

# Estimation of Invalidation and Writeback Rates in Multiple Processor Systems

K. Kant

Intel Corporation

Beaverton, OR

kkant@co.intel.com

## Abstract

The paper presents a simple Markovian model for estimating coherency traffic for a multiprocessor system based on a few parameters including cache write/read fraction and read/write miss ratios. In particular, the model allows the estimation of cache invalidation traffic appearing on the bus, implicit writeback traffic and explicit writeback traffic. The basic model is developed for a symmetric multiprocessor (SMP) system with a single level of processor cache, and then generalized for multilevel caches and clustered systems (i.e., several SMP systems connected via an interconnect that optionally provides coherency traffic filtering). Filtering involves “back invalidations”, i.e., cache invalidations forced by the filter; a simple scheme is developed here for estimating the impact of back invalidations.

The proposed model is a part of the methodology for predicting the impact of various hardware platform features on the performance of commercial workloads. In this context, the proposed model along with a suitable mechanism for estimating cache miss ratios, has proven very useful for performance predictions.

## 1 Introduction

This paper is motivated by the need for accurately predicting the impact of various hardware platform features on the performance of commercial workloads. In particular, we are interested in quantifying

the impact of the following platform parameters on the performance web-server benchmarks such as SPECweb96 [3] and SPECweb99 [5]: (a) number of processors in the SMP (symmetric multiprocessor) system, (b) cache sizes and cache latencies at various caching levels, (c) processor, bus, and memory speeds, (d) various clustering arrangements of SMP nodes in a clustered server, etc. Reference [4] presents a detailed model of SPECweb96 benchmark to fulfill this need. Among other things, such a model needs to estimate the cache miss ratios and coherency traffic appearing on the processor bus. Both of these are governed by low level characteristics of not only the workload but also that of the O/S (e.g., resource locking/contention at O/S and application levels). Typically, these aspects can be accurately studied only via detailed address traces for a variety of configurations [1]. In view of difficulties associated with (and often the impracticability of) a trace-based analysis, our approach is to “transform” the values of certain overall parameters from a baseline system to those for the modeled system based on a rather limited set of measurements. In particular, reference [4] shows how this is done for the cache miss ratios and fetch/read/write rates at various caching levels. The present paper is concerned with using these miss ratios and reference rates in order to compute coherence traffic rates in both SMP and clustered systems. Here again, we look for simple models that allow us to transform baseline coherence traffic rates into those for the modeled system. Such an approach obviates the need to directly characterize such complexities as the low-level locking/contention behavior of the workload and the O/S. However, this also means that any significant change to the workload or the O/S would require recalibrating the model; fortunately, this is not an issue for the purposes for which the model is needed.

Here we consider a traditional SMP system where the processors, memory controller and perhaps the I/O controller all hang off a shared address/data bus. Each processor has a local cache but no local main memory or I/O devices. To keep the caches synchronized, a coherency protocol is needed, which involves snooping, invalidations, and writebacks. In particular, we assume the so-called MESI coherency protocol [2], which recognizes four states of a cacheline: exclusively held by a processor in modified state (M), exclusively held by a processor in clean state (E), shared, i.e., held by multiple processors in clean state (S), and invalidated or nonexistent (I). When a cacheline is initially brought into a processor cache, it is placed in the E state in anticipation of the normal situation where no other processor would request it. Every memory transaction issued by a processor involves a *snoop phase* where all processors on the bus check if the target cacheline resides in any of their caches and its state. If a cache holds the requested

data in modified state, that data must be delivered to the requesting cache. This happens in parallel with writing of the modified data to main memory, henceforth known as *implicit writeback*. If a processor wishes to gain exclusive access to a cacheline in S state, it must explicitly put its address on the bus so that other processors can snoop and invalidate their copy of the cacheline. Finally, a modified cacheline evicted from the highest level cache must be written back to the main memory, which we henceforth refer to as *explicit writeback*. Accordingly, in a SMP system, there are 3 types of coherency traffic that needs to be estimated: (a) *bus invalidation* or simply *invalidation* traffic, which affects only the address bus, (b) *implicit writeback* traffic, which affects address/data buses and memory channels, and (c) *explicit writeback* traffic, which also affects address/data buses and memory channels. This paper discusses a simple Markovian model to estimate these three types of traffic in a SMP.

Because of the limitations placed by electrical characteristics, control logic, and bus loading, SMP systems are typically limited to 2 or 4 processors. Larger systems are built by *clustering* multiple SMP nodes using some sort of interconnect. The purpose of the interconnect is to (a) route remote memory requests/responses, and (b) route and filter snoop requests and responses. In a uniform memory access (UMA) type of arrangement, all memory might be shared among all nodes, and is accessed via the interconnect. In the nonuniform memory access (NUMA) type of arrangement, each node may have its own local memory, which is remotely accessible via the interconnect. The memory access arrangement is not relevant for the purposes of this paper. The relevant aspect is the handling of coherency traffic. In the simplest case, the interconnect simply broadcasts all snoops and responses to all non-local nodes. In this case, the entire cluster effectively acts like a SMP and thus has same scalability problems as a SMP. In a more scalable setup, the glue logic provides a *snoop filter* for each node. The filter stores the tag for each cacheline stored in the processor caches of the node along with its status (exclusive/modified or shared).<sup>1</sup> Any snoop to a node results in a filter lookup and the snoop is placed on the bus only if necessary, i.e., if the requested cacheline is cached by the node and the access is either a write access or read access while the cacheline is in exclusive/modified mode. This considerably cuts down nonlocal snoops and thus significantly reduces address bus load.

Unfortunately, the filter introduces two undesirable side effects, which may or may not be significant depending on the implementation: (a) additional latency due to snoop filter lookups, and (b) *back inval-*

---

<sup>1</sup>The filter usually cannot distinguish between exclusive and modified states, and thus has a single indicator for both.

*invalidations* due to limited storage space for cacheline tags. The filter must maintain tags of all cachelines stored in all caches of all processors of the node. If the processor caches are large, it may not be cost effective to make the *tag-cache* large enough to contain tags of all stored cachelines. However, even when the tag-cache is properly sized, it cannot precisely mirror what is contained in the processor caches. The problem is that when a processor cache evicts a clean cacheline, nothing appears on the processor bus, and thus the filter knows nothing about the eviction. (Some processors may specifically put out “hints” on the bus to help tag-cache management, but we assume that this is not the case here.) Consequently, when a tag is selected for eviction from the filter, the corresponding cacheline could still be present in some cache. Thus to maintain consistency, it is important to treat the filter as yet another bus agent which attempts to grab the victim cacheline in the exclusive mode and thereby forces its invalidation in processor caches. This mechanism is known as “back invalidation”, and it obviously increases the address bus traffic. Furthermore, if the invalidated cacheline is modified it must be written back to memory and thus results in additional implicit writebacks. Finally, back invalidations may evict useful cachelines and thereby degrade cache hit ratios.

