.00	4.00	8.00	20.00	0.03	0.06	0.12	0.24	0.59	1.00
10	2	2.12	4.47	9.88	31.76	0.04	0.09	0.19	0.41	1.32	2.00
11	3	2.18	4.74	11.28	46.34	0.05	0.10	0.22	0.52	2.15	3.00
12	4	2.20	4.88	12.19	63.08	0.05	0.11	0.24	0.59	3.06	4.00
13	5	2.21	4.95	12.74	81.27	0.05	0.11	0.24	0.63	4.02	5.00
14	6	2.22	4.98	13.04	100.33	0.05	0.11	0.25	0.65	4.99	6.00
15	7	2.22	4.99	13.19	119.86	0.05	0.11	0.25	0.66	5.98	7.00
16	8	2.22	5.00	13.27	139.64	0.05	0.11	0.25	0.66	6.98	8.00
17	9	2.22	5.00	13.30	159.53	0.05	0.11	0.25	0.66	7.97	9.00
18	10	2.22	5.00	13.32	179.48	0.05	0.11	0.25	0.67	8.97	10.00
19	11	2.22	5.00	13.33	199.46	0.05	0.11	0.25	0.67	9.97	11.00
20	12	2.22	5.00	13.33	219.45	0.05	0.11	0.25	0.67	10.97	12.00
21	13	2.22	5.00	13.33	239.45	0.05	0.11	0.25	0.67	11.97	13.00
22	14	2.22	5.00	13.33	259.45	0.05	0.11	0.25	0.67	12.97	14.00
23	15	2.22	5.00	13.33	279.45	0.05	0.11	0.25	0.67	13.97	15.00
24	16	2.22	5.00	13.33	299.44	0.05	0.11	0.25	0.67	14.97	16.00
25	17	2.22	5.00	13.33	319.44	0.05	0.11	0.25	0.67	15.97	17.00
26	18	2.22	5.00	13.33	339.44	0.05	0.11	0.25	0.67	16.97	18.00
27	19	2.22	5.00	13.33	359.44	0.05	0.11	0.25	0.67	17.97	19.00
28	20	2.22	5.00	13.33	379.44	0.05	0.11	0.25	0.67	18.97	20.00

Figure 4

Simple MVA example showing iterative solution for queues, residence times, and request rates with four servers, each having a different service time.

