Analytical analysis of finite cache penalty and cycles per instruction of a multiprocessor memory hierarchy using miss rates and queuing theory

by R. E. Matick
T. J. Heller
M. Ignatowski

Advances in technology have provided a continuing improvement in processor speed and capacity of attached main memory. The increasing gap between main memory and processor cycle times has required increasingly more levels of caching to prevent performance degradation. The net result is that the inherent delay of a memory hierarchy associated with any computing system is becoming the major performance-determining factor and has inspired many types of analysis methods. While an accurate performance-evaluation tool requires the use of trace-driven simulators, good approximations and significant insight can be obtained by the use of analytical models to evaluate finite cache penalties based on miss rates (or miss ratios) and queuing theory combined with empirical relations between various levels of a memory hierarchy. Such tools make it possible to readily determine trends in performance vs. changes in input parameters. This paper describes such an analysis approach—one which has been implemented in a spreadsheet and used successfully to perform early engineering tradeoffs for many uniprocessor and multiprocessor memory hierarchies.

1. Introduction

Queuing theory has a long history and has been applied to numerous systems [1]. It has historically been used for analyzing telephone and switching network traffic and for performance evaluations of early computer systems [2]. These early computer systems typically contained many I/O devices which served multiple user programs, resulting in characteristics similar to those of a telephone network.

While the processing power of computers has provided the ability to analyze very complex systems in more detail,

0018-8646/01/$5.00 © 2001 IBM

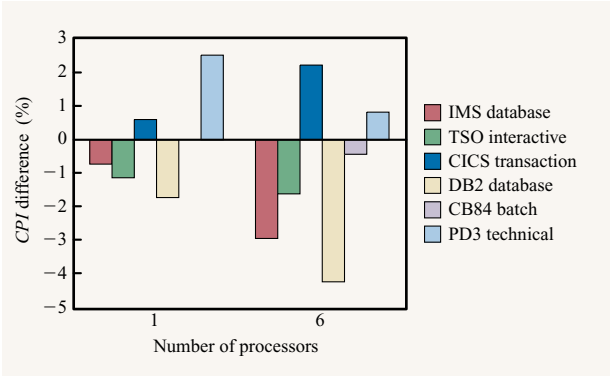


Figure 1

Difference between *CPI* of ES/9000 system and that of model, for various applications running on a uniprocessor and 6-way multiprocessor.

nevertheless, the system on the drawing board has always overtaxed the ability to model and analyze it. For instance, a uniprocessor of the near future could be running at a 2-GHz cycle time (0.5-ns cycle) and processing an average of two instructions per cycle. To model such a system using a trace-driven simulation and to simulate 60 seconds of its CPU time would require about 240 billion instructions in the trace. If each instruction were to require eight bytes of storage, the trace alone would require nearly 2000 GB to store. In addition, the time to perform the simulation would be unreasonable. A cycle-by-cycle simulation model can typically model the next-generation system at a rate of 0.1 to 1 million instructions per second, depending on the complexity of the model and the system used to perform the modeling. Assuming the worst case, to execute a cycle-simulation model of 60 seconds of the above next-generation design on an available processor would require nearly 666 hours, or more than a month of processor time. Analytical analysis can reduce this by orders of magnitude, typically requiring at most a few seconds.

This paper describes one application of open input queuing theory to rather complex multiprocessor memory hierarchy performance analysis. The input is assumed to follow a Poisson distribution, which allows the use of relatively simple equations to depict queues and queue delays. Such analysis, in various forms, has been used within IBM for more than 15 years. It is one of many tools used for the projection of hardware system performance for servers. Several variations of this methodology have also been used for performance projections, including some for recent IBM servers. A highly accurate projection of the performance of future high-end server designs has been achieved using the analysis technique described. The accuracy has been greatest for designs which are structurally similar to the preceding generation design. Accurate miss-rate data obtained from carefully validated processor and system bus traces are obviously required for good results. The effort required to collect and process these traces exceeds the effort required to build the performance model. Accuracy of ±3% was achieved for the prediction of IBM ES/9000* performance (measured in cycles per instruction, or *CPI*) with a model structure that had been verified using actual performance data from IBM ES/3090* mainframes. These comparisons are summarized in **Figure 1** for different applications (miss statistics) running on a uniprocessor and a 6-way multiprocessor (MP). More detailed and recent comparisons are currently proprietary and hence have not been included.

The model to be described is a memory hierarchy driven by cache misses. As indicated previously, the model assumes an open input queue, which means that any number of input misses can be outstanding at any given time. Since the actual hardware allows only a fixed number of outstanding misses per processor, typically one or at most a few, modified types of analytical models have been pursued to account for this restriction. One well-used technique is mean value analysis (MVA) [3], from which a closed-input model can be constructed, thus restricting the number of outstanding input misses. A separate detailed study was performed which compared the results of an MVA closed queuing model to those of our open-queue, constant-service-time model, for a relatively complex MP model similar to that described in this paper. The results showed that the two techniques were typically within about 3%, with a maximum difference of 10% over a wide range of miss rates and cache sizes.[1] In addition, the trend curves of performance and bus utilizations vs. various miss rates had identical shapes for essentially all cases studied. Since open-queue models are easier to model, this method is still widely used. Other types of analytical analysis of different aspects of an MP system are also possible [4, 5].

Basic concepts and analytical approach

The ideal, raw processing power of a processor is measured in *CPI* for an infinite cache [6], i.e., its first-level cache functions as if there were no cache misses and thus no reload penalties. The actual performance for any system with a memory hierarchy is considerably less because of stalls and delays caused by cache reloading. This additional delay, measured as cycles per instruction executed, is typically known as the finite cache penalty

[1] The open input queue is self-limiting for memory hierarchy models of the type used herein—see Appendix A. Thus, the use of an open rather than a closed input queue appears to be a second-order consideration for such systems and the assumed workload statistics. This is not generally true, but often can be so.

(*FCP*). These two parameters are added together to obtain the actual system performance:

$$CPI[\text{system}] = CPI[\text{infinite cache}] + FCP. \quad (1)$$

The *CPI*[infinite cache] is independent of the memory hierarchy and is assumed to be given. The attached memory hierarchy affects only the *FCP* term. We start with a uniprocessor having a simple memory hierarchy and derive the additional cycles per instruction incurred by cache misses and reloads from the hierarchy. The resulting equation for the *FCP* of a uniprocessor is relatively simple, involving miss rates (or miss ratios) and delays.

We then add complexities such as those occurring in actual systems when multiple miss requests from multiple processors are permitted to percolate throughout the memory hierarchy.[2] This introduces *two additional features* to the uniprocessor hierarchy model: 1) additional delay structures representing some or all levels of the hierarchy in order to provide delay paths representing cross-interrogation for cache coherency[3] or for address-space-shared, distributed memory[4] and 2) additional queue delays at various points in the model. Fundamentally, this complexity only adds more terms to the *FCP* part of Equation (1). Each such term is represented by a separate calculation, and the final *FCP* is a sum of all of these terms. In the general case, multiple processors can and will create multiple requests on the various resources in the memory hierarchy, such as buses, arrays, staging buffers, etc. Such multiple requests result in queuing delays as part of the reload delay terms in the *FCP* equation. Standard queuing equations are used to approximate all queue delays. However, as will be seen, these queue delays depend on the request rates for reloads, which in turn are inversely proportional to the *CPI* (system)—the smaller the *CPI*, the faster the instructions are processed, thereby generating more memory requests per second. But any increase in memory requests per second (due to a decrease in the *CPI*) creates larger queue delays, which then increase the *FCP* value and hence the *CPI* value, making the queue delays smaller, etc. The new queue delay must then be used to recalculate the resulting new *CPI* value; this recalculation is continued until the current *CPI* value produces queues that match (approximately) the new *CPI* for the new queues. Thus, an iterative type of recalculation can be used and is available in all common spreadsheets or mathematical analysis software such as Mathsoft's Mathcad**. (The time needed for the recalculations is insignificant for all cases encountered.) An iterative calculation is not fundamentally necessary, since the final *FCP* value can be expressed as a large polynomial equation. Any tool with a polynomial or simultaneous equation solver can be used to evaluate it. However, the order of the polynomial typically goes as $x + 1$, where x is the number of queues in the system. Expressing the complete polynomial as one function can be quite complex and prone to error, thus making a spreadsheet or similar equation-solving software more user-friendly.[5]

2. Finite cache penalty for uniprocessor memory hierarchy

A uniprocessor system consists of a single processor having an n-level memory hierarchy, as illustrated in **Figure 2**. A processor and first-level cache (L1) are typically a single, self-contained unit on an island or chip and thus are not available for optimization as part of the hierarchy analysis. Rather, the processor and L1 are the source which generates misses (and possibly castouts)[6] at a given miss rate of mr_1 misses per instruction executed. If there were *no* misses (infinite L1 cache) or if the miss latency time were 0, the resulting cycles per instruction would be the *CPI* value at infinite cache. It is assumed that this parameter is given. Our task is to determine the *FCP*, which is the additional number of cycles per instruction required for the given memory hierarchy. Miss rates are used as the measure of the miss characteristics of each level of the hierarchy, where

$$\textit{miss rate} = \textit{number of misses per instruction executed by the processor } (mr_1). \quad (2)$$

It is possible to express the *FCP* in terms of a miss ratio: the number of misses per access to that level. These parameters are equivalent, except for one small difficulty with the miss ratio of the L1 cache, as discussed in Appendix B.

In Figure 2, the miss latency time at each level is assumed to be a fixed constant T_n, where $n = 2, 3, 4$, etc. for each downstream level as indicated. This time is the total effective time to reload a hit at any level, including all electrical, logical, and trailing-edge delays, as discussed in Section 3. The L1 miss rate, mr_1, serves as input to the memory hierarchy for the *FCP* calculation. The task at hand is to determine the total average number of cycles per instruction required to reload these mr_1 misses per instruction, or "misses/I." All L1 misses first interrogate a second-level cache, L2, producing some L2 hits, with the remainder being L2 misses. The L2 misses propagate to L3, producing some L3 hits, with the remainder being L3 misses. This hit–miss behavior continues to the level that

[2] A similar situation can result from a uniprocessor architecture which allows multiple outstanding cache misses.
[3] See the section on private or semiprivate cache with cross-interrogation for definition.
[4] A linear, logical memory address space can be implemented as a single physical array, or it can be broken into pieces which are distributed in arrays that are physically distinct and separated from one another. The latter is known as address-space-shared, distributed memory—see the section on shared cache, address-sliced.
[5] Lotus 1-2-3** and Mathcad have been used to implement models.
[6] See the section on castout ratio approximation.

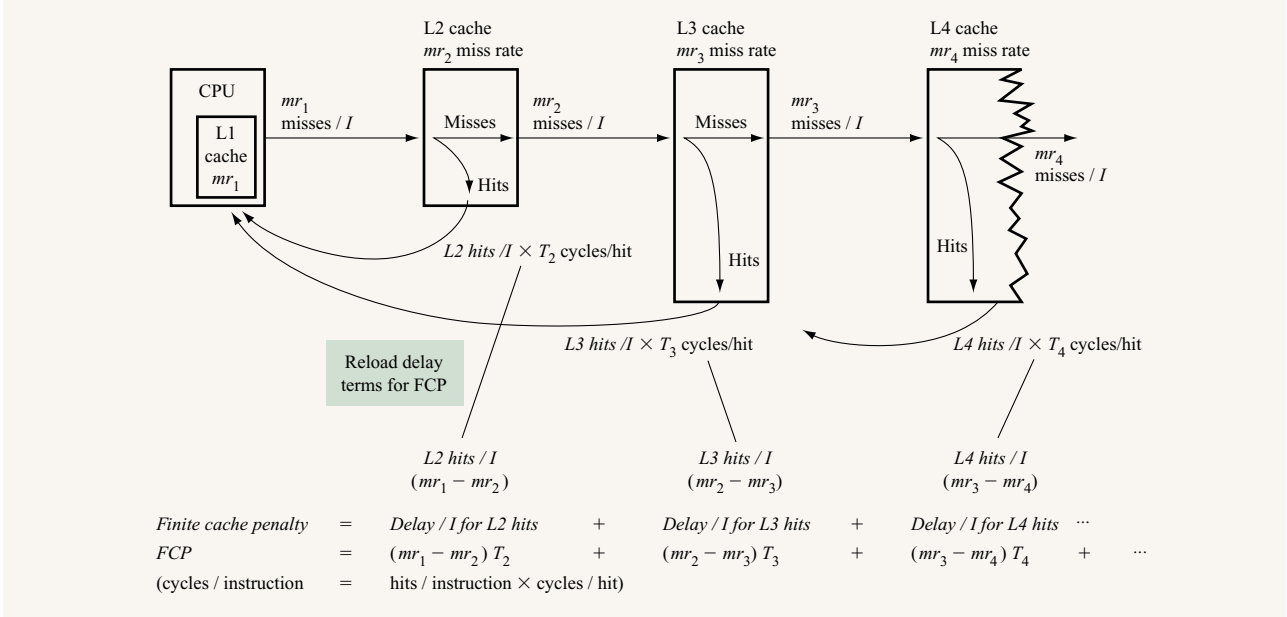


Figure 2

Uniprocessor with simple n-level memory hierarchy.

produces only hits, which is main memory in this model. I/O is included indirectly and affects only the queuing demand on the data buses and main memory. A specific hit at any level is reloaded with appropriate delay, and terminates any further interrogations downstream.[7] Portions of the total average reload delay will come from each level. For a general case, this will include hits in L2 with delay T_2, hits in a third-level cache, L3, with delay T_3, hits in L4 with delay T_4, etc., as shown in Figure 2. These amounts are easily determined as follows. The hit rate, hr_n, at any level n (except cross-interrogated levels—see below), is expressed in terms of inputs, outputs, and misses per instruction (I) given by

$$hr_n = inputs/I - outputs/I = misses/I\,[\text{previous level}] - misses/I\,[\text{current level}] = mr_{n-1} - mr_n\,. \quad (3)$$

The portion of the total *FCP* contributed by each level, n, of the cache hierarchy is the number of hits [Equation (3)] multiplied by the average effective reload time per hit of that level. *FCP* is expressed in units of processor cycles per instruction, so all delays T_n are expressed in units of number of processor cycles per hit rather than absolute time. Thus, the contribution to the finite cache penalty, in cycles per instruction, for any level of the hierarchy is given by

$$FCP_n = hr_n \times T_n = (mr_{n-1} - mr_n)T_n\,. \quad (4)$$

[7] Hits flow downstream from L1 to L2 to L3, etc., while reloads flow upstream.