To summarize, the coherency traffic estimation for a cluster with snoop filter, involves estimating following additional parameters: (a) fraction of bus invalidations that must be reflected on a remote bus, (b) fraction of cache-to-cache transfers that require going across the interconnect, (c) probability of back invalidations and corresponding snoop and writeback traffic, and (d) degradation in cache hit ratio due to back invalidations. In this paper, we discuss techniques for estimating these as well in the context of the Markovian invalidation/writeback model.

## 2 Basic Model

In order to keep the model simple, we represent only the most dominant mode of operation and concentrate on the coherency protocol followed by a specific processor vendor (Intel). In particular, we assume that multilevel caches are strictly inclusive, all caching is “write-back” type (i.e., no “write-through”), and cache-to-cache transfer occurs only for the dirty data. To begin with, we consider invalidations/writebacks in a SMP system with only a single level of cache. Extension of the model to clusters and multilevel caches are considered in later sections.

The basic model represents various states that a given cacheline could be in and how it changes state as processors make read and write requests to it. Although the model considers only one cacheline, it is not necessary to assume that the behavior of all cachelines is captured by that of a single “average” cacheline. Instead, one could have some  $K$  cacheline types, and separately compute results for each type, followed by a weighted superposition of the results. Indeed, it is highly desirable to consider at least two types of cachelines:

1. Globally shared data: Cachelines containing semaphore data. These are characterized by high degree of sharing between processors and a high write/read ratio.
2. All other data, which is characterized by very small sharing and some application-specific write/read ratio.

The main problem with multiple cacheline types is obtaining adequate data for model calibration. For this reason, our application of the model so far has used only one cacheline type.

For a given cacheline (CL) type, we make the following modeling assumptions:

1. Each cacheline has a “home” processor, but others may occasionally access it. The “home” processor is the one that grabs the CL initially (i.e., when the cacheline does not reside in any of the caches). For successive cycles (where a cycle begins with an initial entry of CL into a cache and ends when the CL is out of all caches), the home processor may be the same or different — this aspect is not relevant for the model.
2. All processors uniformly process the entire workload, instead of each being assigned special set of functions. For example, the frequent practice of binding all interrupts to a given processor is not captured by the model.
3. Since the model is concerned with only one cacheline, a cacheline eviction is *not* initiated as a result of miss on another CL; instead, the CL stays in the cache for an appropriate amount of *residence time* and then leaves.
4. Residence time of a cacheline in a cache is exponentially distributed. This assumption is merely to enable a simple Markovian model. Using phase-type distributions, the model can be extended

to to more complex distributions; however, it appears adequate to account for non-exponential residence time by a slight adjustment of writeback rate, as explained later.

5. Only data read/write are considered in the model. Code fetches do not affect invalidations/writeback and are ignored.

With these assumptions, a N-processor system can be represented by a Markov chain with state given by the pair  $(n, m)$ ,  $n = 0 \dots N$ ,  $m = 0, 1$ , where

1.  $n = 0 \dots N - 1$ : Number of non-home processors holding the cacheline in clean state.
2.  $m = 0, 1$ : Indicates that home processor does not hold (holds) the CL.
3.  $n = N$ : Cacheline held in dirty state by home processor ( $m = 1$ ) or a non-home processor ( $m = 0$ ). These are modified (M) states of the MESI protocol.

State (0,0) represents the empty state, i.e., cacheline is not present in any processor cache. State (0,1) represents both S and E states, with S and E components denoted as (0,1s) (shared) and (0,1e) (exclusive). Now, to characterize the transitions, we consider the following processor initiated bus transactions in a SMP system:

1. BRL (bus read line): Read access to a line absent from the cache, which happens with rate  $\lambda_{ra}$ .
2. BRIL (bus read & invalidate line): Reading a line absent from the cache with intent to modify it (hence the invalidation of all other copies). This happens with rate  $\lambda_{wa}$ .
3. BIL (bus invalidate line): Gaining exclusive access to a line that is already available in the cache but in shared mode, with intent to modify it. This happens with rate  $\lambda_{wp}$ .

Note that the subscripts of the rates  $\lambda$ , the symbol  $r$  refers to read,  $w$  refers to write (or modify),  $a$  refers to absent from cache, and  $p$  refers to present in cache. We also introduce the following notation:

1.  $\alpha_w$ : Probability of write access to cache (unconditional value, doesn't matter if it is a hit or miss).
2.  $q_r$ : Read miss ratio (Probability of cache miss, given that it is a read).

3.  $q_w$ : Write miss ratio (Probability of cache miss, given that it is a write).
4.  $q_t$ : Overall miss ratio.
5.  $f_{sh}$ : Probability that a non-home processor accesses the CL.
6.  $\lambda_{ec}$ : Explicit eviction rate of a cacheline from home processor in shared and exclusive (i.e., clean) states. (Explicit eviction refers to eviction due to capacity misses, as opposed to removal of a cacheline from a cache because of exclusive access to it by another processor.)
7.  $\lambda_{ed}$ : Explicit eviction rate of a cacheline from home processor in modified (or dirty) state.
8.  $\lambda_a, \lambda_p$ : Aggregate access rates to a cacheline absent/present in the cache, i.e.,  $\lambda_a = \lambda_{ra} + \lambda_{wa}$ , and  $\lambda_p = \lambda_{rp} + \lambda_{wp}$ .
9.  $\lambda$ : Total cacheline reference rate by *home* processors, i.e.,  $\lambda = \lambda_a + \lambda_p$ .
10.  $\lambda_{tot}$ : Overall reference rate to the cache from all processors.
11.  $\lambda'_{ra}, \lambda'_{wa}, \lambda'_a, \lambda'_{wa}, \lambda'_{wp}, \lambda'_p, \lambda'_{ec}, \lambda'_{ed}, \lambda'$ : same as above, but for non-home processors.

