Analytical Cache Models with Applications to Cache Partitioning

G. Edward Suh, Srinivas Devadas, and Larry Rudolph
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
{suh,devadas,rudolph}@mit.edu

ABSTRACT
An accurate, tractable, analytic cache model for time-shared systems is presented, which estimates the overall cache miss-rate of a multiprocessing system with any cache size and time quanta. The input to the model consists of the isolated miss-rate curves for each process, the time quanta for each of the executing processes, and the total cache size. The output is the overall miss-rate. Trace-driven simulations demonstrate that the estimated miss-rate is very accurate. Since the model provides a fast and accurate way to estimate the effect of context switching, it is useful for both understanding the effect of context switching on caches and optimizing cache performance for time-shared systems. A cache partitioning mechanism is also presented and is shown to improve the cache miss-rate up to 25% over the normal LRU replacement policy.

1. INTRODUCTION
This paper presents an analytical model for the behavior of a cache in a multiprocessing system that can accurately estimate overall miss-rate for any cache size and any time quantum. An evaluation method for miss-rate is essential to optimize cache performance. Traditional cache performance evaluation is done by simulations [25, 16, 12], which provide accurate results, but simulation time is often long. Hardware monitoring can dramatically speed up the process [26], however, it is limited to the particular cache configuration. As a result, both simulations and hardware monitoring can only be used to evaluate the effect of context switches [14, 10]. Moreover, simulations and monitoring rarely provide intuitive understanding making it difficult to improve cache performance. To provide both performance prediction and insight into improving performance, analytical cache models are required.

We use our model to determine the best cache partitioning so as to improve performance. Partitioning is needed to mitigate the effects of conflicts among concurrently executing processes, especially for large caches. In the past, caches were small and it was best to let each process consume the entire cache space, since process footprints were much larger than the cache. In modern microprocessors, caches are much larger; some Level 1 (L1) caches range up to one MB [5], and L2 caches are up to several MB [2, 13]. Large caches allow potential performance improvement by partitioning. Since each process may not need the entire cache space, the effect of context switches can be mitigated by keeping useful data in the cache over context switches. It is crucial for modern microprocessors to minimize inter-process conflicts by proper cache partitioning [21, 9] or scheduling [17, 23].

Our model requires information that is relatively easy to acquire. The characteristics for each process are given by the miss-rate as a function of cache size when the process is isolated, which can be easily obtained either on-line or off-line. The time quantum for each process and cache size are also given as inputs to the model. With this information, the model estimates the overall miss-rate for a given cache size when an arbitrary combination of processes is run. The model provides good estimates for any cache size and any time quantum, and is easily applied to real problems since the input miss-rate curves are both intuitive and easy to obtain in practice. Therefore, we believe that the model is useful for any study related to the effect of context switches on cache memory.

After describing related research in Section 2, Section 3 derives an analytical cache model for time-shared systems. Section 4 discusses cache partitioning based on the model and evaluates the model-based partitioning method by simulations. Finally, Section 5 concludes the paper.

2. RELATED WORK
Several early investigations of the effects of context switches use analytical models. Thiébaut and Stone [20] modeled the amount of additional misses caused by context switches for set-associative caches. Agarwal, Horowitz and Hennessy [1] also included the effect of conflicts between processes in their analytical cache model and showed that inter-process conflicts are noticeable for a mid-range of cache sizes that are large enough to have a considerable number of conflicts but not large enough to hold all the working sets. However, these models work only for long enough time quanta, and require information that is hard to collect on-line.

Mogul and Borg [14] studied the effect of context switches through trace-driven simulations. Using a timesharing system simulator, their research shows that system calls, page faults, and a scheduler are the main sources of context switches. They also evaluated the effect of context switches on cycles per instruction (CPI) as well as the cache miss-rate. Depending on cache parameters, the cost of a context switch appears to be in the thousands of cycles, or tens to hundreds of microseconds in their simulations.

Stone, Turek and Wolf [18] investigated the optimal allocation of cache memory between two competing processes that minimizes the overall miss-rate of a cache. Their study focuses on the partitioning of instruction and data streams, which can be thought of as multitasking with a very short time quantum. Their model for this case shows that the optimal allocation occurs at a point where the miss-rate derivatives of the competing processes are equal. The LRU replacement policy appears to produce cache allocations very close to optimal for their examples. They also describe a new replacement policy for longer time quanta that only increases cache allocation based on time remaining in the current time quantum and the marginal reduction in miss-rate due to an increase in cache allocation. However, their policy simply assumes the probability for a evicted block to be accessed in the next time quantum as a constant, which is neither validated nor is it described how this probability is obtained.