The total *FCP* of an n-level hierarchy is the sum of all individual terms, or

$$FCP = \sum_{n=2}^{z} (mr_{n-1} - mr_n)T_n\,. \quad (5)$$

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

$$FCP = (mr_1 - mr_2)T_2 + (mr_2 - mr_3)T_3 + (mr_3 - mr_4)T_4 + mr_4T_{\text{main}}\,. \quad (6)$$

[Note: Hit rates and *FCP* terms for a cross-interrogated portion take a different form, per Equations (8) and (9) in Section 5.]

3. Effective reload time and trailing-edge-effect approximation

In a typical cache memory hierarchy, a miss in L1 will hit in some downstream level, and requires multiple processor cycles to reload the full cache block or line. The hit at the downstream level will usually start at the logical word boundary which caused the original miss. This logical word is "loaded through" on the data buses to the processor to enable the task to restart quickly. In the meantime, the remainder of the cache block is transferred (reloaded) to the requesting cache on subsequent bus cycles while the processor continues processing the task. If the processor requires another logical word in the currently reloading

cache block and it has not yet been reloaded, the processor must wait until it is available. This is known as the "trailing-edge" effect, and it adds additional average delay to the total effective reload time at any given level. An empirical approximation is that the additional trailing-edge delay,[8] on average, is some fixed percentage, P, of the total time to complete the block (line)[9] reload after the first logical word is reloaded. If we let N_{BR} equal the total number of bus cycles to fully reload a block, this can be expressed as

$$\textit{trailing-edge delay} = P(N_{BR} - 1)BT_{n+1,n}, \tag{7}$$

where $BT_{n+1,n}$ is the total bus delay, in cycles, between levels n and $n + 1$, and P can vary from 10% to 30% or more, depending on the application. For instance, if BT_{21} is three processor cycles (i.e., bus running at 1/3 processor frequency), and P is 10%, then for a 256-byte block reloaded from L2 to L1 in 16 bus cycles (i.e., 16 bytes per bus cycle), the predicted trailing-edge delay adder is 4.5 processor cycles. This amount is added to the total time, T_2, to access the first logical word from L2, and the sum becomes the total effective reload delay. Reducing the block size to 128 bytes reduces this delay to 2.1 processor cycles. Thus, this delay adder can be significant or negligible, depending on the design point. Large, high-performance systems are typically designed with wide, high-speed buses such that this effect is small.

4. Castout ratio approximation

A store-in (or write-back) cache is one for which any modified blocks are stored in this cache and the modifications do not appear anywhere else. Thus, when a cache miss requires the replacement of a valid, modified cache block, this block must be "cast out" to some higher level of the cache, typically the next level. In contrast to this, a store-through (or write-through) cache will store all writes to more than one level of cache, typically the current one and next higher level, thus not requiring a castout.

For a store-in cache, the castout (CO) ratio is assumed to be a constant, ranging from 20% to 40% of misses. These are typical values observed for many different workloads over many years. For all cases we have modeled, any value within this range is typically acceptable, since the effect of castouts on the number of *CPI* was too small to justify additional complexity. The user can determine the significance of this effect in any given model by changing the CO ratio and observing the effect on the final *FCP* value. If the effect is significant, more accurate statistics may be needed.

[8] Empirical approximation from the work of Keith Langston, IBM Server Group, Poughkeepsie, New York.
[9] The terms *block* and *line* are both used to refer to the smallest replaceable unit in any given cache level.

5. Definitions of cache types

In order to understand the general case of a multiprocessor hierarchy with multiple processors all making memory requests, it is expedient to consider a single processor making requests to a multiprocessor hierarchy. In this manner, we will see how the single accesses are partitioned to the various multiprocessor memory modules, from which the multiple, simultaneous accesses are easily derived afterward.

A multiprocessor system can be constructed with caches at any level partitioned in various physical configurations. Two major types are typically used for multiprocessor modeling in addition to the simple structures already shown. These two types and their corresponding *FCP* are detailed below.

Private or semiprivate cache with cross-interrogation

A private (or semiprivate) cache refers to one that is dedicated primarily to serving one (private) or a few (semiprivate) processors directly. Other processors access this cache for data only by cross-interrogation for cache coherency. This occurs only if a given processor incurs a miss in its private cache and sharing is supported. If the latter is true, the system must examine these other private caches to see if the data is there and take appropriate action. This analytical analysis models the effects of such cross-interrogations on the array and bus utilizations and resulting queue delays—the model need not and does not do anything with respect to the action or results occurring in an actual system except for the inclusion of queues and timing delays for transfers of data found in other caches.

Cross-invalidation For such private or semiprivate caches, shared data can exist in several such caches simultaneously, in a read-only state. If one cache should subsequently receive a request for a store operation to one such block, the block is already in the local cache, so no data transfer is required. However, this cache block must be invalidated in all other caches. Since the invalidation procedure uses only address and control lines, and these are typically very lightly utilized in comparison to data buses, the cross-invalidations are not modeled—their effect on the *FCP* cannot usually be seen in this model. However, this need not always be true. For such cases, statistics on cross-invalidation rates will be required. These should then be included in the utilization calculations for each appropriate shared bus, directory array, etc.

FCP for a private cache level with cross-interrogates to other private caches at same level, each having different delay

Cross-interrogates (XI) occur in an MP system as follows: A task on one processor makes an access to its private

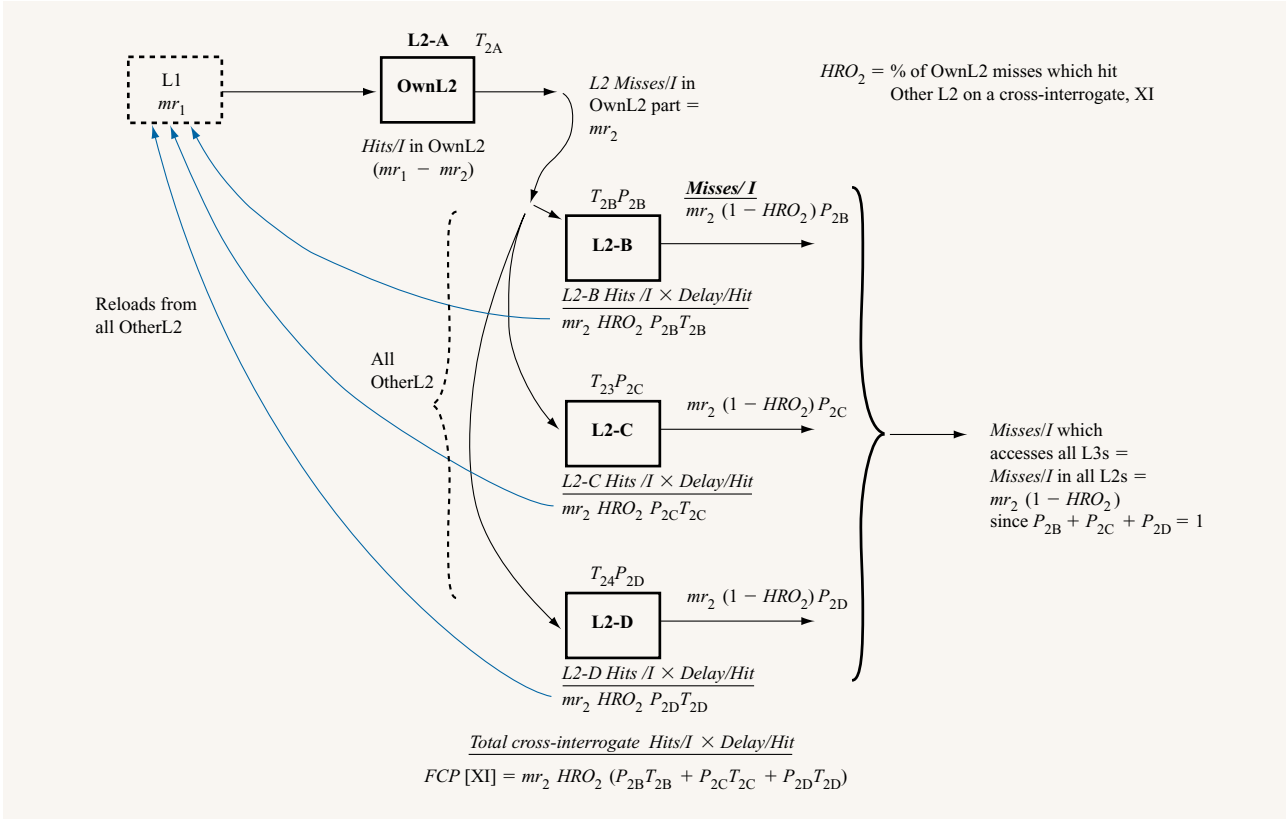


Figure 3

Representation of component of finite cache penalty for cross-interrogates in other modules of private L2s.

cache (say L2), and the access results in a miss. If the system supports sharing and there are other private L2s in the system, the desired information may possibly be residing in one of these other L2s, or "OtherL2."[10] In some cases, it may be the only valid copy in the system. Thus, in order to obtain the correct information, a cross-interrogation with transfer of the cache block (line) may be required.

For such cases, the single processor misses emanating from L2 shown in Figure 2 do not access L3; rather, they are redirected to all OtherL2 in the system which must be cross-interrogated. The L1 reload request first goes to the private "OwnL2" associated with the L1 generating the miss, as illustrated in **Figure 3**. The misses per instruction emanating from OwnL2 then access the OtherL2. The statistical parameter used for determining the hit and miss rates to these OtherL2 is a hit ratio, HRO_2, which specifies the percentage of OwnL2 misses which hit all OtherL2. (Note that HRO_2 does not include cross-invalidates which only invalidate a block in another cache, with no data transfer.) This hit ratio is typically in the range of 10% to 50% for many cases, the exact percentage being dependent on the system and application[11] [7]. Thus, we know that the hit rate to all OtherL2 will be

$$hit\ rate[\text{hits/I}]\ \text{to all OtherL2} = hro_2 = mr_2 \times HRO_2, \qquad (8)$$

where mr_2 is the usual miss rate for OwnL2 and HRO_2 is the percentage of OwnL2 misses which hit all OtherL2. The OtherL2 typically consists of several modules with different reload delay times T_{2X}, as indicated in Figure 3. (Modules with equal or nearly equal delays can be grouped together as one delay group.) The hits to all OtherL2 are apportioned to each module, B, C, D, etc. of L2 in terms of probability of access, derived later. The misses from all OtherL2 become the input accesses to L3 as indicated. Thus, the *FCP* component contributed by cross-interrogation to all OtherL2 is given by

[10] If this level is address-slice-shared, the desired block could not reside in another cache at this same level.

[11] For example, Table 4.1 of Reference [7] gives HRO_2 of 13.5% for TPC-B** and 16.1% for TPC-D (sum of rows 4, 5, and 6) for a 512KB L2. Larger L2s tend to have larger values.

$$FCP[\text{OtherL2}] = mr_2\, HRO_2\, P_{2B}T_{2B} + mr_2\, HRO_2\, P_{2C}T_{2C} + mr_2\, HRO_2\, P_{2D}T_{2D} + \cdots = mr_2\, HRO_2\, (P_{2B}T_{2B} + P_{2C}T_{2C} + P_{2D}T_{2D} + \cdots) \quad (9)$$

where T_{2B}, T_{2C}, and T_{2D} are the effective reload times of each private L2, and P_{2B}, P_{2C}, and P_{2D} are the probabilities of accessing L2, typically each equal to 1/#OtherL2. [Note that Equation (9) is different from Equation (4), which represents hits out of non-cross-interrogated levels. This results from the use of a known miss ratio (HRO_2) rather than a miss rate, which are easily shown to be equivalent.

The accesses/I to L3 are now the sum of the misses/I out of all OtherL2, or

$$Access/I[\text{to L3}] = mr_2(1 - HRO_2)P_{2B} + mr_2(1 - HRO_2)P_{2C} + mr_2(1 - HRO_2)P_{2D} = mr_2(1 - HRO_2)(P_{2B} + P_{2C} + P_{2D} + \cdots). \quad (10)$$

The sum of all probabilities P_{2B}, P_{2C}, P_{2D}, etc. must equal 1, so Equation (10) reduces to

$$Access/I[\text{to L3}] = mr_2(1 - HRO_2). \quad (11)$$

If there are no cross-interrogates ($HRO_2 = 0$), the above reduces to mr_2 as it should.