Then the following relationships can be easily verified:

$$\lambda'_x = f_{sh} \lambda_x \quad \text{for all } x, \quad (1)$$

$$\lambda_{ra} = \lambda(1 - \alpha_w)q_r, \quad (2)$$

$$\lambda_{wa} = \lambda\alpha_w q_w, \quad (3)$$

$$\lambda_{wp} = \lambda\alpha_w(1 - q_w) \quad (4)$$

$$q_t = (\lambda_{ra} + \lambda_{wa})/\lambda = (1 - \alpha_w)q_r + \alpha_w q_w \quad (5)$$

At this point, we note one glaring deficiency of a Markovian model. Cacheline accesses are typically non-uniform and show periods of high and low activities. (This relates to traditional notions of different working sets in different phases of a computation.) The LRU caching exploits this non-uniform behavior by retaining frequently used cachelines longer. Even if we match the results for the baseline system, a model that does not account for this non-uniformity will not scale correctly with respect to the cache size.

To address this issue, we introduce an empirical parameter  $\beta$ , defined by the relationship  $\lambda_{ed} = \beta\lambda_{ec}$ . That is,  $\beta$  quantifies by how much the eviction rate in dirty state (typically part of active phase) changes because of the LRU action. The model is calibrated such that  $\beta = 1$  for the baseline system; therefore,  $\beta$  only quantifies the sensitivity with respect to cache size. We assume that  $\beta = [C_0/C]^\gamma$ , where  $0 < \gamma < 1$  is parameter,  $C_0$  is the baseline cache size, and  $C$  is the current cache size. Note that if  $C > C_0$ ,  $\beta < 1$ , and thus gives the change in LRU retention factor as a function of cache size.

The calibration of the model is necessarily based on measurements from a baseline system and appropriate scaling of input parameters. In particular, we obtain  $\alpha_w$ ,  $q_r$ ,  $q_w$ , and cache reference rate  $\lambda_{tot}$  for the baseline system and scale them appropriately for the modeled system [4]. The home-processor cache reference rate  $\lambda$  can be related to  $\lambda_{tot}$  as shown later. The eviction rate  $\lambda_{ec}$  is not known a-priori, and is determined by iteration as also shown in section 3. The only other input parameter is the sharing fraction  $f_{sh}$ , which can be regarded primarily as the characteristic of the application/operating system combination. In particular, sharing is primarily a result of access to global data or contention for access to such data. To estimate  $f_{sh}$ , we match the model results against measured values for the baseline system. Let  $N^{(0)}$  and  $f_{sh}^{(0)}$  denote, respectively, the number of processors and the sharing fraction for the baseline system. Then for the modeled system with  $N$  processors, we let

$$f_{sh} = \left[ \frac{(N^{(0)} - 1)f_{sh}^{(0)}}{N - 1} \right]^\beta \quad (6)$$

This equation preserves the sharing over all processors, since that is considered to be the property of application/OS, rather than the hardware platform. The factor  $\beta$  accounts for the fact that a greater cacheline retention results in greater potential sharing of cacheline between home and non-home processors. Note that for  $C \rightarrow 0$ ,  $\beta \rightarrow \infty$ , and hence  $f_{sh} \rightarrow 0$ . On the other hand, if  $C \rightarrow \infty$ ,  $\beta \rightarrow 0$ , which means that  $f_{sh} \rightarrow 1$ . That is, the equation correctly models the fact that the sharing increases monotonically from 0 to 1 as the cache size increase from 0 to  $\infty$ .

Figure 1 shows the Markov chain and the transitions. The horizontal forward transitions among the shared states  $(n, m) \rightarrow (n + 1, m)$  represent situations where a new non-home processor experiences a read-miss on the cacheline and brings a copy in its cache as well. The backward transitions in these cases correspond to the eviction of clean lines from non-home processors. The vertical transitions among



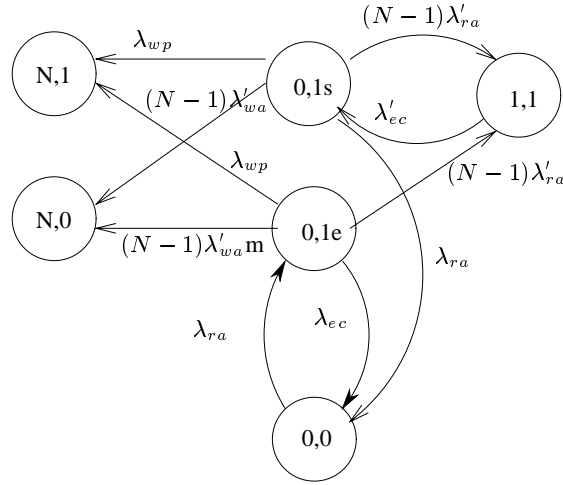


Figure 2: Lumpability of states (0,1s) and (0,1e)

(0,1s), but the designation remains “shared”. It can be seen from figure 2 that states (0,1s) and (0,1e) can be lumped into a single state (1,0) as far as the overall Markov chain is concerned, and thus need not be represented explicitly. Also, by writing the balance equation for state (0,1s), we get

$$P(0, 1s) = \frac{\lambda'_{ec}P(1, 1)}{\lambda_{wp} + \lambda_{ec} + (n-1)\lambda'_a} \quad (7)$$

which means that state probabilities  $P(0,1s)$  and  $P(0,1e)$  can be computed easily, once the probability  $P(1,1)$  is known.

### 3 Solution of Basic Model

Let  $Q$  denote the transition rate matrix of the Markov chain. Since  $Q$  is irreducible, it has a unique steady state probability vector  $\Pi = [P(0, 0), P(1, 0), P(1, 1) \dots P(N, 0), P(N, 1)]$  given by:

$$\Pi.Q = 0 \quad \text{and} \quad \Pi.E = 1 \quad (8)$$

where  $E$  is the unit vector. Due to the lack of significant structure in  $Q$  and a fairly small  $N$  for SMPs, the linear system above is best solved directly.<sup>2</sup> Recall that the home reference rate  $\lambda$  and eviction rate  $\lambda_{ec}$  are not known a-priori in the model. To determine  $\lambda_{ec}$ , we need an iterative procedure, described below.