Thiébaut, Stone and Wolf applied their partitioning work [18] to improve disk cache hit-ratios [21]. The model for tightly interleaved streams is extended to be applicable for more than two processes. They also describe the problems in applying the model in practice, such as approximating the miss-rate derivative, non-monotonic miss-rate derivatives, and updating the partition. Trace-driven simulations for 32-MB disk caches show that the partitioning improves the relative hit-ratios in the range of 1% to 2% over the LRU policy.

Our analytical model and partitioning differ from previous efforts that tend to focus on some specific cases of context switches. Our model works for any specific time quanta, whereas the previous models focus only on long time quanta. Also, our partitioning works for any time quanta, whereas Thiébaut's algorithms only works for very short time quanta. Moreover, the inputs of our model (miss-rates) are much easier to obtain compared to footprints or the number of unique cache blocks that previous models require.

3. ANALYTICAL CACHE MODEL

The analytical cache model estimates the overall cache miss-rate for a multi-processing system. The cache size and the time quantum length for each job is known. The cache size is given by the number of cache blocks, and the time quantum is given by the number of memory references. Both are assumed to be constants (See Figure 1 (a)). In addition, associated with each job is its miss-rate curve, i.e., the number of cache misses as a function of the cache size.

This section explains the development of the model in several steps. Heavy use is made of the individual, isolated miss-rate curve (iimr). This curve is the miss-rate for a process as a function of cache size assuming no other processes are running. There is much information that can be gleaned from this equation. For example, we can compute the miss rate of a process as a function of time (Section 3.2.1) from the miss-rate of a process as a function of space.

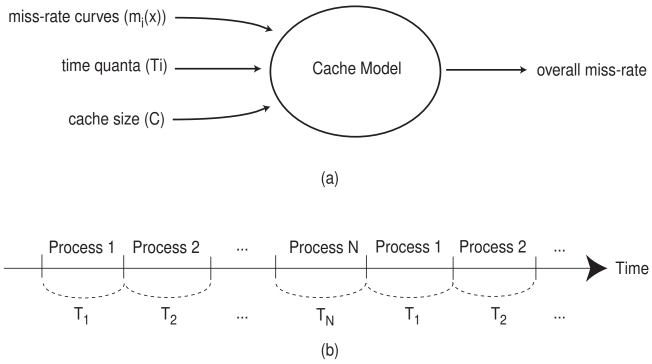

Figure 1: (a) The overview of an analytical cache model. (b) Round-robin schedule.

Observe that as a process executes, it either references an item in the cache, in which case its footprint size remains the same, or it gets a cache miss thereby increasing its footprint size. In other words, we know how much cache is allocated to a process as a function of time: from the iimr curve, we compute the independent, isolated footprint as a function of time (iifp) (Section 3.2.2).

If one knows how much cache is allocated to a process when it begins executing its time quantum and how much more cache it will need during the execution of that time quantum, we can compute how much cache will be left for the next process that is about to begin its time quantum execution. In other words, from the iifp curves of all the concurrent processes, we compute the individual, dependent footprint (dfp) as a function of time (Section 3.2.3).

At each time step, we know how much cache is allocated to the running process (from dfp(t)) and we know the miss rate for that size (from iimr(S)) for the executing process and so we can get the dependent miss rate as a function of time (dmr(t)) (Section 3.2.1).

Finally, integrating or summing the dmr(t) over time, gives the overall average miss rate for a given cache size, given time quantum sizes, and a given set of concurrent processes (Section 3.2.4).

The following subsection gives an overview of our assumptions. The development of the cache model is then presented, following the outline given above. Finally, this section ends with experimental verification of the model.