Shared cache, address-sliced

Suppose a system has four separate L3 arrays in which the address space is divided into four separate linear pieces, with the first piece mapped to the first L3 array, second piece to the second array, etc. In other words, the address is bit-sliced on two of the real address bits. We model this by assuming random addresses so that any given cache access will have the same probability (equal to ¼) of being accessed from any of the four arrays. Thus, since the arrays will often have different reload delays, the average reload time is scattered randomly among the four address-sliced arrays which make up the single logical L3.

FCP for a multiple-module, address-slice-shared cache level with each module having different delay

Consider a hierarchy similar to that in Figure 2, except that one of the levels, say L3, consists of m physical modules as in **Figure 4**, each module having a different effective reload delay, T_{3m}. It is assumed that the L3 address space is divided among these modules such that any given address will always access module 1 with probability P_{31}, module 2 with probability P_{32}, module 3 with probability P_{33}, etc., and module m with probability P_{3m}. If the address space is equally divided among these m modules, which is often the case, the module probabilities

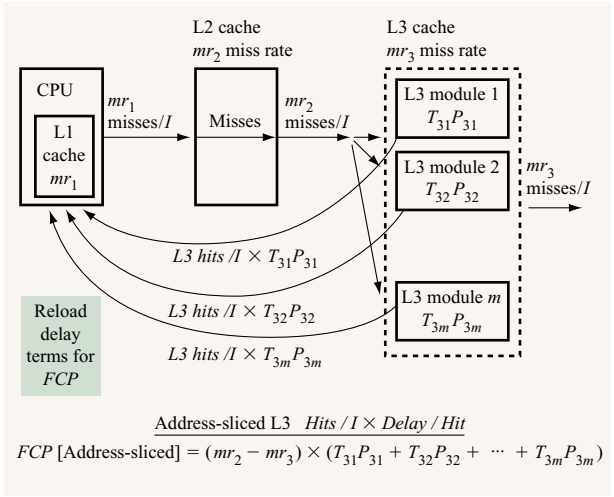


Figure 4

Memory hierarchy with m modules of address-sliced L3, each with a separate reload delay.

are identical and are all equal to $1/m$. Note that mr_3 is the miss rate of the full address range seen by any one processor, so it is the miss rate of all of the combined L3 slices, i.e., the logical L3 seen by a processor.

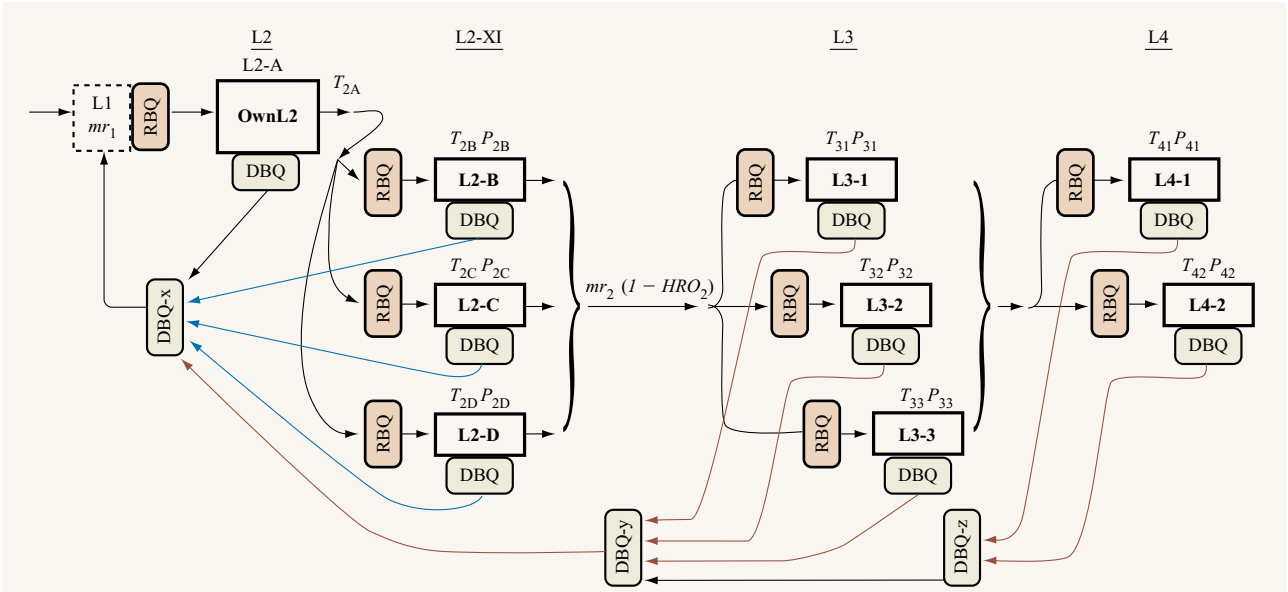
For the general case, the finite cache penalty FCP_3 for this level consists of m separate components, as shown in Figure 4. The essential idea is to apportion the reload delay to the separate L3 modules in direct relation to the probability of accessing that module for the reload request from L2. As shown in Figure 4, the FCP component for an address-slice-shared L3 is

$$FCP_3 = (mr_2 - mr_3)(T_{31}P_{31} + T_{32}P_{32} + T_{33}P_{33} + \cdots + T_{3m}P_{3m}). \quad (12)$$

Obviously, since the above component reduces to the L3 component in Equation (6) when the L3 address space is divided equally among all modules, the probabilities are all equal, summing to 1, and the reload delays are each equal to T_3. The above two analytical formulations for finite cache penalty terms of a single processor accessing a multiprocessor hierarchy having private and shared cache levels are used below in the analysis of a general multiprocessor example.

6. Multiprocessor systems: General model

A multiprocessor system is basically an interconnection of multiple simple hierarchies of Figure 2 in which the various cache levels each have several modules (components) which can be private, semiprivate, locally shared, or globally shared. The analysis calculates the *FCP of one processor in a manner similar to that done previously*,



Total FCP =

L2 $[mr_1 - mr_2]\,[T_{2A} + \Sigma(RBQ) + DBQ_{2A} + DBQ_x] +$

L2-XI $[mr_2\,HRO_2][(P_{2B}(T_{2B} + \Sigma(RBQ) + DBQ_{2B} + DBQ_x) + P_{2C}(T_{2C} + \Sigma(RBQ) + DBQ_{2C} + DBQ_x) + \text{etc.}] +$

L3 $[mr_2\,(1 - HRO_2) - mr_3]\,[(P_{31}(T_{31} + \Sigma(RBQ) + DBQ_{31} + DBQ_y + DBQ_x) + P_{32}(T_{32} + \Sigma(RBQ) + DBQ_{32} + DBQ_y + DBQ_x) + \text{etc.} +$

L4 $[mr_3 - mr_4]\,[(P_{41}(T_{41} + \Sigma(RBQ) + DBQ_{41} + DBQ_y + DBQ_x + DBQ_z) + P_{42}(T_{42} + \Sigma(RBQ) + DBQ_{42} + DBQ_y + DBQ_x + DBQ_z)$

+ any remaining components.

Figure 5

Representation of queue and delay components for *FCP* for all levels.

but includes the additional delays resulting from the other processors and memory hierarchies. In essence, these other processors and memory structures will only increase the *FCP* of the previous uniprocessor by increasing the average reload time. To include these in the analytical model, it is fundamentally important to understand that we essentially model a multiprocessor as a uniprocessor in a manner similar to that of Figure 2, but incorporate *two additional types* of delays to account for the remainder of the system: 1) additional modules (components) at each level to account for either cross-interrogates (similar to Figure 3) or shared-memory accessing (similar to Figure 4); 2) queue delays for shared resources such as request buses (i.e., address/control), data buses, and, where appropriate, array access. Each additional module represents some other memory hierarchy component introduced by the other multiple processors. These are included in the determination of the total average latency of each level. This is illustrated for a very general case in **Figure 5**. The L2 cache might consist of several modules, as in Figure 3, each having a separate probability and delay per hit. Similarly, L3 might consist of several modules, as in Figure 4. Queues arise because the multiple processors may have simultaneous cache misses scattered at any level or all levels of the hierarchy. As a result, it is possible for multiple requests to occur at any shared resource such as address/control (i.e., request) buses, data buses, or cache arrays. This produces queues at each shared resource, as indicated generally in Figure 5, where *RBQ* refers to request bus queues and *DBQ* refers to data bus queues. Even though the modeling is done independently of the hardware design, it is necessary to be sure that the intended hardware will actually implement these queues; otherwise, a different model is required. Buses may be shared and are usually as narrow (in byte width) as possible in order to reduce cost. This makes the hit reload delay path more complex, as shown by the finite cache penalty contribution of each level of the hierarchy at the bottom of Figure 5. Note that the *FCP* calculation for a multiprocessor has new terms, namely *Q* delays, in the total reload path. Because the queues for the request buses are difficult to illustrate in this type of schematic, they are not fully shown and are indicated only by $\Sigma(RBQ)$, the sum of the appropriate request bus queues. These are illustrated later, in the example. The multiprocessor system also increases the bus electrical and circuit delay

terms, T_2, T_3, etc., which obviously increase the *FCP*. These delays are assumed to be given parameters and are not calculated here. In order to calculate the *FCP* of a multiprocessor system such as that depicted in Figure 4, we proceed as follows:

1. Calculate the probabilities of hits for each cache module of the hierarchy.
2. Calculate queue delays for each bus (requires an assumed initial *CPI* value).
3. Calculate total delay per hit for each module, including multiple queue and bus delays in the total path, as needed.
4. Sum all terms for the total *FCP*, as in Figure 4, and obtain a new *CPI* value.
5. Compare the new, calculated *CPI* value with that assumed for the initial *Q* calculations: If different, redo the *FCP* and *CPI* calculations using a new *CPI* value for *Q* calculations; and repeat until the *CPI* values match sufficiently closely.

The *Q* calculations provide the recursive connection between the delay and the *FCP*, as will be seen. In the following, the general expressions for the probability of hits per module are first derived. Afterward, queue calculations and a specific example are presented.

General hit probabilities and delay categories

The required probabilities are obtained separately below for a private cache with cross-interrogates and address-shared cache.

Hit probabilities for private cache with cross-interrogated data

The L2 shown in Figure 3 represents a general case of the type needed for our analysis. It is necessary to determine the probabilities of accessing each individual module or groups of modules that have a different reload delay from all the rest, within the OtherL2 group in Figure 2. We must first specify the makeup of each module, then determine its probability of hits. This is easily done as follows.

General case: delay categories

For the general case, there can be many L2s on a card and multiple cards in the system, as shown in **Figure 6(a)**. The effective reload delay can be different for each L2, but usually there are groups of L2s which have the same or very nearly the same delay. All of the OtherL2 are thus grouped into a few convenient delay categories, with those in any given category having identical or approximately the same delay, as specified in **Figure 6(b)**. The adjacent L2s, or "AdjL2," are those which are typically the closest, distance- and delay-wise, to the OwnL2. The "OppL2" category contains those which are typically at a greater

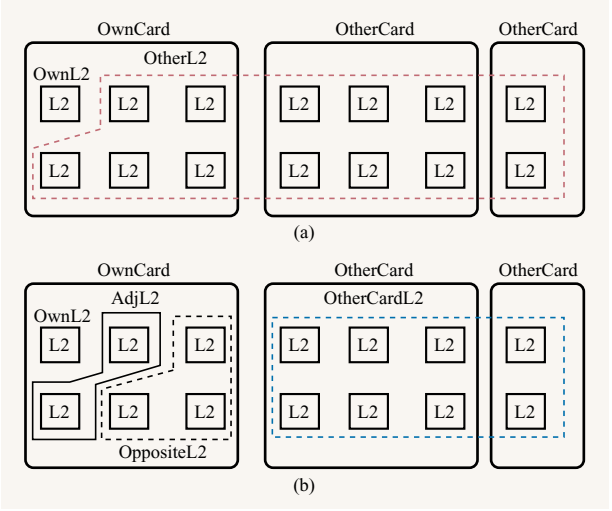


Figure 6

Partitioning of a cross-interrogated L2 into delay categories, each with a separate reload delay and *FCP* contribution: (a) MP system with cross-interrogated L2 showing OwnL2 and OtherL2 categories on all cards; (b) delay categories derived from (a). Each category is assumed to have one fixed reload delay.

diagonal distance and hence have larger delay. If there is more than one L2 in this category, the delay difference between these is often small enough to allow them to be assumed equal. If not, additional delay categories are necessary.

If there are more than two cards present, the difference in delay from the given processor to two L2s on different cards is usually small enough to allow all other cards to be categorized as "OtherCardL2" for *FCP* calculations. If the delay differences to these L2s are significant, additional delay categories are needed.

Using the categories defined above and in Figure 6(b), the general expression for the hits per category requires only determination of the probabilities in Figure 3. For all cases, it is assumed that the accessing patterns are random with uniform distributions, so that any module of equal size has the same probability of being accessed.