1. Set  $\lambda = 1$ , and  $\lambda_{ec} = q_t$  (the target cache miss ratio). Note that  $\lambda_{ec} < q_t \lambda$  since a cacheline is also removed from the cache implicitly if another processor wants to grab it in exclusive mode.
2. Solve for  $\Pi$ , and determine (a)  $\lambda_{\text{ref}}$ , total rate at which cachelines are referenced, and (b)  $\lambda_{\text{inp}}$ , total rate at which cachelines are brought into the cache.

$$\lambda_{\text{ref}} = \sum_{i=0}^{N-1} \sum_{m=0}^1 [i\lambda'_p + (N-1-i)\lambda'_a + m\lambda_p + (1-m)\lambda_a]P(i, m) - (N-1)\lambda'_a P(0, 0) \\ + [\lambda_a + \lambda'_p + (N-2)\lambda'_a]P(N, 0) + [\lambda_p + (N-1)\lambda'_a]P(N, 1) \quad (9)$$

$$\lambda_{\text{inp}} = \sum_{i=0}^{N-1} \sum_{m=0}^1 [(1-m)\lambda_a + (N-1-i)\lambda'_a]P(i, m) - (N-1)\lambda'_a P(0, 0) \\ + [\lambda_a + (N-2)\lambda'_a]P(N, 0) + (N-1)\lambda'_a P(N, 1) \quad (10)$$

3. Compute the new cache miss ratio  $q'_t = \lambda_{\text{inp}}/\lambda_{\text{ref}}$ . If  $q'_t = q_t$  within an error bound, stop.
4. Suitably adjust  $\lambda_{ec}$  for the next iteration. The technique that results in fast convergence keeps track of the error  $(q'_t - q_t)/q_t$  for the current and previous steps, henceforth denoted as  $E_c$  and  $E_p$  respectively. If  $E_p$  and  $E_c$  are of *opposite sign*, we choose  $\lambda_{ec}$  using linear interpolation:

$$\lambda_{ec} = \lambda_{ec}^{(p)}\eta + \lambda_{ec}(1 - \eta) \quad \text{where} \quad \eta = \left| \frac{E_c}{E_c - E_p} \right| \quad (11)$$

where  $\lambda_{ec}^{(p)}$  refers to  $\lambda_{ec}$  value from the previous iteration. If  $E_c$  and  $E_p$  have the same sign, we increase/decrease  $\lambda_{ec}$  further by 25% (or by a smaller amount if the error is very small).

$$\lambda_{ec} = \lambda_{ec} \left[ 1 - \text{sgn}(E_c) \min(0.25, 100|E_c|) \right] \quad (12)$$

where the function  $\text{sgn}(x)$  refers to the sign of the argument  $x$ .

---

<sup>2</sup>For large clusters,  $N$  might become large, and an iterative technique might be useful.

We note here that one can establish monotonic convergence of the algorithm if we modify  $\lambda_{ec}$  using the schema:  $\lambda_{ec} = \lambda_{ec}(q_t - q'_t)/q_t$ ; however, the convergence is quite slow in this case. The procedure above changes  $\lambda_{ec}$  aggressively and thus achieves a faster convergence, although it clearly does not result in monotonic convergence.

It is important to note that the probability of the empty state (0,0) is not the same as the miss ratio. In fact,  $P(0,0) < q_t$ , since state (0,0) represents the situation that the cacheline is not in any of the caches, whereas  $q_t$  represents the probability that on an arbitrary access by some processor, the cacheline is not found to be in the cache. It is clear from figure 1 that the read (write) miss rates in every state have the common term  $\lambda_{ra}$  ( $\lambda_{wa}$ ). It thus follows from equation (5) that so long as  $q'_t = q_t$ , we are assured that  $q'_r = q_r$  and  $q'_w = q_w$ , i.e., the computed read and write miss ratios will also match the assumed ones.

Let  $\lambda_{tot}$  denote the total number of cache references per transaction (determined from measurements on a baseline system and appropriate scaling for the modeled system). Since  $\lambda_{ref}$  above was computed assuming  $\lambda = 1$ , the desired value of  $\lambda$  is given by:

$$\lambda = \lambda_{tot}/\lambda_{ref} \quad (13)$$

All the rates computed above could now be scaled for this actual  $\lambda$  value. Finally, we determine rates for invalidations and writebacks, as follows:

**Bus Invalidation Rate:** Bus invalidation rate is the overall BIL rate, i.e.,  $\lambda_{wp}$  and  $\lambda'_{wp}$  terms summed over all *shared* states. That is,

$$\lambda_{b\_inval} = \sum_{i=1}^{N-1} \sum_{m=0}^1 [m\lambda_{wp} + i\lambda'_{wp}]P(i, m) + \lambda_{wp}P(0, 1s) \quad (14)$$

**Implicit Writeback Rate:** This is the BRL/BRIL rate in modified states ( $\lambda_{xa}, \lambda'_{xa}$ ,  $x = r, w$  terms in states (N,0) and (N,1).)

$$\lambda_{i\_wback} = (N-1)\lambda'_a P(N, 1) + [\lambda_a + (N-2)\lambda'_a]P(N, 0) \quad (15)$$

**Explicit Writeback Rate:** This is eviction rate ( $\lambda_{ed}$  and  $\lambda'_{ed}$  terms) in states (N,0) and (N,1).

$$\lambda_{e\_wback} = \lambda_{ed}P(N, 1) + \lambda'_{ed}P(N, 0) \quad (16)$$

## 4 Extensions to Multi-level Caches

Most current processors have at least 2 levels of caches: Level 1 (L1) cache, which is typically quite small in size (16 KB–64 KB) and may have logically separate portions for code and data, and Level 2 (L2) cache, which could span a large range (256 KB–4 MB), and is typically unified. The L1 cache is invariably a part of the processing core, and thus has very low latency (1–2 clock cycles per access), whereas L2 cache is on the periphery and may even be off-die and could span a large range in latency (few cycles to few tens of cycles). Some systems even have a level 3 (L3) cache, which is still slower but larger.

Multilevel caches usually (but not always) operate with the *inclusion property*, i.e., a cacheline can be present in level  $i$  cache only if it is present in level  $i + 1$  cache (with memory at the highest level). Given that all execution requires data access from L1, any miss on a cacheline will propagate down the cache hierarchy until the required data is either available in a cache or needs to be fetched from memory. This data will be placed in all the caches (by successively moving up the hierarchy). Subsequently, the CL may be evicted from some cache  $L_i$  but still reside in caches  $L_j, j > i$ . To maintain inclusion property, this line must also be evicted immediately from caches  $L_j, j < i$ . If there is a write access to a cacheline in L1 (BRIL or a BIL), the modified data is not propagated down the hierarchy (assuming write-back, as opposed to write-through); however, the protocol must have some means of returning the up to date copy of the data when another processor accesses this cacheline.