A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2		SD[L2dir}	S2[L2 array]	S3 [L3 array]						mr1 =	0.05	L2 Miss Ratio =			
3		2	12	140	Tcpu =	1		CPIinf =	1.2	mr2=	0.005	V3 = m2/mr1 =	0.10		
4		Residence	Residence	Residence	Delay		L2 Dir	L2 Array	L3 Array	# requests					
5	# in system	time in L2Dir	time in Q2	time in Q3	time in CPU	Req. Rate	Q length	Q length	Q length	held in CPU=	Sum of	FCP=	CPI =	Sum of Q's	
6		TrD	Tr2	Tr3	TrC	RR	QD	Q2	Q3	#CPU	all QL + #CPU				
7	n	SD(1+ QD[n-1])	V2*S2(1+ Q2[n-1])	V3*S3(1+ Q3[n-1])	CPI*Tcpu/mr1	n/(TrD+Tr2+Tr3+TrC)	RR*TrD	RR*Tr2	RR*Tr3	RR* TrC	QD+Q2+Q3+#CPU	mr1(TrD+Tr2+Tr3)	FCP + CPI[inf}	QD+Q2+Q3	
8							0.00	0.00	0.00						
9	1	2.00	10.80	14.00	50.80	0.013	0.03	0.14	0.18	0.65	1.00	1.34	2.54	0.35	
10	2	2.05	12.30	16.53	54.88	0.023	0.05	0.29	0.39	1.28	2.00	1.54	2.74	0.72	
11	3	2.10	13.90	19.40	59.39	0.032	0.07	0.44	0.61	1.88	3.00	1.77	2.97	1.12	
12	4	2.13	15.55	22.59	64.28	0.038	0.08	0.59	0.86	2.46	4.00	2.01	3.21	1.54	
13	5	2.16	17.23	26.10	69.49	0.043	0.09	0.75	1.14	3.02	5.00	2.27	3.47	1.98	
14	6	2.19	18.89	29.89	74.97	0.048	0.10	0.90	1.42	3.57	6.00	2.55	3.75	2.43	
15	7	2.21	20.52	33.94	80.67	0.051	0.11	1.05	1.73	4.11	7.00	2.83	4.03	2.89	
16	8	2.23	22.10	38.22	86.54	0.054	0.12	1.19	2.05	4.64	8.00	3.13	4.33	3.36	
17	9	2.24	23.61	42.71	92.56	0.056	0.13	1.32	2.39	5.17	9.00	3.43	4.63	3.83	
18	10	2.25	25.04	47.40	98.69	0.058	0.13	1.44	2.73	5.69	10.00	3.73	4.93	4.31	
19	11	2.26	26.40	52.27	104.93	0.059	0.13	1.56	3.09	6.21	11.00	4.05	5.25	4.79	
20	12	2.27	27.67	57.31	111.25	0.060	0.14	1.67	3.46	6.73	12.00	4.36	5.56	5.27	
21	13	2.27	28.87	62.51	117.65	0.062	0.14	1.78	3.85	7.24	13.00	4.68	5.88	5.76	
22	14	2.28	29.98	67.84	124.10	0.062	0.14	1.87	4.24	7.75	14.00	5.01	6.21	6.25	
23	15	2.28	31.02	73.31	130.61	0.063	0.14	1.96	4.64	8.26	15.00	5.33	6.53	6.74	
24	16	2.29	31.98	78.89	137.17	0.064	0.15	2.04	5.04	8.77	16.00	5.66	6.86	7.23	
25	17	2.29	32.88	84.60	143.77	0.065	0.15	2.12	5.46	9.27	17.00	5.99	7.19	7.73	
26	18	2.30	33.71	90.40	150.40	0.065	0.15	2.19	5.88	9.78	18.00	6.32	7.52	8.22	
27	19	2.30	34.47	96.30	157.07	0.065	0.15	2.26	6.31	10.29	19.00	6.65	7.85	8.71	
28	20	2.30	35.18	102.29	163.77	0.066	0.15	2.32	6.74	10.79	20.00	6.99	8.19	9.21	
29	21	2.30	35.83	108.36	170.49	0.066	0.15	2.37	7.18	11.29	21.00	7.32	8.52	9.71	
30	22	2.31	36.44	114.50	177.24	0.067	0.15	2.43	7.62	11.80	22.00	7.66	8.86	10.20	
31	23	2.31	37.00	120.71	184.01	0.067	0.15	2.47	8.07	12.30	23.00	8.00	9.20	10.70	
32	24	2.31	37.51	126.98	190.80	0.067	0.15	2.52	8.52	12.81	24.00	8.34	9.54	11.19	
33	25	2.31	37.99	133.31	197.61	0.067	0.16	2.56	8.98	13.31	25.00	8.68	9.88	11.69	
34	26	2.31	38.43	139.69	204.43	0.068	0.16	2.60	9.44	13.81	26.00	9.02	10.22	12.19	
35	27	2.31	38.84	146.12	211.27	0.068	0.16	2.63	9.90	14.31	27.00	9.36	10.56	12.69	
36	28	2.31	39.22	152.59	218.12	0.068	0.16	2.66	10.36	14.82	28.00	9.71	10.91	13.18	
37	29	2.31	39.57	159.10	224.98	0.068	0.16	2.69	10.83	15.32	29.00	10.05	11.25	13.68	
38	30	2.32	39.89	165.64	231.85	0.068	0.16	2.72	11.30	15.82	30.00	10.39	11.59	14.18	
39															
40			V2 = (1 - mr2/mr1) = probability of visiting L2												
41				V3 = mr2/mr1 = probability of visiting L3 {no misses in L3}											
42															

Figure 6

Spreadsheet calculation of MVA model with L2 directory and array, L3 array, and CPU as a delay center.