Probability of accessing adjacent L2s

There are a total of $\#AdjL2$ modules of L2 adjacent to OwnL2 on any card, so this probability is simply this value divided by the total number of other L2s in the system, or

$$Prob[\text{AdjL2}] = \#AdjL2/\#OtherL2 = \#AdjL2/(Total\#L2 - 1), \quad (13)$$

where

$$\#OtherL2 = Total\#L2 - OwnL2 = (Total\#L2 - 1). \quad (14)$$

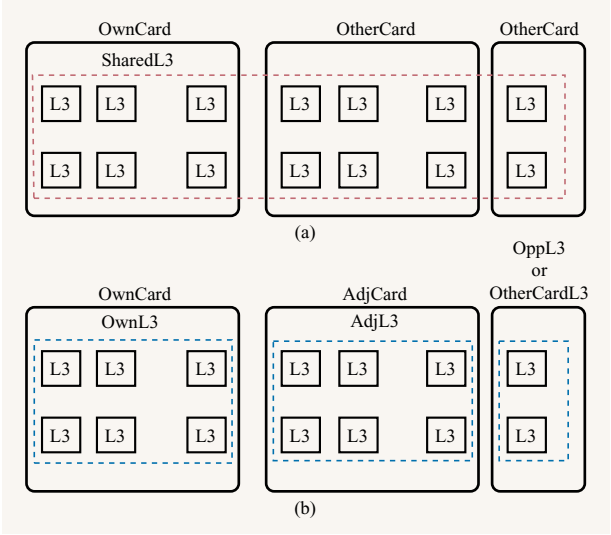


Figure 7

Partitioning of a shared L3 into delay categories, each with a separate reload delay and *FCP* contribution: (a) MP system with shared L3; (b) delay categories derived from (a). Each category is assumed to have one fixed reload delay.

Probability of accessing opposite L2s

There are a total of *#OppL2* modules of L2 either opposite or further from OwnL2 on any card, so this probability is this value divided by the total number of other L2s in the system, or

$$Prob[\text{OppL2}] = \#OppL2/\#OtherL2 = \#OppL2/(Total\#L2 - 1). \quad (15)$$

Probability of accessing L2s on other cards

In direct analogy to the above, assuming that the delay to L2s on other cards is the same for all cards, this probability is

$$Prob[\text{OtherCardL2}] = \#OtherCardL2/\#OtherL2 = \#OtherCardL2/(Total\#L2 - 1) = (Total\#L2 - \#L2perCard)/(Total\#L2 - 1). \quad (16)$$

Hit probabilities for shared L3s

A general case of the type needed for our analysis is represented by L3, shown in Figure 4. It is necessary to determine the probabilities of accessing each individual module or groups of modules that have a different reload delay from all the rest, within the L3 group (Figure 4). Delay categories similar to that above for L2 are used for L3. One important difference is that all misses from OtherL2 are scattered randomly across all L3 modules, unlike L2, where all L1 misses are directed first to OwnL2, and only misses out of OwnL2 are scattered across OtherL2. Thus, the probability expressions for L3 typically have in the denominator a term *Total#L3* in place of the term (*Total#L2* − 1) in Equations (13) and (16).

The total shared L3 is illustrated for a general case in **Figure 7(a)**, with several possible delay categories shown in **Figure 7(b)**. It is assumed that the CPU which creates this path is on the card OwnL3. Typically, if there are multiple L3s on one level of a package, such as a card, the total effective reload delay is the same, or nearly so. Thus, all L3 modules at this level, i.e., card in this case, are categorized as OwnL3 with a fixed delay. If this is not true, additional delay categories may be needed. If the cards are arranged somewhat as shown, delay to the L3s on the adjacent card will be larger than to those on the "OwnCard," and hence constitute another category, *AdjL3*. The L3s on cards farther away will have even larger reload delays and provide additional categories. Depending on the physical arrangement, we may have cards facing opposite the OwnCard, or in Other positions, as indicated. The probability determination is straightforward for any new cases. It is assumed, as previously, that all L3 modules in Figure 7(b) are of the same size and equally divided across the full address space. Thus, the probability of accessing any one given L3 module, no matter where it resides, is 1/*Total#L3*. The probability of accessing any given delay category is simply the number of L3 modules in this category times 1/*Total#L3*.

Probability of accessing OwnL3

There are *#L3perCard* accessible modules on any card, so the probability is this value divided by the total number of address-sliced L3 in the system, or

$$Prob[\text{OwnL3}] = \#L3perCard/Total\#L3. \quad (17)$$

Probability of accessing adjacent L3s

If the adjacent L3s consist of several cards, *#AdjCards*, the probability is this number of cards times Equation (17), or

$$Prob[\text{AdjL3}] = \#AdjCards \times \#L3perCard/Total\#L3. \quad (18)$$

Probability of accessing opposite L3s

This is Equation (18) with the parameter *#AdjCards* replaced by parameter *#OppCards*, giving

$$Prob[\text{OppL3}] = \#OppCards \times \#L3perCard/Total\#L3. \quad (19)$$

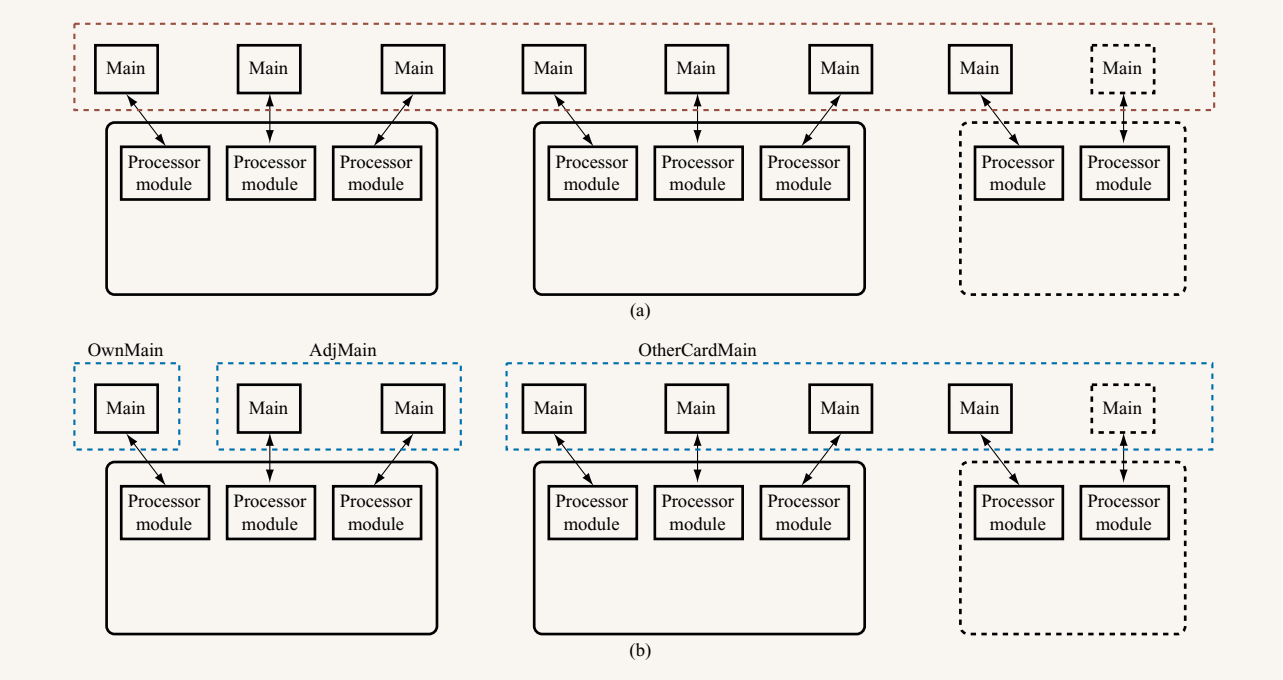


Figure 8

Partitioning of a shared main memory into delay categories, each with a separate reload delay and *FCP* contribution: (a) MP system with shared main memory; (b) delay categories derived from (a). Each category is assumed to have one fixed reload delay.

Probability of accessing L3s on other cards
This is given by Equation (19) with the parameter *#AdjCards* replaced by *#OtherCards*:

$$Prob[\text{OtherCardL3}] = \#OtherCards \times \#L3perCard/Total\#L3. \quad (20)$$

Hit probabilities for shared main memory
Hit probabilities for shared main memory (or, simply, "main") are completely analogous to those for shared L3s, except that the delay categories are defined slightly differently. As with L3, main memory is assumed to consist of several separate self-contained modules, with each module containing one slice of the full address space, each of equal size. It is also assumed that each main slice is associated with one processor module; i.e., access to any main module must first proceed through the associated processor module. A general case might be similar to that in **Figure 8(a)**, in which each processor module is closely connected to one module of main, its "OwnMain," with an access/reload time much smaller than to any other main slice. Thus, OwnMain is one delay category, as indicated in **Figure 8(b)**. The other main memory slices associated with processor modules on the same packaging level (card) will have longer delays, but typically nearly equal to one another. Main memory slices associated with these are grouped together as "AdjMain," all with the same delay for reloads to the processors/L1 (on processor module 1 connected to OwnMain), as shown in Figure 8(b). All other main slices on other cards have very nearly the same access/reload delay and are thus grouped together in the delay category "OtherCardMain," as shown in the figure. The hit probabilities for these delay categories are analogous to those for L3 and are given below. Since it is assumed that the main memory miss rate is 0, the number of hits for all cases is given by the probability multiplied by mr_3, as can be seen from Figures 2 or 5, by letting $mr_4 = 0$.

Probability of accessing OwnMain

$$Prob[\text{OwnMain}] = \#OwnMain/Total\#Main. \quad (21)$$

Probability of accessing AdjMain

$$Prob[\text{AdjMain}] = \#AdjMain/Total\#Main. \quad (22)$$

Probability of accessing OtherCardMain

$$Prob[\text{OtherCardMain}] = \#OtherCardMain/Total\#Main. \quad (23)$$

The hit probabilities derived above will be useful in the following queue calculations.

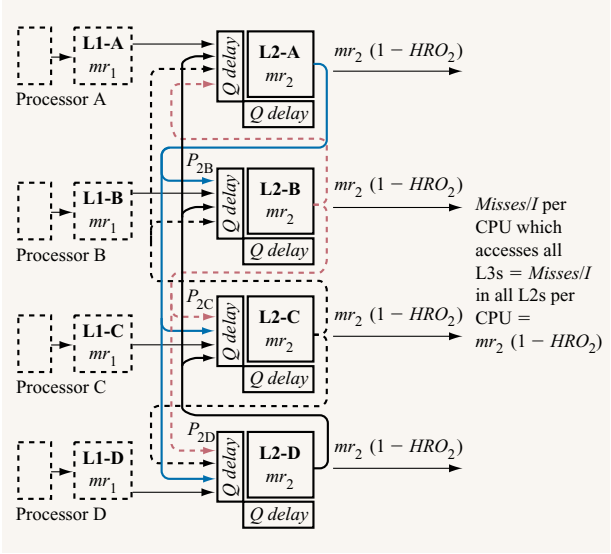


Figure 9

Representation of component of *FCP* for cross-interrogates in other modules of private L2s.

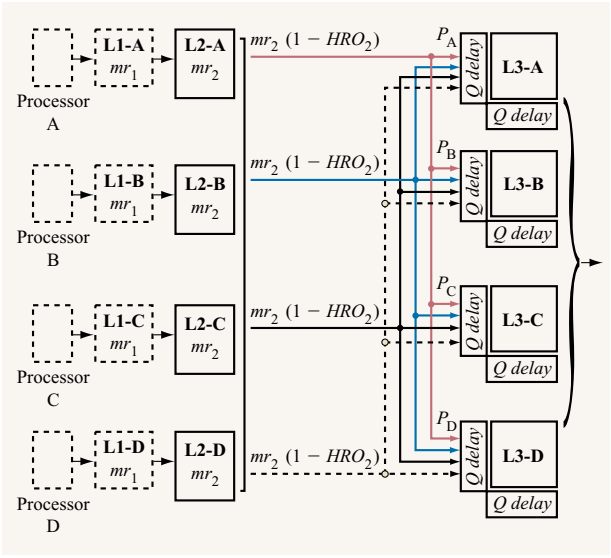


Figure 10

Representation of component of *FCP* for level L3 of addressed-sliced, shared memory.

General queue delays

Readers not familiar with queuing theory can acquire all needed fundamentals from Appendix A. Queues arise as follows. In Figure 3, only one processor is shown accessing its OwnL2, with misses propagating to the OtherL2 of the system. A more detailed configuration for a multiprocessor system is shown in **Figure 9**, in which each processor can be accessing its OwnL2, have a miss, and require a cross-interrogate access to the other L2s. These additional processors affect only the original L2 of Figure 2 by making simultaneous requests, as indicated. Thus, they are included in the original uniprocessor model only as a queue delay at the appropriate points. Some such points are shown in Figure 5 and can occur at the input to request buses (address/control buses which send and communicate the reload requests), DRAM array accesses for reasons discussed later, and at the input to data buses.

In a similar manner, Figure 4 shows only one processor accessing the shared L3 with misses propagating to L4. A more detailed configuration is shown in **Figure 10**, again indicating that multiple requests occur on buses and arrays. Figure 5 shows some of the request and data bus queues and how they might contribute to the total *FCP*. The final structure is very dependent on the specifics of the system (a detailed example is given later, in Figure 13). These are the queues which must be calculated. In order to do this, every possible activity on the bus or array must first be identified; then, request rates, utilizations, and finally queue delays for each component are calculated. The total utilization, U, also allows determination of the average length of the queue using Equation (A6). From Equation (A5a), any resource with a constant service time and having a total average U of about 73% will have an average queue length of one or more, increasing rapidly as U increases. Generalized queue categories are not readily definable because queues are so dependent on the hierarchy details, particularly the busing structure. Queue calculations are best understood by example, as shown below.