A  $K$ -level cache model requires basic parameters like  $\alpha_w, q_r, q_w, \beta$ , and  $\lambda_{tot}$  for each level. Again, these would be obtained by an appropriate scaling of parameters from a baseline system. For consistency across levels, it is important that the read and write miss traffic from level  $i$  cache be equal to the level  $i + 1$  cache access plus any speculative read traffic. A  $K$ -level inclusive cache hierarchy can be represented as a Markov chain with state space  $(n_1, n_2, \dots, n_K, m)$  with the following feasible states:

1.  $n_i = 0 \dots N - 1$ : Number of non-home processors holding clean CL in caches  $L_i, L_{i+1}, \dots, L_K$ .
2.  $m = 0, \dots, K$ :  $m = 0$  means that the CL is not in any home processor cache, and  $m > 0$  means that the CL exists in caches  $L_m, L_{m+1}, \dots, L_K$  of the home processor.
3.  $n_i = N, n_{j \neq i} = 0, m = 0$ : Cacheline held in dirty state by a non-home processor in cache  $L_i$ . As far as the model is concerned, it can be assumed that caches  $L_{i+1}, \dots, L_K$  also contain dirty information.
4.  $n_i = N, n_{j \neq i} = 0, m = 1$ : Cacheline held in dirty state by the home processor in cache  $L_i$ .

Appropriate transition probabilities could now be defined, as illustrated in Figure 3 for a 2-level cache case. Here the subscripts 1 and 2 on various rates indicate whether the rate relates to L1 or L2 cache. To avoid clutter, not all states, transitions, or transition probabilities are shown.

For K-level cache system, number of feasible states is  $2^{\binom{N+K-1}{K}} + 2K$ . Since the Markov chain does not possess any structure that could be exploited for an efficient exact solution, the state space size could become an issue for multilevel caches with a large number of processors.

Note that every write access to clean data in L1 cache must be reflected (eventually) in a similar write to L2 cache. That is, in terms of our model,  $\lambda_{wp1} = \lambda_{wp2}$ . With this, the states in Figure 3 can be lumped into sets that only retain membership in L2 cache. That is, with  $k$  defined as  $i + j$ , the lumped states are  $(k, 2) \cup (k, 1)$  when the home-processor contains the cacheline, and  $(k, 0)$  otherwise. This lumping defines an aggregated Markov chain similar to the one for a single level cache. However, the lumped model only provides invalidations and implicit writebacks when state transitions occur with respect to L2.

This still leaves the problem of determining the invalidations and implicit writebacks due to data modification in L1. (Explicit writebacks from L1 are not relevant). This is equivalent to aggregating states with the same value of  $i$ . It is clear from Figure 3 that such an aggregation is not exact, but instead results from assuming certain “average” value of  $j$ , i.e., some constant number of processors hold the cacheline only in L2.

In spite of the errors introduced by the approximate aggregation, we have thus far used this model in our estimations for the sake of simplicity. Note that once both L1 and L2 parameters are obtained, the

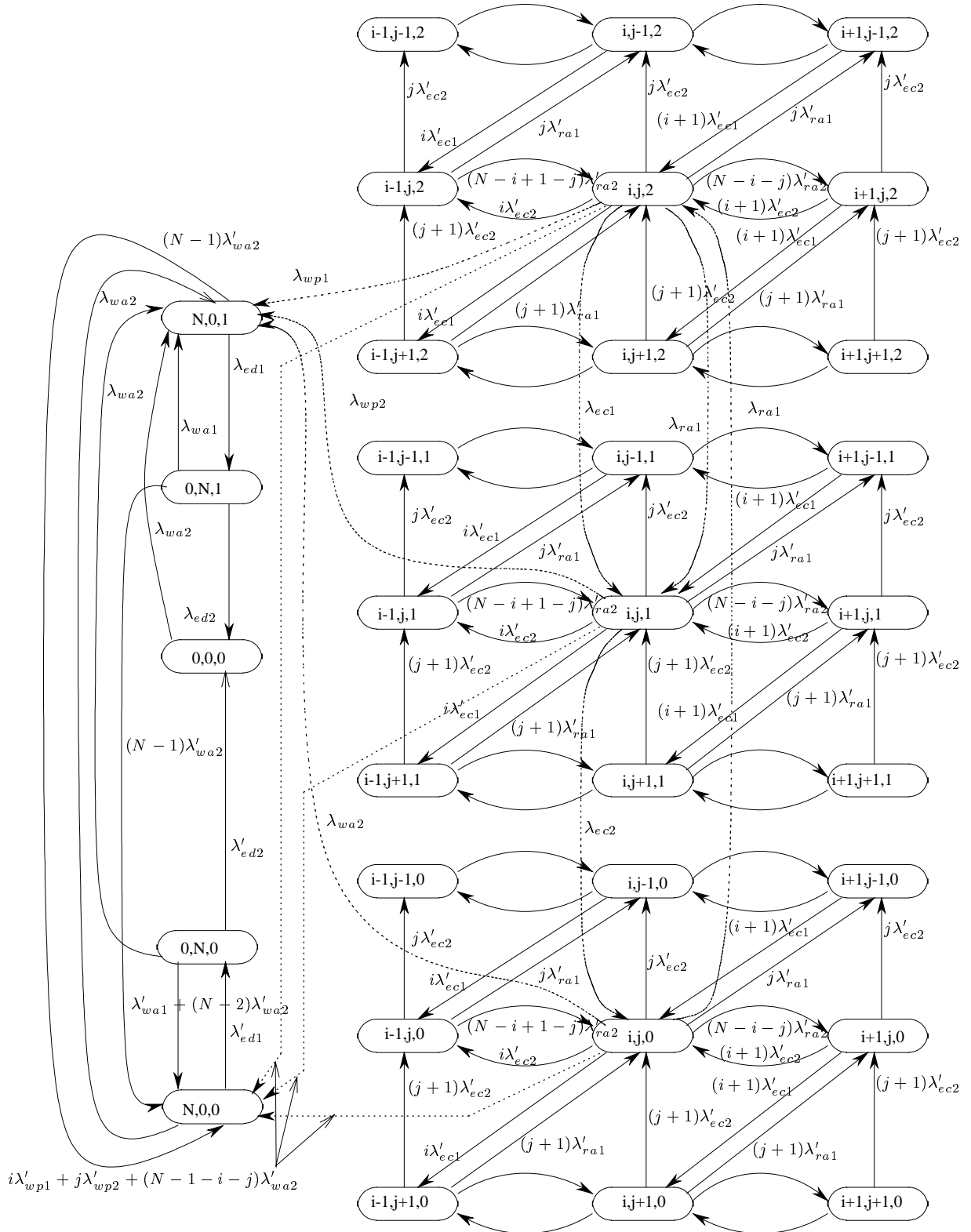


Figure 3: Markov Chain Model for 2-level caches

total invalidations and implicit writebacks are obtained by summing the L1 and L2 values. The relevant explicit writebacks are, of course, those out of L2 only.