3.1 Assumptions

The memory reference pattern of each process is assumed to be represented by a miss-rate curve that is a function of the cache size. Moreover, this miss-rate curve is assumed not to change over time. Although real applications do have

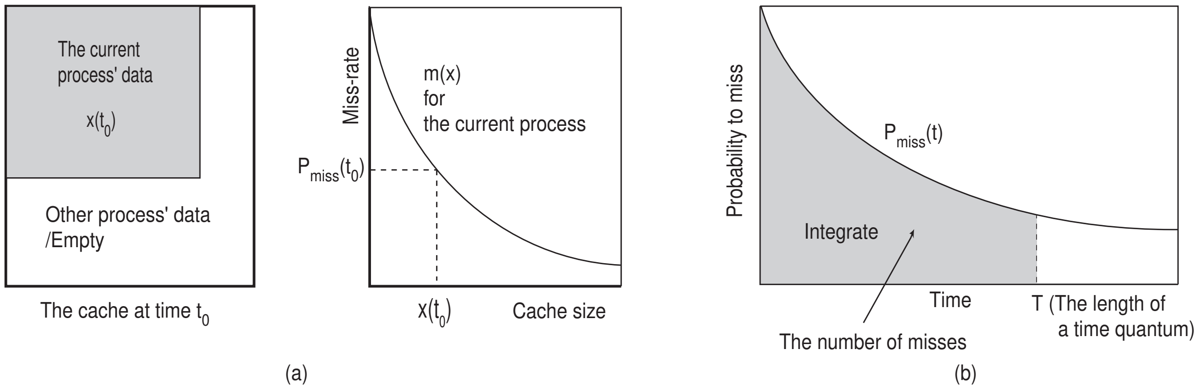

Figure 2: (a) The probability of a miss at time t_0. (b) The number of misses from $P_{miss}(t)$ curve.

dynamically changing memory reference patterns, our results show that, in practice, an average miss-rate function works very well. For abrupt changes in the reference pattern, multiple miss-rate curves can be used to estimate an overall miss-rate.

There is no shared address space among processes. This assumption is true for common cases where each process has its own virtual address space and the shared memory space is negligible compared to the entire memory space that is used by a process.

Finally, a round-robin scheduling policy with a fixed time quantum for each process is assumed (see Figure 1 (b)), an LRU replacement policy is used, and the cache is fully associative. Although most real caches are set-associative, a model for fully-associative caches is very useful for understanding the effect of context switches because the model is simple. Moreover, cache partitioning experiments demonstrate that the fully-associative model can also be applied to set-associative caches in practice (Section 4). Elsewhere, we have extended the model to handle set-associative caches [19]. A model assuming many other scheduling methods and replacement policies can be similarly derived.

We make use of the following notations:

t the number of memory references from the beginning of a time quantum.

$x(t)$ the number of cache blocks that belong to a process after t memory references.

$m(x)$ the steady-state miss-rate for a process with cache size x.

T the number of memory references in a time quantum.

3.2 Cache Model

The goal is to predict the average miss-rate for a multiprocess machine with a given cache size and set of processes.

3.2.1 *Miss rate as function of time*

Given the independent, isolated miss-rate of a process as a function of cache size, we compute its miss-rate as a function of time. Let time t start at the beginning of a time quantum, not at the beginning of execution. Since all time quanta for a process are identical by our assumptions, we consider only one time quantum for each process.

Although the cache size is C, at certain times, it is possible that only part of the cache is filled with the current process' data (Figure 2 (a) shows a snapshot of a cache at time t_0). Therefore, the effective cache size at time t_0 can be thought of as the amount of the current process' data $x(t_0)$ in the cache at that time. The probability of a cache miss in the next memory reference is given by

$$P_{miss}(t_0) = m(x(t_0)). \qquad (1)$$

Once we have $P_{miss}(t_0)$, it is easy to estimate the miss-rate over time during that time quantum. The number of misses for the process over a time quantum can be expressed as a simple integral, Figure 2 (b), where the miss-rate is expressed as the number of misses divided by the number of memory references.

$$\text{miss-rate} = \frac{1}{T}\int_0^T P_{miss}(t)dt = \frac{1}{T}\int_0^T m(x(t))dt \qquad (2)$$

3.2.2 *Footprint as a function of time*

We now estimate $x(t)$, the amount of a process' data, i.e. its footprint, in a cache as a function of time. Let us begin with the assumption that a process starts executing during a time quantum with an empty cache in order to estimate cache performance for cases when a cache gets flushed for every context switch. Virtual address caches without process ID are good examples of such a case. We show later how to estimate $x(t)$ when the cache is not empty at the start of a time quantum.