7. Multiprocessor example

One specific example of a general MP is shown in **Figure 11**. The basic building block is the 4-way card. It is assumed to contain four processors (CPUs), each with its own private, nonshared L1 cache (i.e., no cross-interrogates to L1, only to L2). Two processors with L1s are packaged on an assumed processor module, along with a semiprivate, inclusive L2[12] which is electrically very close, as indicated. Misses from only these two L1s can make direct (non-cross-interrogate) access to the semiprivate L2. It is assumed that shared data can be stored in L2, so the L2 levels of other processor modules require cross-interrogation for cache coherency, as discussed later. The two processor modules on the 4-way card are connected by means of buses which connect through a switch and routing logic function shown in Figure 11. These buses have separate request (address/control) buses as shown,

[12] Use of the term *inclusive L2* implies that any block in L1 is also in L2 (locked). Thus, any block reloaded to L1 must also be loaded to L2. This is an assumption, and is often used in such a system.

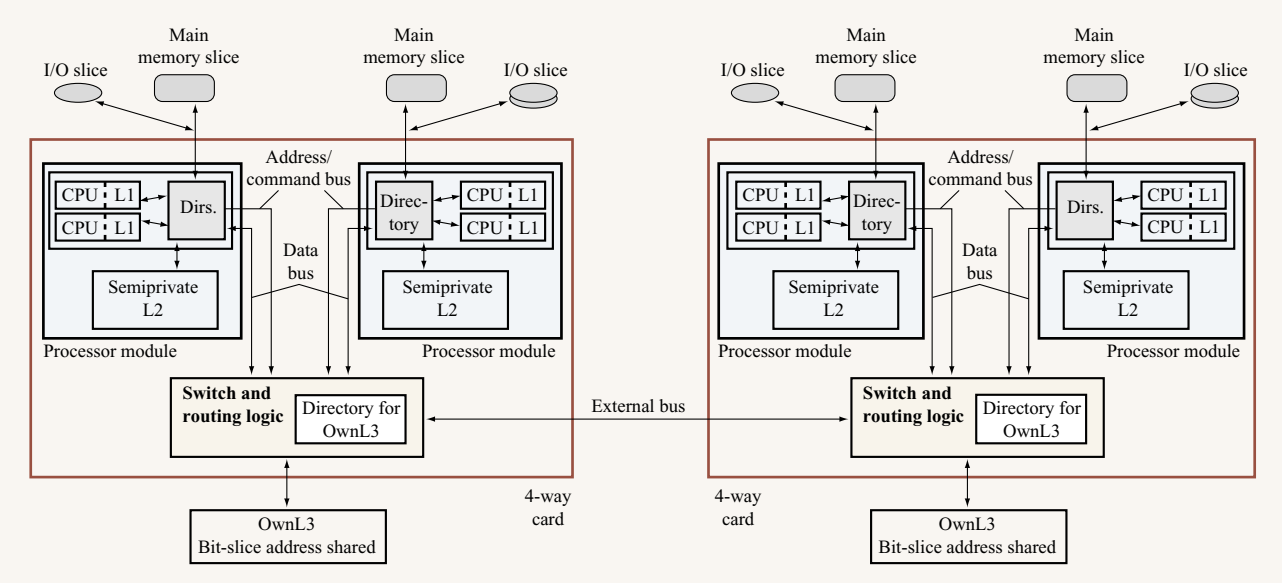


Figure 11

Generalized 8-way multiprocessor configured from two separate 4-way cards, having private, semiprivate, and globally shared, address-sliced levels.

and either unidirectional or bidirectional data buses to/from each processor module. In the model structure, the choice between a unidirectional and a bidirectional bus affects only the terms in the bus queue delay. A bidirectional bus must include request rates in both directions, and thus has higher utilization. In addition, bidirectional buses generally have longer latencies because of electrical turnaround time. This switch also connects to an L3 which is assumed to be address-sliced-shared such that all L3s in the system contain only part of the full address space. A larger system is built by connecting multiple 4-way MP cards using the external bus, as shown for the 8-way system of Figure 11. The bus interconnection is achieved by means of the switch function, but other methods are also possible.[13] It is assumed in all cases that a hit at any given level will be reloaded to *all upstream* levels (e.g., a hit in L3 due to an L1 miss causes the reloading block to be put into L2 as well, the L2 being upstream of L3). All cache levels are assumed to be store-in, thus requiring some castouts which are a fixed percentage of the misses. It is further assumed that all I/O slices and main memory slices are connected via one data bus to the processor module interface unit as shown in Figure 11, with one I/O slice and one memory slice per processor module (i.e., per two CPUs). The general techniques and equations previously derived are now applied to this model.

[13] For example, the IBM POWER4 GHz Processor [8] uses a ring-type bus structure.

Full-model FCP path

The final model for the *FCP* calculation of the example of Figure 11 is similar to Figure 5, in which the L2 and L2-XI are of the form shown in Figure 3, and the L3 cache as well as main are of the form shown in Figure 4, with main memory replacing L4. Modules and buses have various queues which depend on the detailed configuration. **Figure 12** maps the general delay categories of Figure 5 into the specific categories defined above in Section 6. The equations expressing the probability of hits-per-instruction are shown in Figure 12 for each of these delay categories for both the 4-way (one card) and 8-way configurations. The more complex 8-way parameter is derived below using the relations derived previously. Evaluation of the accompanying queue delays for some request bus, data bus, and appropriate array access queue, as well as total path delays, will be discussed afterward.

Hit probabilities and hits/instruction for MP example with two cards (8-way MP)

Hits/instruction for L2 delay categories using hit probability equations

OwnL2

Regardless of whether one or two cards are used, there is always only one OwnL2, as shown in Figure 12, with the *hits/I* contribution to *FCP* given by

$$Hits/I[\text{OwnL2, 4- or 8-way}] = mr_1 - mr_2 . \quad (24)$$

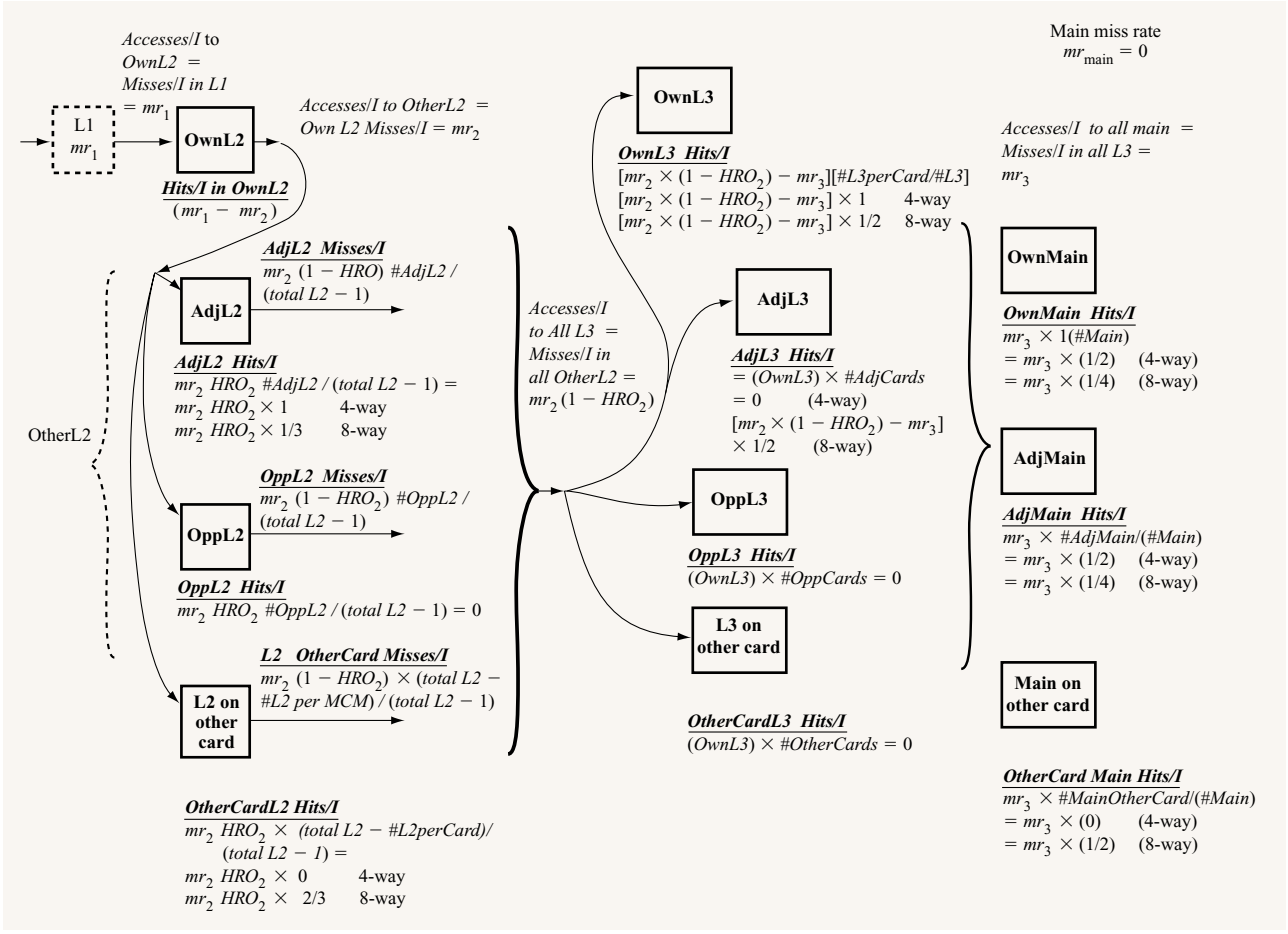


Figure 12

Progression of hits and misses per instruction through entire 4-way or 8-way MP memory hierarchy example used for *FCP* calculation.

AdjL2 and OtherCardL2

As indicated in Figure 12, there are a total of four L2, of which three are OtherL2. One is the AdjL2, and the other two, which have identical reload delays, are on the second card in the category OtherCardL2. Assuming randomly scattered addresses, the probability of accessing any of these three OtherL2 is 1/3. From Equation (13), the probability P_{Adj} of accessing AdjL2 in Figure 12 is now 1/3. Since category OtherCardL2 contains two such modules, each with probability of 1/3, its probability of being accessed is 2/3. This is in agreement with Equation (16). Thus, the new *Hits*/*I* components for *FCP* contributions are

$$Hits/I[\text{AdjL2, 8-way}] = mr_2\, HRO_2/3 \tag{25}$$

and

$$Hits/I[\text{OtherCardL2, 8-way}] = mr_2\, HRO_2\, 2/3. \tag{26}$$

Hits/instruction for shared L3 delay categories of MP example

As indicated in Figures 11 and 12, there are two L3 modules in the full system. Thus, $OwnL3 = 1$, $AdjL3 = 1$, and $Total\#L3 = 2$, so by Equation (17),

$$Prob[\text{OwnL3, 8-way}] = 1/2, \tag{27}$$

and the *Hits*/*I* component for its *FCP* contribution is given by

$$Hits/I[\text{OwnL3, 8-way}] = [mr_2(1 - HRO_2) - mr_3]1/2. \tag{28}$$

Comparable terms for the *AdjL3* component from Equation (18) are

$$Prob[\text{AdjL3, 8-way}] = 1/2, \tag{29}$$

and the *Hits*/*I* component for its *FCP* contribution is given by

$$Hits/I[\text{AdjL3, 8-way}] = [mr_2(1 - HRO_2) - mr_3]1/2. \tag{30}$$

There are no OppositeL3 nor OtherL3 for this example, so these terms equal 0. Note, as required, that the sum of all probabilities for each cache level (4-way or 8-way) must equal 1.

The hits/I for main memory delay categories of our multiprocessor example are obtained in a manner similar to that above for L3, using the equations previously derived for these delay categories. All hit components for both 4-way and 8-way configurations are summarized in Figure 12; also shown, for completeness, are some categories defined earlier which are irrelevant for this example but which might be required for more complex cases. Each of these individual HIT components must be multiplied by the total effective reload time for each path. The path delays, which must include all queues, latency, and trailing-edge terms, are derived separately below, after the queue evaluations.

Queue calculations for MP example

For the example of Figure 11, all queue delays needed for determining the total effective reload delay of each delay category are shown in **Figure 13**. Request bus queues, or RBQs, indicate the points at which multiple sources compete for use of that bus, in order to send a reload or other request. Array access queues, or AAQs, appear only at the input to DRAM arrays, and represent the point at which multiple sources compete for use of the array.[14] In this example, L2 is assumed to be SRAM and L3 is DRAM, so L2 has no AAQ. Data bus queues, or DBQs, are on the inputs to all buses that can have multiple pending requests for data transfer. Obviously, because of symmetry, many of the queues will be identical. To obtain the queue delays, it is necessary to calculate the utilization U of each of these resources, which first requires the specification of all traffic on each. Figure 13 shows the traffic on the major buses. We include the utilization and queue calculations for only several resources, since the details become somewhat repetitive and tedious—extensions to remaining and other cases should be obvious. The following queue calculations are done only for the 2-card 8-way MP of Figure 10, each card having the same queue structure as Figure 13, connected by way of the external bus.

Processor module internal queues

Queues for the processor module of our example are determined using the general expressions previously determined in Section 6.

[14] SRAM array cycle times are typically too small to cause any significant queue. DRAM array cycle plus refresh times can be significant and create queues.