## 5 Model for Clusters

We consider a cluster with  $K$  *identical* SMP nodes, each with  $n$  processors. We assume that the cluster interconnect provides a direct route between any two nodes (e.g., via a crossbar), and also provides a snoop filter for each node to block unnecessary remote snoops. The memory architecture (e.g., centralized memory for all nodes or distributed per-node memory) is not relevant for the model here.

Let  $N = n \times K$  denote the total number of processors in the system where  $n$  is the number of processors per node. Other than some coherency traffic reduction by the snoop filter and interconnect delays, the entire system basically acts like a  $N$ -processor SMP, hence, we need a model like the one in figure 1. Although multilevel caches can be handled exactly as before, a much larger value of  $N$  in the cluster context makes the approximate single level cache model even more attractive, and will be considered here. As discussed in section 1, the additional parameters to compute for a cluster are (a) inter-node snoop, invalidation and writeback traffic, and (b) impact of back invalidations on miss ratios and hence on invalidations/writebacks.

### 5.1 Inter-node Traffic

In order to compute inter-node traffic, we designate one node as *local* node and lump all others into a *remote* node. If the application is not optimized for a cluster environment, any node is equally likely to request access to a given cacheline. In this *completely symmetric* case, one local-remote pair suffices for computations. In a more general case, multiple local-remote pairs may be needed. Suppose that the cacheline currently resides in  $i$  processors ( $i \in 1..N$ ). Let  $\alpha_{ij}$  denote the probability that  $j$  processors out of  $i$  belong to the *local* node. The range for  $j$  is  $\max(0, i - N + n).. \min(i, n)$ , where the lower bound results from the fact that if  $i > N - n \triangleq (K - 1)n$ , we have the situation where the cacheline exists in every processor of all remote nodes, and thus at least  $i - (N - n)$  local processors must hold it

too. In the completely symmetric case,

$$\alpha_{ij} = \binom{n}{j} \binom{N-n}{i-j} / \binom{N}{i} \quad (17)$$

In the nonsymmetric case, the application will be divided up into “modules”, such that each module is assigned to a particular node and inter-module communication is kept to minimum. Here each module defines a cacheline type (see discussion at the beginning of section 2), and we would need a separate local-remote pair for each cacheline type. The local node then corresponds to the node at which the selected module is allocated, and the traffic of interest is the communication between this and other modules. Given this communication pattern, an appropriate expression for  $\alpha_{ij}$  could be obtained. Let the triple  $(i, j, k)$  identify the *cluster state* where the parameter  $k = 0, 1$  denotes whether home-processor does not hold or holds the cacheline. Note that if  $k = 1$ , the cacheline is held by only  $i - 1$  non-home processors. We now write down expressions for the rate at which BIL, BRL, and BRIL transactions are generated in the cluster state  $(i, j, k)$ .

$$\begin{aligned} \text{BIL}(i, j, 1) &= \alpha_{ij} (j/i) [\lambda_{wp} + (j-1)\lambda'_{wp}] \\ \text{BIL}(i, j, 0) &= \alpha_{ij} j\lambda'_{wp} \end{aligned} \quad (18)$$

where  $j/i$  is the probability that the home processor lies on the local side. If  $j = n$ , no BRLs/BRILs can be generated locally, and for  $j < n$ , we can write the following equations for BRL and BRIL rates.

$$\begin{aligned} \text{BRL}(i, j, 1) &= \alpha_{ij} (n-j)\lambda'_{ra} \\ \text{BRL}(i, j, 0) &= \alpha_{ij} \left( \frac{n-j}{N-i} \right) [\lambda_{ra} + (N-1-i)\lambda'_{ra}] \end{aligned} \quad (19)$$

$$\begin{aligned} \text{BRIL}(i, j, 1) &= \alpha_{ij} (n-j)\lambda'_{wa} \\ \text{BRIL}(i, j, 0) &= \alpha_{ij} \left( \frac{n-j}{N-i} \right) [\lambda_{wa} + (N-1-i)\lambda'_{wa}] \end{aligned} \quad (20)$$

where  $\frac{n-j}{N-i}$  gives the probability that that the home processor falls on the local side, given that it does not hold the cacheline. In this case, the home-processor and the  $(n-j-1)$  non-home processors on the local side can generate a BRL with rates  $\lambda_{ra}$  and  $\lambda'_{ra}$  respectively. If the non-holder home processor lies on the remote side, all  $(n-j)$  local processors general BRL with rate  $\lambda'_{ra}$ . Thus, the average number of

non-home processors generating BRL is  $(n - j)(1 - 1/(N - i))$ . The same argument applies to BRILs. Note that  $i = N$  implies  $j = n$ , and so the equations above do not have a singularity.

Now, to compute overall rates, we multiply these rates with appropriate state probabilities for the Markov chain in figure 1 and sum over all relevant  $i$ ,  $j$ , and  $k$  values. For  $i = 1$ , only one processor has the CL, and we need to correctly address two cases: (1) Distinction between shared and exclusive-clean states (a BIL needs to be generated only in the shared mode), and (2) Modified (exclusive-dirty) state (no BIL is generated but BRLs/BRILs are generated normally).

$$\text{BIL} = \sum_{i=1}^N \sum_{j=\max(0, i-N+n)}^{\min(i, n)} \left[ \text{BIL}(i, j, 0)P(i, 0) + \text{BIL}(i, j, 1)P(i - 1, 1) \right] - \text{BIL}(1, 1, 1)P(0, 1e) \quad (21)$$

Notice that the empty state  $(0,0)$  (which corresponds to  $i = j = k = 0$ ) is specifically excluded in the above equation. For BRL/BRILs, this state must be considered; however, only a home-processor can make requests. Thus, for BRL, we only consider the  $\lambda_{ra}$  term in  $\text{BRL}(0, 0, 0)$ , which is  $\alpha_{00}(n/N)\lambda_{ra} = \lambda_{ra}/K$ . Hence, we get:

$$\begin{aligned} \text{BRL} = & \sum_{i=1}^N \sum_{j=\max(0, i-N+n)}^{\min(i, n)} \left[ \text{BRL}(i, j, 0)P(i, 0) + \text{BRL}(i, j, 1)P(i - 1, 1) \right] + \frac{\lambda_{ra}}{K}P(0, 0) \\ & + \sum_{j=0}^1 \left[ \text{BRL}(1, j, 0)P(N, 0) + \text{BRL}(1, j, 1)P(N, 1) \right] \end{aligned} \quad (22)$$

$$\begin{aligned} \text{BRIL} = & \sum_{i=1}^N \sum_{j=\max(0, i-N+n)}^{\min(i, n)} \text{BRIL}(i, j, 0)P(i, 0) + \text{BRIL}(i, j, 1)P(i - 1, 1) + \frac{\lambda_{wa}}{K}P(0, 0) \\ & + \sum_{j=0}^1 \left[ \text{BRIL}(1, j, 0)P(N, 0) + \text{BRIL}(1, j, 1)P(N, 1) \right] \end{aligned} \quad (23)$$