A	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2		SD[L2dir}	S2[L2 array]	S3[L3 array]						mr1 =	0.05	L2 Miss Ratio =		
3		2	12	140	Tcpu=	1		CPIinf=	1.2	mr2=	0.005	V3 = m2/mr1	0.10	
4		Residence	Residence	Residence			L2 Dir	L2 Array	L3 Array					
5	# in system	time in L2Dir	time in Q2	time in Q3		Req. Rate	Q length	Q length	Q length		Sum of	FCP=	CPI =	Sum of Q's
6		TrD	Tr2	Tr3		RR	QD	QL2	QL3		all QL + #CPU	mr1(TrD+Tr2+Tr3)	FCP + CPI[inf}	QD+Ql2+Ql3
7	n	SD(1+ QD[n-1])	V2*S2(1+ Q2[n-1])	V3*S3(1+ Q3[n-1])		n/(TrD+Tr2+Tr3)	RR*TrD	RR*Tr2	RR*Tr3		QD+QL2+QL3			
8							0.00	0.00	0.00					
9	1	2.00	10.80	14.00		0.037	0.07	0.40	0.52		1.00	1.34	2.54	1.00
10	2	2.15	15.15	21.31		0.052	0.11	0.78	1.10		2.00	1.93	3.13	2.00
11	3	2.22	19.28	29.45		0.059	0.13	1.13	1.73		3.00	2.55	3.75	3.00
12	4	2.26	23.06	38.28		0.063	0.14	1.45	2.41		4.00	3.18	4.38	4.00
13	5	2.28	26.46	47.71		0.065	0.15	1.73	3.12		5.00	3.82	5.02	5.00
14	6	2.30	29.49	57.68		0.067	0.15	1.98	3.87		6.00	4.47	5.67	6.00
15	7	2.31	32.16	68.15		0.068	0.16	2.19	4.65		7.00	5.13	6.33	7.00
16	8	2.31	34.49	79.08		0.069	0.16	2.38	5.46		8.00	5.79	6.99	8.00
17	9	2.32	36.51	90.43		0.070	0.16	2.54	6.30		9.00	6.46	7.66	9.00
18	10	2.32	38.26	102.15		0.070	0.16	2.68	7.16		10.00	7.14	8.34	10.00
19	11	2.33	39.75	114.20		0.070	0.16	2.80	8.04		11.00	7.81	9.01	11.00
20	12	2.33	41.02	126.54		0.071	0.16	2.90	8.94		12.00	8.49	9.69	12.00
21	13	2.33	42.09	139.13		0.071	0.16	2.98	9.85		13.00	9.18	10.38	13.00
22	14	2.33	43.00	151.96		0.071	0.17	3.05	10.78		14.00	9.86	11.06	14.00
23	15	2.33	43.75	164.97		0.071	0.17	3.11	11.72		15.00	10.55	11.75	15.00
24	16	2.33	44.38	178.15		0.071	0.17	3.16	12.68		16.00	11.24	12.44	16.00
25	17	2.33	44.91	191.46		0.071	0.17	3.20	13.64		17.00	11.94	13.14	17.00
26	18	2.33	45.34	204.90		0.071	0.17	3.23	14.60		18.00	12.63	13.83	18.00
27	19	2.33	45.70	218.43		0.071	0.17	3.26	15.58		19.00	13.32	14.52	19.00
28	20	2.33	45.99	232.05		0.071	0.17	3.28	16.55		20.00	14.02	15.22	20.00
29	21	2.33	46.23	245.74		0.071	0.17	3.30	17.53		21.00	14.72	15.92	21.00
30	22	2.33	46.43	259.49		0.071	0.17	3.31	18.52		22.00	15.41	16.61	22.00
31	23	2.33	46.59	273.28		0.071	0.17	3.33	19.51		23.00	16.11	17.31	23.00
32	24	2.33	46.72	287.11		0.071	0.17	3.34	20.50		24.00	16.81	18.01	24.00
33	25	2.33	46.82	300.97		0.071	0.17	3.34	21.49		25.00	17.51	18.71	25.00
34	26	2.33	46.91	314.86		0.071	0.17	3.35	22.48		26.00	18.21	19.41	26.00
35	27	2.33	46.97	328.77		0.071	0.17	3.35	23.48		27.00	18.90	20.10	27.00
36	28	2.33	47.03	342.70		0.071	0.17	3.36	24.47		28.00	19.60	20.80	28.00
37	29	2.33	47.07	356.65		0.071	0.17	3.36	25.47		29.00	20.30	21.50	29.00
38	30	2.33	47.11	370.60		0.071	0.17	3.36	26.47		30.00	21.00	22.20	30.00
39														
40			V2 = (1 - mr2/mr1) =probability of visiting L2											
41				V3 = mr2/mr1 = probability of visiting L3 {no misses in L3}										
42														
43														
44														

Figure 17

Spreadsheet calculation of MVA model with L2 directory and array, L3 array, and no CPU.

Acknowledgments

The author gratefully acknowledges the vital assistance of Brian O'Krafka (formerly with IBM Austin, now with HP) in the use of EZMVA methodology and execution of the 16-way multiprocessor MVA model. Without his help, this comparison would not have been possible. The early work of Miko Lapasti (formerly at IBM Rochester, now at the University of Wisconsin) was helpful in the initial phases of constructing the MVA model of the 16-way multiprocessor. The author also thanks Men Chow Chiang of IBM Austin for various discussions concerning MVA modeling, Mark Squillante of IBM Research for assistance and guidance in various aspects of queuing theory, and Philip Heidelberger of IBM Research for helpful discussions on MVA theory.