Queues associated with L2

For the configuration as shown in Figure 13, the L2 directory is located within the bus-interface unit which receives all L2 traffic for both the directory and array. As a result, the traffic to the directory and array will be somewhat different, as detailed below. We consider only requests to the array which involves the request bus RBL2 and data bus DBL2, with associated queues RBQL2 and DBQL2, respectively, as well as L2 array queue AAQL2. All requests for reloads, castout fetches (castouts require separate array fetches), or stores to L2 use the request bus and array. The data bus, if bidirectional, would do likewise, but is assumed to be unidirectional as shown. The separate events which occur on these buses are shown for this specific MP example in Figure 13 and occur as follows.

L2 request bus RBL2 and queue RBQL2

- *OwnL2 hits—reload* Each processor module has two L1s (designated in Figure 13 as CPU/L1), each making requests which hit at the rate given by Equation (24), or $(mr_1 - mr_2)$ Hits/I. These are processed at the rate of $1/CPI$ instructions per cycle. The bus service time is given as S_{ta2}; thus,

$$U[\text{OwnL2}] = 2\ Hits/I[\text{OwnL2}]S_{ta2}/CPI = 2\ (mr_1 - mr_2)S_{ta2}/CPI. \quad (31)$$

- *OwnL2 castouts, CO, to L3 due to OwnL2 misses* All castouts are assumed to be 1/3 of misses. The OwnL2 miss rate is mr_2 per processor, so this component is

$$U[\text{OwnL2 CO}] = 1/3\ (2\ mr_2\ S_{ta2})/CPI. \quad (32)$$

- *OtherL2 cross-interrogate hits in OwnL2* All processors other than the two used above will perform cross-interrogates (XI) to this L2. In this 8-way MP, there are $(8 - 2) = 6$ other processors, and their XI hits will be spread out evenly to $(4 - 1)$ or 3 OtherL2, of which the current OwnL2 represent one of these. Thus, the probability of any one of these processors performing an XI access to this OwnL2 is 1/3, and we thus have six processors accessing each OtherL2, and $P = 1/3$, $mr_{x-1} = mr_2$, $HR_x = HRO_2$, giving

$$U[\text{XI from OtherL2}] = 6(1/3)\ mr_2\ HRO_2\ S_{ta2}/CPI. \quad (33)$$

- *Castouts to L2 from this processor's own L1* L1 is assumed to be a store-in cache, thus requiring castouts. All castouts are assumed to be 1/3 of the L1 misses. The L1 miss rate is mr_1 per processor, and there are two processors and two L1s, so this component is

$$U[\text{L1 CO to L2}] = 1/3\ (2\ mr_1\ S_{ta2})/CPI. \quad (34)$$

- *Reloads for OwnL2 misses* If a miss in either of the two L1 caches associated with an OwnL2 incurs a

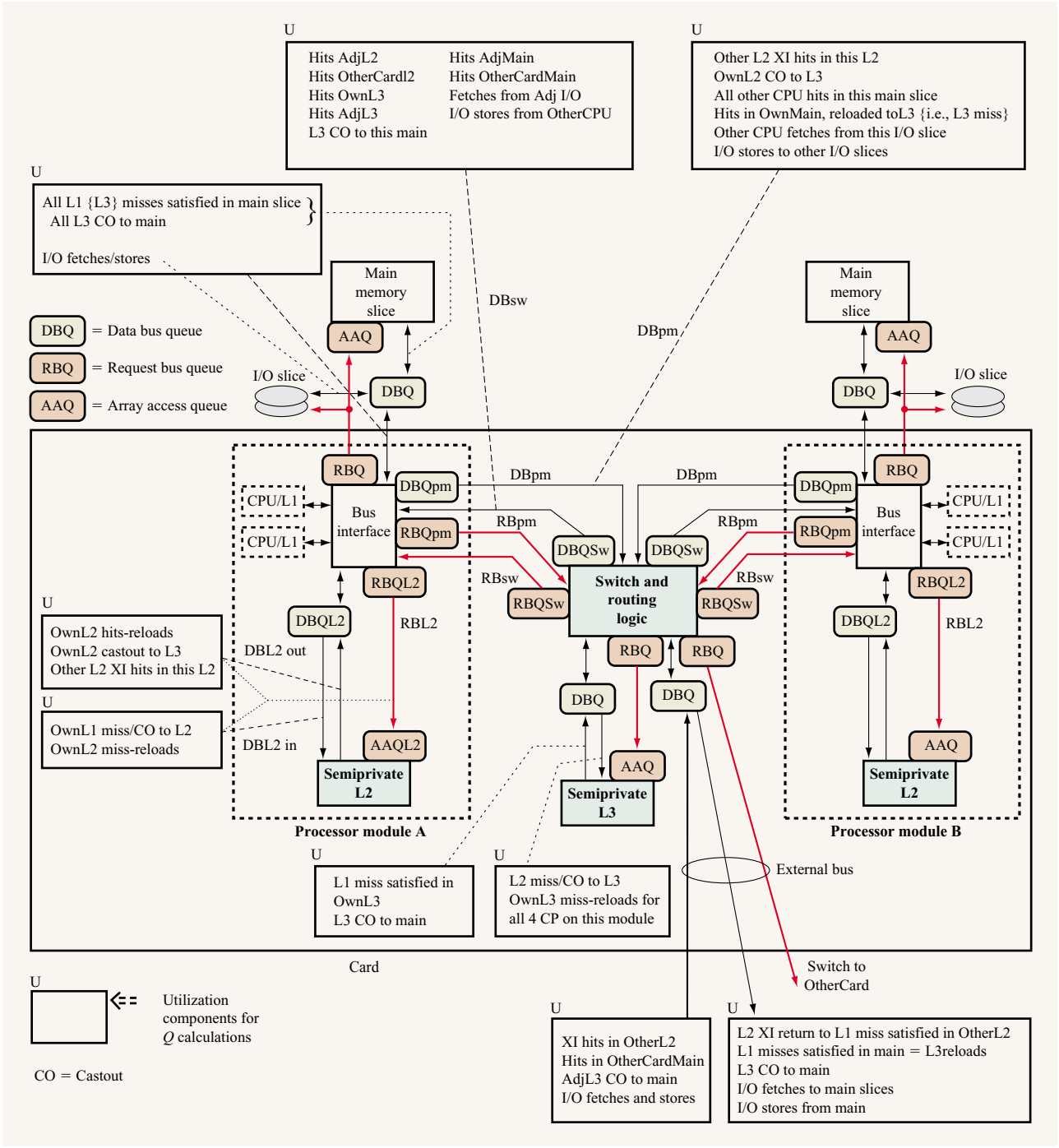


Figure 13

Data bus, request bus, and DRAM array access utilization components and queues for 4-way MP card.

subsequent miss in the L2 directory/logic (in the bus interface unit), a request must be sent to the L2 array to initiate the reloading process. This will occur for all misses in L2 due to its two associated processors accessing L2, each with a miss rate of mr_2. Thus, this component is

$$U[\text{OwnL2 misses}] = 2\, mr_2\, S_{ta2}/CPI. \quad (35)$$

The above five components of Equations (31) through (35) represent all utilization of request bus RBL2. They must be added in order to obtain the total utilization, or

$$U_{\mathrm{RBL2}} = \text{Equation (31)} + \text{Equation (32)} + \text{Equation (33)} + \text{Equation (34)} + \text{Equation (35)}. \quad (36)$$

The actual Q wait delay which will be needed in the final *FCP* calculation is given by Equation (A3) with S_{ta2} used as the service time, or

$$RBQL2 = 0.5 \times S_{\mathrm{tL2}} \frac{U_{\mathrm{RBL2}}}{1 - U_{\mathrm{RBL2}}}. \quad (37)$$

L2 array queue

If SRAM rather than DRAM is used for L2, the array queue will usually be negligible. If DRAM is used, the queue should be evaluated as follows. The requests which appear on the request bus above obviously appear at the input to the L2 array. Thus, the utilization components are the same *except* that the bus cycle time is replaced by the array cycle time, T_{cL2}. The effective DRAM cycle time should include a weighed average contribution due to refresh time, which is a function of the specific chips used. Obviously, the different service time between request bus and array can make the array queue AAQL2 different from the RBQL2.

L2 data bus queues

The data buses DBL2-in and DBL2-out are shown separately, both to clarify the utilizations and to show the procedure if the data bus is actually two unidirectional buses. Obviously, the components on these buses must be responses to the input requests on the request bus and are divided as indicated in Figure 13. The utilization and queue calculations will be similar but require a modified service time. The utilization components of the data-out bus, DBL2-out, are given by Equations (31), (32), and (33), but with service time given by

DBL2 service time

$$= S_{\mathrm{DBL2}} = T_{\mathrm{DBL2}}(\#Bytes/Block)/(\#Bytes/DataBus), \quad (38)$$

where T_{DBL2} = data bus cycle time. If T_{DBL2} = 3 processor cycles,[15] $\#Bytes/Block = 64$, and $\#Bytes/DataBus = 16$, then $S_{\mathrm{DBL2}} = 12$ processor cycles per request.

Utilization of the data-in bus is obtained in a similar manner. The Q delay for each of these is obtained from Equation (A3), using the service time of Equation (38).

[15] All delays are normalized in units of "number of processor cycles."

Processor module external bus queues

This analysis is limited to only one major bus which has complex utilization components, namely the *data bus between switch and processor module*. This bus, as shown in Figure 13, is assumed to consist of two separate unidirectional buses, DBsw and DBpm. Detailed calculations are performed only for the DBsw, which carries data from the switch to the processor module. If only one bidirectional data bus is used, the two separate bus utilization components must be added. The individual activities which occur on bus DBsw are expressed as follows.

DBsw—data bus from switch to processor module

The service time of this bus is assumed to be twelve processor cycles/request. Data coming into processor module A from the switch in Figure 13 will be all requests from the two L1s on module A which miss their OwnL2 except hits to their OwnMain, plus L3 castouts to the main slice attached to processor module A, plus all other processor I/O stores to the I/O slice attached to processor module A. These are broken into finer, individual categories of utilization, as defined in Figure 13:

- *Hits in AdjL2* Request rate *RR* is the hits/I in adjacent L2s from Equation (26) multiplied by two processors, divided by *CPI*, or

$$RR[\mathrm{AdjL2}] = 2\,Hits[\mathrm{AdjL2}]/CPI = 2\,mr_2\,HRO_2/(3\,CPI). \quad (39)$$

  The utilization component is

$$U[\mathrm{AdjL2}] = S_{\mathrm{DBsw}}RR = 12 \times 2\,mr_2\,HRO_2/(3\,CPI) = 8\,mr_2\,HRO_2/CPI. \quad (40)$$

- All utilization component calculations are similar to that above, using the same service time. It is only necessary to derive the correct *RR* for each component, as shown below.
- *Hits in OtherCardL2* *RR*[OtherCardL2] requires substitution of *Hits*[OtherCardL2] given by Equation (26) for *Hits*[AdjL2] in Equation (39) to obtain

$$RR[\mathrm{OtherCardL2}] = 2\,Hits[\mathrm{OtherCardL2}]/CPI = 2\,mr_2\,HRO_2\,2/(3\,CPI). \quad (41)$$

  The utilization component is

$$U[\mathrm{OtherCardL2}] = S_{\mathrm{DBsw}}RR[\mathrm{OtherCardL2}] = 12 \times 4\,mr_2\,HRO_2/(3\,CPI) = 16\,mr_2\,HRO_2/CPI. \quad (42)$$

- *Hits in OwnL3* *RR*[OwnL3] requires substitution of *Hits*[OwnL3] given by Equation (28) for *Hits*[AdjL2] in Equations (39) and (40) to obtain

$$U[\text{OwnL3}] = S_{\text{DBsw}}\, RR[\text{OwnL3}]$$
$$= 12 \times 2\,[mr_2(1 - HRO_2) - mr_3]\,1/2\,(1/CPI)$$
$$= 12\,[mr_2(1 - HRO_2) - mr_3]/CPI. \quad (43)$$

- *Hits in AdjL3* Analogous to that above, using Equation (30) for the hit rate gives

$$U[\text{AdjL3}] = S_{\text{DBsw}} RR[\text{AdjL3}]$$
$$= 12 \times 2\,[mr_2(1 - HRO_2) - mr_3]\,1/2\,(1/CPI)$$
$$= 12\,[mr_2(1 - HRO_2) - mr_3]/CPI. \quad (44)$$

Note that there are no Opposite L3 nor OtherCard L3.