Now, the local bus invalidation rate is same as BIL rate since every generated BIL will be placed on the local bus for snooping. A BIL needs to be passed to a remote node only if the cacheline is stored remotely. That is, we should exclude the BIL rate when  $j = i$  (i.e., when all cachelines are on the local

side).

$$\lambda_{l\_inval} = \text{BIL} \quad (24)$$

$$\lambda_{r\_inval} = \text{BIL} - \sum_{i=1}^n \left[ \text{BIL}(i, i, 0)P(i, 0) + \text{BIL}(i, i, 1)P(i-1, 1) \right] + \text{BIL}(1, 1, 1)P(0, 1e) \quad (25)$$

The local snoop rate is the sum of BRL and BRIL rates since all such transactions would involve the snoop phase.<sup>3</sup> A local snoop needs to be passed to a remote node only if (a) the snoop agent issues a BRL and the line is stored in exclusive or modified state in the remote node (i.e.,  $j = 0, i = 1$ ), or (b) the snoop agent issues a BRIL and the line is cached by the remote node (i.e.,  $j < i$ ). Then

$$\lambda_{l\_snoop} = \text{BRL} + \text{BRIL} \quad (26)$$

$$\lambda_{r\_snoop} = \lambda_{r\_snoop,a} + \lambda_{r\_snoop,b} \quad (27)$$

where the RHS quantities correspond to the two cases identified above and are given by:

$$\lambda_{r\_snoop,a} = \text{BRL}(1, 0, 1)P(0, 1e) + \text{BRL}(1, 0, 0)P(N, 0) + \text{BRL}(1, 0, 1)P(N, 1) \quad (28)$$

$$\begin{aligned} \lambda_{r\_snoop,b} = & \text{BRIL} - \sum_{i=1}^n \left[ \text{BRIL}(i, i, 0)P(i, 0) + \text{BRIL}(i, i, 1)P(i-1, 1) \right] \\ & - \lambda_{wa}P(0, 0)/K - \text{BRIL}(1, 1, 0)P(N, 0) - \text{BRIL}(1, 1, 1)P(N, 1) \quad (29) \end{aligned}$$

Finally, a locally generated BRL/BRIL would result in a hit-modified in the remote node (and hence cause data transfer from remote to local node), if and only if the cacheline is held in modified state by the remote node. That is, the local and remote implicit writeback rates (i.e., portions of locally generated BRL/BRILs that cause HITM locally and remotely) are given by:

$$\lambda_{l\_iwback} = [\text{BRL}(1, 1, 0) + \text{BRIL}(1, 1, 0)]P(N, 0) + [\text{BRL}(1, 1, 1) + \text{BRIL}(1, 1, 1)]P(N, 1) \quad (30)$$

$$\lambda_{r\_iwback} = [\text{BRL}(1, 0, 0) + \text{BRIL}(1, 0, 0)]P(N, 0) + [\text{BRL}(1, 0, 1) + \text{BRIL}(1, 0, 1)]P(N, 1) \quad (31)$$

In a completely symmetric case, the substitution of equations (19) and (20) in the above equations, and the simplification using equation (17) yields the following simple relationship between local and remote

---

<sup>3</sup>Since bus invalidations are being considered separately, they are not included in the snoop rate calculations.

implicit writebacks:

$$\lambda_{r\_iwback} = \left( \frac{N - n}{n - 1} \right) \lambda_{l\_iwback} \quad (32)$$

In the symmetric case, the outgoing traffic from a node equals the total incoming traffic. Thus one could obtain the above relationship directly by observing that if a processor is holding a cache-line in modified state, access requests to it are equally likely from any of the other  $n - 1$  local processors, or  $N - n$  remote processors. Unfortunately, such simplification does not seem possible for invalidations and snoops.

Notice that the calculations above are for the traffic generated by a single node of the cluster. In the symmetric case, the total traffic (i.e., invalidations, snoops, and implicit writebacks) will be simply  $K$  times that of the calculated values. In the nonsymmetric case, we must repeat the calculation  $K$  times and then use the sum.

## 5.2 Estimating impact of back invalidations

First suppose that the tag cache size equals the sum of all processor caches. Whenever a processor experiences a cacheline miss and reads in new cacheline, the corresponding entry must be recorded in the tag cache, if not there already. Since the tag cache keeps tags for all processor caches, a miss by a processor for a cacheline that is already in some other cache does not result in a new entry in the tag cache (the modified/shared status of the entry may, however, need to be changed at this time). If the cacheline is not in any processor cache, a new location must be allocated in the tag cache to hold the new tag. If the selected location has a valid entry, a back invalidation must be initiated since the tag cache has no knowledge whether that entry is stale. Note that the tag cache can explicitly invalidate one of its entries only when the corresponding cacheline appears on the bus because of an explicit writeback initiated upon eviction from the processor cache. Thus we can write the following equation for the back invalidation rate:

$$\lambda_{b\_inval} = \lambda_a P(0, 0) (1 - \lambda_{e\_wback}/\lambda) \quad (33)$$

where  $\lambda_a P(0, 0)$  is the rate at which a cacheline is brought into processor cache in the empty state, and the last term is the probability that the selected location contains a valid entry. This simple equation does not consider the associativity of the tag or processor caches, and is reasonable for fully associative caches. If the tag cache has smaller associativity than the processor caches, the back invalidation rate is expected to be higher.