**Trademark or registered trademark of Standard Performance Evaluation Corporation.

References

1. P. G. Emma, "Understanding Some Simple Processor-Performance Limits," *IBM J. Res. & Dev.* **41,** No. 3, 215–232 (May 1997).
2. M. Reiser and S. S. Lavenberg, "Mean-Value Analysis of Closed Multichain Queuing Networks," *J. ACM* **27,** No. 2, 313–322 (April 1980).
3. R. W. Wolff, *Stochastic Modeling and the Theory of Queues*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989.
4. E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.
5. D. Fink and D. Christiansen, *Electronics Engineers' Handbook*, Third Edition, McGraw-Hill, New York, 1989, pp. 5–46.
6. R. E. Matick, T. J. Heller, and M. Ignatowski, "Analytical Analysis of Finite Cache Penalty and Cycles per Instruction of a Multiprocessor Memory Hierarchy Using Miss Rates and Queuing Theory," *IBM J. Res. & Dev.* **45,** No. 6, 819–842 (November 2001).
7. P. Schweitzer, "Approximate Analysis of Multiclass Closed Networks of Queues," *Proceedings of the International Conference on Stochastic Control and Optimization*, Amsterdam, 1979.
8. M. K. Vernon, R. Jog, and G. S. Sohi, "Performance Analysis of Hierarchical Cache-Consistent Multiprocessors," *Perform. Eval.* **9,** No. 4, 287–302 (November 1989).
9. D. Sorin, V. Pai, S. Adve, M. Vernon, and D. Wood, "Analytic Evaluation of Shared-Memory System with ILP Processors," *Proceedings of the 25th International Symposium on Computer Architecture* (*ISCA*), Barcelona, 1998, pp. 180–191.
10. D. J. Sorin, J. L. Lemon, D. L. Eager, and M. K. Vernon, "Analytic Evaluation of Shared-Memory Architectures for Heterogeneous Applications," *Technical Report No. 1404*, Computer Science Department, University of Wisconsin, Madison, 1999; see *http://www.cs.wisc.edu/multifacet/papers/tr1404b_model.pdf*.
11. M. Reiser, "A Queueing Network Analysis of Computer Communication Networks with Window Flow Control," *IEEE Trans. Commun.* **27,** No. 8, 1199–1209 (August 1979).

Received August 27, 2002; accepted for publication January 30, 2003

Richard E. Matick *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (matick@us.ibm.com).* Dr. Matick is a Research Staff Member in Systems Technology and Microarchitecture. After receiving his B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from Carnegie Mellon University in 1955, 1956, and 1958, respectively, he joined the IBM Research Division and worked in the areas of thin magnetic films, memories, and ferroelectrics. As manager of the Magnetic Film Memory group from 1962 to 1964, he received an IBM Outstanding Invention Award for the invention and development of the thick-film read-only memory. He joined the Technical Staff of the IBM Director of Research in 1965 and remained until 1972, serving in various staff positions and as Technical Assistant to the Director of Research. In 1972 he took a one-year sabbatical to teach at the University of Colorado and IBM in Boulder, Colorado. Dr. Matick spent the summer of 1973 teaching and doing research at Stanford University. In 1986, he received an IBM Outstanding Innovation Award and in 1999 an IBM Corporate Patent Portfolio Award as co-inventor of the industry-standard video RAM memory chip, used to provide the high-speed, high-resolution display bit buffer in personal computers and many workstations. His work in high-density CMOS cache memory design, for which he received an IBM Outstanding Technical Achievement Award in 1990, served as the foundation for the high-speed cache system used in the IBM RISC/6000 series processors. He is co-initiator of the concept of logic-based embedded DRAM, which has become a key IBM strategy for systems on a chip. It was initially conceptualized in 1990 and became a reality in the late 1990s, and is now being offered to all IBM ASIC customers. Dr. Matick is the author of the books *Transmission Lines for Digital and Communication Networks*, McGraw-Hill, 1969 (reprinted as an IEEE Press Classic Reissue in 1995 and in paperback in 2001, and *Computer Storage Systems and Technology*, John Wiley & Sons, 1977. He is also the author of chapters on memories in *Introduction to Computer Architecture* (H. Stone, Editor), SRA 1975 (First Edition), 1980 (Second Edition) and in *Electronics Engineers' Handbook*, Second and Third Editions, McGraw-Hill, 1982 and 1989. He also contributed the "Cache Memory" entry in the *Encyclopedia of Computer Science*, Third Edition, Van Nostrand Reinhold, 1993. Dr. Matick is a member of Eta Kappa Nu and an IEEE Fellow.