Castouts from all L3 to this main memory slice
Castouts are taken as 1/3 of all misses. We first calculated all misses, scattered them equally to all main memory slices, and multiplied by 0.333. There are two L3 slices, with a combined miss rate of mr_3, so we may consider these as one memory with miss rate mr_3 per processor. The eight processors miss the memory at this rate, and the castouts so produced are scattered over four main slices, so the *RR* component for these castouts to main slice A is

$$RR[\text{L3CO}] = 0.333 \times 8\, mr_3/(4\, CPI) = 0.333\,(2\, mr_3/CPI). \quad (45)$$

The utilization contribution is

$$U[\text{L3CO}] = 12 \times 0.333\,(2\, mr_3/CPI) = 8\, mr_3/CPI. \quad (46)$$

Note that since main memory has no misses, all L3 misses are main hits:

- *Hits in AdjMain* *RR*[AdjMain] requires substitution of *Hits*[AdjMain] for *Hits*[AdjL2] in Equation (39) to obtain

$$RR[AdjMain] = 2\, Hits[\text{AdjMain}]/CPI = 2\, mr_3/(4\, CPI). \quad (47)$$

- The utilization component is

$$U[\text{AdjMain}] = S_{\text{DBsw}} RR[\text{AdjMain}]$$
$$= 12\, mr_3/(2\, CPI) = 6\, mr_3/CPI. \quad (48)$$

- *Hits in OtherCardMain* Analogous to that above, using Equation (44) for the hit rate gives

$$U[\text{OtherCardMain}] = S_{\text{DBsw}} RR[\text{OtherCardMain}]$$
$$= 12 \times 2\, mr_3/(2\, CPI) = 12\, mr_3/CPI. \quad (49)$$

I/O contribution to queues
In order to include I/O fetches and stores, we assume some fixed number of I/O fetches per instruction and I/O stores per instruction, as for a miss rate. The I/O fetches only pass data from the I/O modules to main memory, and I/O stores only pass data from main memory to I/O over all appropriate buses. All I/O modules are considered as one large, shared resource similar to main memory. Thus, the I/O fetch requests from any one processor are randomly accessed from all I/O modules, with block transfers randomly scattered across all main memory modules.

Fundamentally, there are two components to I/O traffic, one due to page swapping (assuming a paged virtual memory environment) and the other due to normal I/Os. In systems with small main memory, the paging I/O can be significant and will vary with main size. However, the normal I/O rates are independent of main memory size and correlate very well with the instructions executed (see [9], p. 38). For systems with large main memory, the paging I/O is typically negligible and is the inherent assumption implied in the above MP model. If page-swapping I/O is significant, the modeler can either use a slightly different model which includes an effective miss rate for main memory to represent the paging I/O rate, or simply increase the above I/O rate to account for the additional activity.

Fetches from adjacent I/Os
I/O fetches per instruction executed, *IOF*/*I*, are assumed to occur at a fixed average rate of $IOF/I \sim 1/1500$ per processor, meaning that each processor makes this number of requests per instruction executed, I_E, to all I/O modules.[16] There are two processors on the processor module, each requesting *IOF*/*I*, and these are scattered to four I/O modules. Three of these I/O will supply fetches over this bus, so the request rate is

$$RR[\text{IOF}] = 2 \times \frac{3\, IOF/I}{4\, CPI} = \frac{3\, IFO/I}{2\, CPI}, \quad (50)$$

and utilization is

$$U[\text{IOF}] = 12 \times \frac{3\, IOF/I}{2\, CPI} = 18\, \frac{IOF/I}{CPI}. \quad (51)$$

Stores from all other processors to this I/O
I/O stores are assumed to occur at the same fixed average rate of $IOF/I = IOS/I \sim 1/1500$. There are six other processors, each performing stores at this rate, scattered over all four I/O modules. Thus the request rate to the one of four I/Os which uses this bus for such stores is

$$RR[\text{IOS}] = \frac{6\, IOF/I}{4\, CPI} = \frac{3\, IFO/I}{2\, CPI}, \quad (52)$$

and the utilization is

[16] Note that this is similar to mr_3, which is the miss rate per processor to all slices of L3 considered as one memory.

$$U[\mathrm{IOS}] = 12 \times \frac{3\ IOF/I}{2\ CPI} = 18\ \frac{IOF/I}{CPI}. \qquad (53)$$

The total utilization of this bus, as before, is the sum of all of the above components, and the average queue is obtained from Equation (A3).

Other buses

The individual utilization components on the DBpm data bus from processor module to switch are listed in Figure 13, as well as those on several other buses. Each component for each bus is calculated in a manner analogous to that above, and all queues illustrated in Figure 13 are determined. Once this is completed, the final *FCP* can be determined.

Summing path delays and calculating the FCP

The final finite cache penalty is calculated in a manner illustrated in Figure 5. Each delay category as defined above, or as defined for a different configuration, has its own total path delay for hits at that level, consisting of various bus queues, array queues if necessary, electrical wiring and logic circuit delays, and trailing-edge delays on data buses. For our example, the delay categories and hit probabilities are shown in Figure 12, with bus queues illustrated in Figure 13. The path delay for OwnL2 is expressed as follows, assuming that the L2 consists of SRAM chips and is sufficiently fast that the array queue, AAQL2 in Figure 13, is not seen:

OwnL2 FCP contribution

$$= (mr_1 - mr_2)(RBQL2 + DBQL2 + T_{\mathrm{RBL2}} + T_{\mathrm{DBL2}} + TE_{\mathrm{DBL2}} + T_{\mathrm{L2}}), \qquad (54)$$

where

$RBQL2$ = queue on request bus RBL2 as determined above;

$DBQL2$ = queue on data bus DBL2 as determined above (if a single bidirectional bus is used, the queue of the combined bus is used; i.e., individual utilizations must be added);

T_{RBL2} = total electrical wiring and circuit delay on request bus RBL2;

T_{DBL2} = total electrical wiring and circuit delay on data bus DBL2 (larger value for a bidirectional bus);

T_{L2} = L2 array access time;

TE_{DBL2} = trailing-edge delays on data bus DBL2 as calculated according to Equation (7).

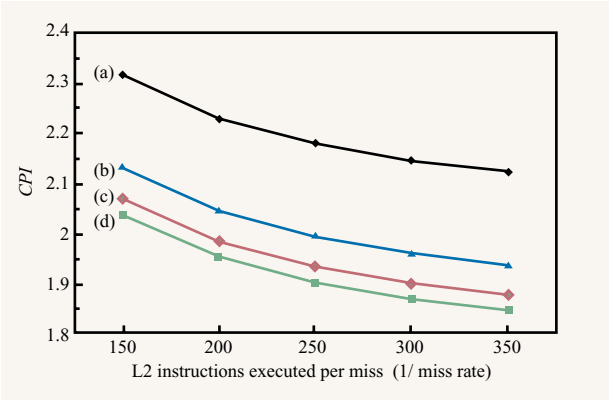


Figure 14

Calculated CPI vs. reciprocal of L2 miss rate, for a multiprocessor similar to that of example, with $mr_3 = 0.02\%$ and L1 miss rates (mr_1) of (a) 6.6%; (b) 3.3%; (c) 2.2%; (d) 1.7%.

Similarly, again using Figures 12 and 13, and recalling that AdjL2 is the L2 on the other processor module of the same card,

AdjL2 FCP contribution

$$= mr_2(HRO_2/3)(RBQpm + RBQsw + RBQL2 + DBQL2 + DBQpm + DBQsw + T_{\mathrm{RBpm}} + T_{\mathrm{RBsw}} + T_{\mathrm{RBL2}} + T_{\mathrm{DBL2}} + TE_{\mathrm{DBL2}} + T_{\mathrm{DBpm}} + TE_{\mathrm{DBpm}} + T_{\mathrm{DBsw}} + TE_{\mathrm{DBsw}} + T_{\mathrm{L2}}). \qquad (55)$$

Note that request buses are used only in the forward direction. Any returning signals are usually transferred by the use of dedicated control lines which are not modeled. If backward traffic is needed and significant, it should be included.

In a similar manner, the *FCP* contributions of each delay category for each level in Figure 12 are determined by multiplying the appropriate hit probability by the sum of the queues and delays in Figure 13, along each path. For paths involving a DRAM array which connects to a data bus, and in which both the array and bus have a queue, these two tandem queues generally cannot be added. Rather, only the maximum one of the two is used, as discussed in Appendix A, section on tandem queues. The final *FCP* is the sum of all of these contributions, as illustrated in Figure 5. The final system *CPI* is the sum of this *FCP* and the given *CPI* [infinite cache], as in Equation (1). Recall that in order to determine the queues, we had to assume some initial value for this system *CPI*. The new value of *CPI* will likely be different from the initial value. Thus, it is necessary to recalculate all of the utilizations and queues using this new value, and find a new *CPI*, as

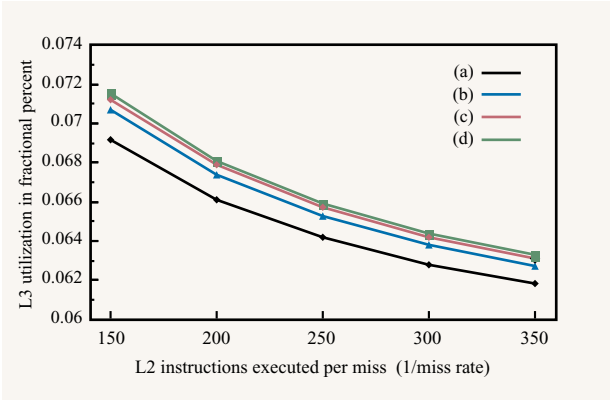


Figure 15

Calculated L3 cache utilization vs. reciprocal of L2 miss rate for a multiprocessor similar to that of example, with mr_3 = 0.02% and L1 miss rates of (a) 6.6%; (b) 3.3%; (c) 2.2%; (d) 1.7%.

above. The process is repeated until the new and previous *CPI* are sufficiently close (about ten recalculations maximum). At this point, the analysis is complete. A spreadsheet or any mathematical analysis tool with recalculation, recursion, or a simultaneous equation solver can conveniently implement the above calculations.

8. Results

Performance results for one system which was similar to the above example are summarized in **Figures 14** and **15**. The total system *CPI* (Figure 14) and L3 cache utilization (Figure 15) are plotted vs. the reciprocal of the L2 cache miss rate for the L3 miss rate fixed at 0.2%, and L1 miss rates of 6.6, 3.3, 2.2, and 1.7%. By the use of Script programming in a spreadsheet, all of these curves and others can easily be generated. By varying various input parameters such as bus width and speed, memory delays, and sizes (via miss rates), the sensitivity of the design to these parameters can be studied. For instance, in Figure 14, the *CPI* for this example varies slowly when the L1 miss rate changes from 3.3% to 1.7%, but increases dramatically at 6.6%. Such sensitivity results are essential for critical design choices.

9. Conclusions

Analytical analyses of memory hierarchies which include queuing delays can provide early design tradeoffs for systems ranging from simple to many-way multiprocessors. Complex behavior of multiple-level memory hierarchies with cross-interrogation can be included with simple statistical parameters and appropriate delays. Even though the correct value of the system *CPI* may not be obtainable to the percentage of accuracy desired, the performance trends obtained as parameters are changed are quite accurate. Thus, if the memory hierarchy configuration is reasonably complete and characterized with appropriate statistics, an analytical analysis can quickly provide relative comparisons as well as sensitivities. For a given system, the sizes and speed of various caches, width and delay of data buses, cross-interrogation rates, etc. can be changed to determine the sensitivity of utilizations and *CPI* to each parameter. In addition, changes in the design, such as busing and memory configuration, can be modeled considerably faster than with a simulation model. Thus, critical bottlenecks or poor designs can often be uncovered and corrected in the initial design stage. Such early analysis tools are an indispensable part of the overall design optimization and are becoming increasingly important as we migrate to larger multiprocessors and complex systems on a chip.

Appendix A: Queuing theory fundamentals for analytical analysis

The key parameter needed for determining the average queue delay is the utilization, U, of any shared resource. The utilization is first determined, and the queue delay is then calculated from it by a well-known queuing theory formula, as below. The queuing equations used here have two inherent requirements. First, the input requests to the server should approximate a Poisson probability function, resulting in an exponential cumulative interarrival time distribution [3(c), 10]. This is approximately true for misses per instruction emanating from any given cache.[17] We use this approximation because no other simple analytical method is available, and it gives reasonable results. The second requirement is that all queues be of unbounded length (i.e., a maximum value cannot be specified), which is hardly the case in actual hardware. However, because the final *FCP* and thus *CPI* increase as queues become larger, which in turn makes the queues smaller, the inherent feedback will always prevent the queues from becoming infinite, or even very large. Nevertheless, we cannot specify any of the queue sizes, but rather only determine the average size after the analysis has converged (see the Introduction for comparison of open- and closed-input queue models).

Server utilization

A server is any shared resource which can have a queue. Utilization U, which is the percentage of time a server is busy, is given by

$$U = \text{input request rate/output service rate.} \qquad \text{(A1)}$$

[17] Separate studies of various SPEC95 and IBM internal traces show many examples which closely approximate a Poisson distribution and many which, while similar, deviate by a significant amount. For cases which deviate, there is no simple method available for estimating the error.

The input request rate, RR, to a server is represented in units of requests per cycle (i.e., customers per cycle) and can be calculated for any level, n, from the known miss rate, mr_n (misses/I), as

$RR = mr/CPI$ miss requests/cycle.

The output service rate is the inverse of the service time, S_t, which is the total time the server is busy with a request and thus cannot accept another request, in units of cycles/input request. For a memory array, this is typically the array cycle time. Thus, utilization is easily seen to be the product of the above two parameters, or

$$U = RR \times S_t . \tag{A2}$$

For a Poisson type of input, with *exponential service time distribution*, the waiting time in the queue line alone is

$$Q_w = S_t \frac{U}{1-U}. \tag{A3a}$$

But buses and arrays typically have constant service time, S_t, which reduces the above queue delay by a factor of 2 and gives the queue time for constant service time[18] as

$$Q_w = 0.5 \times S_t \frac{U}{1-U} = 0.5 \times \frac{RR\, S_t^2}{1 - RR\, S_t}. \tag{A3b}$$

After the Q wait, there remains the normal time, S_t, to get through the server. Thus, the total delay introduced by a constant-service-time server is

$$Q_w + S_t = S_t \left[1 + 0.5 \times \frac{U}{1-U} \right]. \tag{A4}$$

In most of our analyses, S_t is already included in the total reload path delay term, T_n, so that only the Q_w wait time of Equation (A3a) is needed for determining the extra delay.