Now suppose that the tag cache size is  $k$  times the sum of processor cache sizes. In this case, when a new tag cache location needs to be allocated, there are  $k$  potential locations to use. Under steady state, each one of these will contain a previously cached entry, although some entries may have been invalidated as a result of explicit writebacks. If at least one location is invalidated, the request can be satisfied without a back invalidation. It follows that:

$$\lambda_{b\_inval} = \lambda_a P(0, 0) (1 - \lambda_{e\_wback}/\lambda)^{\alpha_k} \quad (34)$$

where  $\alpha_k$  is the average number of entries (out of  $k$ ) that are in invalid state. Note that for  $k > 1$ , at the time of an allocation request, the other cachelines mapping to these  $k$  locations may have been brought in again, and thus the corresponding tag cache location will no longer be in invalid state. It is difficult to estimate  $\alpha_k$ . Based on some simulation data [6],  $\alpha_k$  can be approximated as  $\log_2(k) + 1$ .

The next issue is computing the impact of back invalidations on the cache miss ratio. For this, we use the following approach:

1. From the original model (without back invalidations) compute  $\lambda_{b\_inval}$  and the eviction rate  $\lambda_{ed}$ .
2. Modify the Markov chain by adding transitions with rate  $\lambda_{b\_inval}/(1 - P(0, 0))$  from all non-zero states to the empty state (0,0). These transitions represent back invalidations, with an effective rate of  $\lambda_{b\_inval}$ .
3. Solve the modified model with computed  $\lambda_{ed}$ , and determine the modified miss ratio  $q'_t$ .
4. Modify  $q_r$  and  $q_w$  by the factor  $q'_t/q_t$ .
5. Solve the modified model with modified  $q_r$  and  $q_w$ , and iterate to compute the appropriate value of  $\lambda_{ed}$  that ensures that estimated miss ratio is same as  $q'_t$ .

The rate of implicit writebacks due to back invalidations is given by:  $\lambda_{b\_inval}[P(N, 0) + P(N, 1)]$ . Note that unlike the calculations in the previous section, the excess invalidation and implicit writeback rates computed here are already over all nodes of the cluster, so no multiplication by  $K$  is needed.

## 6 Model Validation

The model has been validated in two ways: direct and indirect. The direct validation consists of sampling low-level hardware counters for a set of SMP configurations and then compare the measured and calculated invalidation, implicit writeback and explicit writeback rates. To date, this form of validation has been somewhat weak because of a number of limitations: (a) the hardware counters can count only one event at a time; therefore, the measured invalidation, implicit writeback and explicit writeback rates necessarily contain some noise, (b) having a consistent set of measurements with all software frozen is always difficult for commercial workloads because of constant improvement at various levels of the hierarchy, and (c) detailed measurements on clustered SMPs are currently unavailable. Consequently, we present direct validation for only SMP configurations.

The indirect validation essentially consists of comparing gross parameters from the model and the measurements. The major parameters are transaction throughput, address bus utilization, data bus utilization, bus transaction latency, etc. The model has consistently provided excellent match with measured values, although here again most results are for SMP systems. Reference [4] presents some of these comparisons.

For a direct validation shown here, we chose a consistent set of recent SPECweb99 runs (using a 500 MHz Intel Pentium III based server running Windows 2000 RTM O/S). Table 1 shows the results for 1 and 2 processor systems with 3 values of level 2 cache (512KB, 1MB and 2MB). The values shown are on a per transaction basis. Note that the 1P system also has a significant number of invalidations and implicit writebacks. These are a result of conflicts between I/O-side and processor-side access to the cached data. For the analytic model, the estimated I/O related invalidations and implicit writeback rates are simply the average per transaction values determined from measurements.

The processor used in these measurements has 2 levels of cache; therefore, the the calculations used the multilevel cache extension discussed in section 4. In particular, we used the approximate method

No of procs	cache size	Invalidations		Expl. wbacks		Impl. wbacks	
		analytic	actual	analytic	actual	analytic	actual
1	512	58	49	464	474	101	99
1	1024	58	53	375	371	101	98
1	2048	58	65	291	262	101	99
2	512	190	207	436	469	370	347
2	1024	227	234	375	366	421	403
2	2048	273	259	313	229	480	460

Table 1: Comparision between Analytic and Actual (measured) results

wherein L1 and L2 parameters are computed separately. Unfortunately, the hardware counters do not allow separate measurements of L1 and L2 contributions; therefore, the reported comparision is only for the overall invalidation and implicit writeback rates.

Given the noise in measured values, it can be seen from Table 1 that the calculated values match measured values well enough at least from an overall performance model perspective.

## 7 Conclusions and Future Work

The paper presented some simple techniques for estimating coherency traffic in a symmetric multiprocessor and clustered servers. Such an estimation is necessary in the course of modeling the impact of hardware platform parameters on workload performance. The techniques proposed conciously avoid the need for address traces or other detailed hardware data since such data is often unavailable, especially for products that are not yet commercially available. The techniques are also geared towards matching results for a baseline system and then extending the results for other systems. Such a method avoids detailed modeling of complex behavior (e.g., precise temporal and spatial characteristics of the traffic seen by the caches). On the negative side, such a model is expected to work well only for systems whose underlying architecture does not differ considerably from that of the baseline system.

The validation data available to date shows that the model is adequate for the purposes it was intended: i.e., to predict coherence traffic accurately enough so that the overall performance prediction remains within a few percent of the actual values.

The future work on the subject includes a more extensive validation of the model for SMP systems and clusters. Because of the typical market segmentation (small SMP systems for front-end use and large cluster systems for back-end use), it is difficult to obtain measurements for the same workload across a wide range of architectural parameters. In a way, this segmentation also helps since a model calibrated for a 2P SMP system is unlikely to be used for a 16P cluster system. Another issue that we wish to pursue aggressively is to obtain detailed address traces for a variety of traffic and hardware parameters and run them through SMP cache simulators. This work should further help in validating and further extending the model.

## References

- [1] A. Agrawal, M. Horowitz, and J. Hennessy, "An Analytical Cache Model", *ACM transactions on Computer Systems*, Vol 7, No 2, May 1989, pp 184-215.
- [2] D.E. Culler and J.P. Singh, *Parallel Computer Architecture – A hardware/software approach*, Morgan Kaufmann, 1999.
- [3] K. Kant & Y. Won, "Server Capacity Planning for Web Traffic Workload", To appear in *IEEE transactions on knowledge and data engineering*.
- [4] K. Kant and C.R.M. Sundaram, "A Server Performance Model for Static Web Workloads", to appear in *proceedings of ISPASS 2000*.
- [5] K. Kant and Y. Won, "Performance Implications of File Caching in SPECweb99 Workload", *Proc. of 2nd IEEE Conference on Workload Characterization*, Austin TX, Oct 1999.
- [6] Tom Kerrigan and B. Seevers, "Impact of Back Invalidates on Saber System Performance", Intel Internal report, Oct 98.