Queue waiting line length

For exponential inputs (Poisson), and exponential [exp] and constant [const] service times, the mean lengths of the waiting lines alone are

$$L_w[\text{exp}] = \frac{U^2}{1-U} \quad \text{and} \quad L_w[\text{const}] = \frac{U^2}{2(1-U)}. \tag{A5a}$$

Note that L_w becomes 1 at approximately $U = 62\%$ and 73% for exponential and constant service times, respectively, and increases rapidly with U, becoming infinite at $U = 1$. The mean size of the queue (number of inputs waiting plus the number in service) for the same case of exponential inputs and service time is

$$Q_{size} = \frac{U}{1-U}. \tag{A5b}$$

For a constant service time, the mean size is slightly less than that given above, approximately

$$Q_{size} = \frac{U - U^2/2}{1-U}. \tag{A6}$$

This is obviously smaller than that of Equation (A5b) by the factor $U^2/2$ in the numerator. The queue sizes are approximately 1 at utilizations of about 0.5 (50%) and 0.58 (58%) for exponential and constant service times, respectively, but increase rapidly for increasing values of U.

Tandem queues

In order to initiate a reload to a cache which is several levels away, the request must propagate from the CPU/L1 control logic through the various intermediate levels of directories and request buses. Directories at each level are either fast, or their access delay for translation is included in the bus latency delays used as input parameters. Clearly, all tandem electrical wire and logic circuit delays are added for any path. Tandem queues can be added together only if the inputs to each queue have approximately a random interarrival time distribution. Since there are many processors making scattered requests over the hierarchy, this is approximately true for most queues, except for the data-out buffer queue discussed in the next section. All other tandem queues in the system are linearly added to the path delay time as appropriate.

Tandem DRAM—data bus queue

Typical arrays in a hierarchy access a full cache block on one array cycle and load this into an output buffer, which connects to the data bus. Most data buses have a much narrower data width than the output buffer and hence require multiple bus cycles to transfer the full block back to the requester. If no queue is provided on the output of the data-out buffer (DOB), i.e., in front of the data bus, a new array access cannot start until the DOB is fully empty; otherwise, the previous data can be overwritten. Since both the array access/cycle time and the DOB transfer time over the bus can be quite long, we assume that a queue is provided on the input to the data bus to reduce the total reload time. If there are multiple arrays at the same level, all using the same data bus, they also benefit from this queue. For U and Q calculations, the bus service time per block request is given by Equation (38), which can be significantly larger than the bus service time. For example, for a 128-byte cache block transfer over a bus of 16 bytes width and 3 processor cycles per bus transfer, the total service time per block request is $3 \times 128/16 = 24$ cycles.

[18] See Reference [10], Vol. II, p. 10.

An important issue relating to tandem queues arises in the actual queue delays to be used in the above path. The queue theory equations require Poisson inputs, meaning that the arrivals are scattered in time in a certain way. The input requests to the array are assumed to have this distribution, while if the array access queue always has an entry, the input requests to the data bus queue arrive at fixed intervals, namely the array cycle time. We model this situation by calculating both queues, assuming that the inputs have exponential interarrival times via Equation (A3), and then using only the maximum of these two queues in the *FCP* reload delay, namely

$$\textit{Total path queue delay} = RBQn + RBQ(n+1) + \max[AAQ(n+1) \text{ or } DBQ(n+1] + \cdots, \quad \text{(A7)}$$

where

n is the level of the hierarchy where the path delay starts, and progresses to subsequent levels, $n + 1$, $n + 2$, etc.;

$RBQn$ and $RBQ(n + 1)$ = request bus queues of any sort on levels n and $n + 1$, respectively;

$AAQ(n + 1)$ = DRAM array queue delay for level $n + 1$;

$DBQ(n + 1)$ = tandem queue delay of the data bus fed by the DRAM array.

The justification for the use of the maximum of the two queues [$AAQ(n + 1)$ or $DBQ(n + 1)$] above requires the consideration of two cases:

- Case 1: $S_{\text{array}} > S_{\text{bus}}$, so $AAQ(n + 1) > DBQ(n + 1)$. If the bus service time component S_{bus} is smaller than S_{array} of the array such that the array Q, $AAQ(n + 1)$, is larger than the bus Q, $DBQ(n + 1)$, and if the array has a significant queue (i.e., length > 1), the output rate of the array will be constant. Hence, the bus input request rate, RR_{bus}, is a constant, equal to the reciprocal of the service time of the array. Since the bus service time was assumed to be smaller than that of the array, and since the bus request rate $RR_{\text{bus}} = 1/S_{\text{array}}$, the bus will process the requests faster than the array, so there can be no queue on the bus. The queue equations are not valid for the bus, and the analysis is trivial. The only queue seen in the actual path is the larger array Q, $AAQ(n + 1)$, so discarding $DBQ(n + 1)$ is correct for this case.
- Case 2: $S_{\text{array}} < S_{\text{bus}}$, so $AAQ(n + 1) < DBQ(n + 1)$. Assume that the bus service time component is larger than that of the array, so that the array Q is smaller than the bus Q. If for some reason the $AAQ(n + 1)$ happens to be significant (i.e., length > 1), then, as above, the array output will provide a constant rate into the bus of value $RR_{\text{bus}} = 1/S_{\text{array}}$. Since the bus service time is larger than the array service time, the utilization term will be greater than 1, so the actual system will attempt to develop an infinite bus queue. However, the *CPI* will increase to the point where this does not occur. The actual array Q will decrease to a point such that it does not produce a constant output rate. Therefore, the array Q is negligible, and only the bus Q is seen in the actual system.

Appendix B: Miss rates vs. miss ratios

Miss rates and miss ratios are related by simple expressions at all cache levels except L1. Unfortunately, at the L1 cache level, conversion of miss rate to miss ratio requires a new and imprecise parameter, A_{I}, as shown below.[19] At all other levels, *FCP* terms can be expressed using either miss rates or miss ratios, since conversion requires no extra parameters. Miss rate, *mr*, is defined as the number of misses at any level of a cache, as a percentage of the *total number of instructions executed* by the processor, I_{E}:

$$\textit{L1 miss rate} = mr_1 = \textit{total \# L1 misses}/I_{\text{E}} = ML_1/I_{\text{E}}, \quad \text{(B1)}$$

with similar definitions for L2, L3, etc.

Miss ratios are defined as the number of misses per cache access, or

$$\textit{L1 miss ratio} = MR_1 = \textit{total \# L1 misses/total \# L1 accesses}, \quad \text{(B2)}$$

and

$$\textit{L2 miss ratio} = MR_2 = \textit{total \# L2 misses/total \# L2 accesses},$$

with similar definitions for L3, L4, etc.

Miss ratios, MR_n, are related to miss rates, mr_n, by the following conversion relations:

$$MR_1 = \frac{mr_1}{A_{\text{I}}}, \quad \text{(B3)}$$

$$MR_2 = \frac{mr_2}{mr_1}, \quad \text{(B4)}$$

$$MR_3 = \frac{mr_3}{mr_2}, \quad \text{(B5)}$$

where A_{I} is the average number of L1 (memory) accesses issued by the processor per instruction executed, expressed as

$$A_{\text{I}} = I_{\text{PA}}/I_{\text{E}} \quad \text{or} \quad I_{\text{E}} = I_{\text{PA}}/A_{\text{I}}. \quad \text{(B6)}$$

[19] See Reference [11], p. 272, for further discussion of A_{I} and analysis of first-level caches.

I_{PA} is the total number of processor instructions, or parts thereof, which make one access to the L1 cache, and thus equals the total number of L1 accesses. To use MR_1 for L1, we *must* know A_I, which requires a knowledge of the value of I_{PA}. The parameter I_{PA} causes the difficulty because some instructions will make multiple accesses to L1 (load or store multiple, etc.), the number depending on the program input variables, which cannot be known ahead of time and are not easily measured. Thus, we do not have a measure of I_{PA}, the actual number of L1 references, so A_I and thus the L1 miss ratio cannot be determined. However, it is relatively easy to measure the number of L1 misses, since this comes off the external bus, and we always know the total number of instructions executed. Hence, the L1 miss rate is easily obtained from Equation (B1).

Assuming that we know I_{PA} and A_I, the proof of the identity in Equation (B3) is as follows. Substituting Equation (B6) into Equation (B1) gives

$$mr_1 = ML_1 A_I / I_{PA} .$$

Dividing both sides by A_I and substituting Equation (B2) gives

$$\begin{aligned} mr_1/A_I &= ML_1/I_{PA} \\ &= \textit{total \# L1 misses/total \# L1 accesses} \\ &= MR_1 . \end{aligned}$$

The proof of the identities for MR_2 and MR_3 follows directly from their definitions.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Lotus Development Corporation, MathSoft Engineering & Education, Inc., or Transaction Processing Performance Council.

References

1. U. Bhat, "Sixty Years of Queueing Theory," *Mgt. Sci.* **15,** B280–B294 (1969).
2. S. S. Lavenberg, "A Perspective on Queueing Models of Computer Performance," *Perform. Evaluation* **10,** 53–76 (1989).
3. (a) E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984; (b) R. Wolff, *Stochastic Modeling and the Theory of Queues*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989; (c) S. Lavenberg, *Computer Performance Modeling Handbook*, Academic Press, Inc., New York, 1983.
4. M. Vernon, E. Lazowska, and J. Zahorjan, "An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols," *Proceedings of the 15th Annual International Symposium on Computer Architecture*, May 30–June 2, 1988, Honolulu, Hawaii, pp. 308–315.
5. *Electronics Engineers' Handbook*, 3rd Ed., D. Fink and D. Christiansen, Eds., McGraw-Hill Book Co., Inc., New York, 1989, pp. 5-39–5-60.
6. P. G. Emma, "Understanding Some Simple Processor-Performance Limits," *IBM J. Res. & Dev.* **41,** 215–232 (1997).
7. T. Lovett and R. Clapp, "STiNG: A CC-NUMA Computer System for the Commercial Marketplace," *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1996, p. 308.
8. K. Diefendorff, "Power4 Focus on Memory Bandwidth," *Microprocessor Report* **13,** No. 13, 1–8 (October 6, 1999).
9. R. Matick, *Computer Storage Systems and Technology*, John Wiley & Sons, Inc., New York, 1977. (*Author's note*: The footnote on p. 40 should read $I/B = 1$ to 4 instead of 1/16 to 1/4.)
10. L. Kleinrock, *Queueing Systems*, *Vol. I: Theory* (1975); *Vol. II: Computer Applications* (1976), John Wiley & Sons, Inc., New York.
11. R. Matick and D. T. Ling, "Architecture Implications in the Design of Microprocessors," *IBM Syst. J.* **23,** 264–280 (1984).

Received December 4, 2000; accepted for publication July 5, 2001

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 received B.S., M.S., and Ph.D. degrees in electrical engineering from Carnegie Mellon University, Pittsburgh, in 1955, 1956, and 1958, respectively. He then joined IBM to work in the areas of thin magnetic films, memories, and ferroelectrics. As the manager of the Magnetic Film Memory Group from 1962 to 1964, he received an IBM Outstanding Invention Award for his work on a thick-film read-only magnetic memory. Dr. Matick 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 Boulder. During the summer of 1973, he taught and carried out research at Stanford University. In 1986, he received an IBM Outstanding Innovation Award, and in 2000 an IBM Corporate Patent Portfolio Award, as co-inventor of the industry-standard "Video RAM" memory chip used to provide the bit buffer for high-speed, high-resolution displays for PCs and many workstations. 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 paperback in 2001) and *Computer Storage Systems and Technology*, John Wiley, 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), as well as in *Electronics Engineers' Handbook*, second and third editions, McGraw-Hill, 1982 and 1989, and *Cache Memory*, in van Nostrand Reinhold's *Encyclopedia of Computer Science* (third edition, 1993). His work in high-density CMOS cache memory design served as the foundation for the high-speed cache system used in the IBM RISC/6000 series processors; he received an IBM Outstanding Technical Achievement Award for that work in 1990. Dr. Matick is a member Eta Kappa Nu and a Fellow of the Institute of Electrical and Electronics Engineers.

Thomas J. Heller *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12601 (tjheller@us.ibm.com).* Mr. Heller received a B.S. degree in electrical engineering from Massachusetts Institute of Technology in 1985. He joined the IBM Data Systems Division that same year to work on large computing system designs. He is currently a Senior Engineer working on system design aspects and performance analysis of IBM's large servers. Mr. Heller holds several patents in the areas of multiprocessing and memory hierarchy design. His professional interests include the design and performance analysis of multiprocessor systems, memory caching algorithms, and distributed, shared-memory architectures.

Michael Ignatowski *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12601 (ignatow@us.ibm.com).* After receiving a B.S. degree in physics from Michigan State University in 1979 and an M.S. degree in computer engineering from the University of Michigan in 1982, Mr. Ignatowski joined IBM in Poughkeepsie, New York, to work on performance analysis of IBM S/390 servers. In 1991 he received an IBM Outstanding Innovation Award for contributions to the ES/9000 storage hierarchy and MP design. Mr. Ignatowski is a coauthor of five patents in the areas of storage hierarchy and MP design. He is currently a Senior Technical Staff Member, working on large-website performance and future server technology.