Consider $x^\infty(t)$ as the amount of the current process' data at time t for an infinite size cache. We assume that the process starts with an empty cache at time 0. There are two possibilities for $x^\infty(t)$ at time $t+1$. If the $(t+1)^{th}$ memory reference results in a cache miss, a new cache block is brought into the cache. As a result, the amount of the process's cache data increases by one block. Otherwise, the amount of data remains the same. Therefore, the amount of the process' data in the cache at time $t+1$ is given by

$$x^{\infty}(t+1) = \begin{cases} x^{\infty}(t)+1 & (t+1)^{th} \text{ reference misses} \\ x^{\infty}(t) & \text{otherwise.} \end{cases} \quad (3)$$

Since the probability for the $(t+1)^{th}$ memory reference to miss is $m(x^{\infty}(t))$ from Equation 1, the expected value of $x(t+1)$ can be written by

$$\begin{aligned} E[x^{\infty}(t+1)] &= E[x^{\infty}(t) \cdot (1 - m(x^{\infty}(t))) \\ &\quad + (x^{\infty}(t)+1) \cdot m(x^{\infty}(t))] \\ &= E[x^{\infty}(t) + 1 \cdot m(x^{\infty}(t))] \\ &= E[x^{\infty}(t)] + E[m(x^{\infty}(t))]. \end{aligned} \quad (4)$$

Assuming that $m(x)$ is convex[1], we can use Jensen's inequality [3] and rewrite the equation as a function of $E[x^{\infty}(t)]$.

$$E[x^{\infty}(t+1)] \geq E[x^{\infty}(t)] + m(E[x^{\infty}(t)]). \quad (5)$$

Usually, a miss-rate changes slowly. As a result, for a short interval such as from x to $x+1$, $m(x)$ can be approximated as a straight line. Since the equality in Jensen's inequality holds if the function is a straight line, we can approximate the amount of data at time $t+1$ as

$$E[x^{\infty}(t+1)] \simeq E[x^{\infty}(t)] + m(E[x^{\infty}(t)]). \quad (6)$$

We can calculate the expectation of $x^{\infty}(t)$ more accurately by calculating the probability for every possible value at time t [19]. However, calculating a set of probabilities is computationally expensive. Also, our experiments show that the above approximation closely matches simulation results.

If we further approximate the amount of data $x^{\infty}(t)$ to be the expected value $E[x^{\infty}(t)]$, $x^{\infty}(t)$ can be expressed with a differential equation:

$$x^{\infty}(t+1) - x^{\infty}(t) = m(x^{\infty}(t)), \quad (7)$$

which can be easily calculated in a recursive manner.

To obtain a closed form solution, we can rewrite the discrete form of the differential equation 7 to a continuous form:

$$\frac{dx^{\infty}}{dt} = m(x^{\infty}). \quad (8)$$

Solving the differential equation by separating variables, the differential equation becomes

$$t = \int_{x^{\infty}(0)}^{x^{\infty}(t)} \frac{1}{m(x')} dx'. \quad (9)$$

We define a function $M(x)$ as an integral of $1/m(x)$, which means that $dM(x)/dx = 1/m(x)$, and then $x^{\infty}(t)$ can be written as a function of t:

$$x^{\infty}(t) = M^{-1}(t + M(x^{\infty}(0))) \quad (10)$$

where $M^{-1}(x)$ represents the inverse function of $M(x)$.

Finally, for a finite size cache, the amount of data in the cache is limited by the size of the cache C. Therefore, $x^{\phi}(t)$, the amount of a process' data starting from an empty cache, is written by

$$x^{\phi}(t) = MIN[x^{\infty}(t), C] = MIN[M^{-1}(t + M(0)), C]. \quad (11)$$

[1]If a replacement policy is smart enough, the marginal gain of having one more cache block monotonically decreases as we increase the cache size.

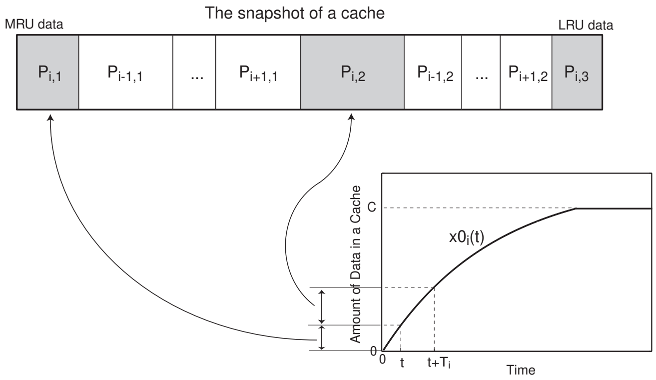

Figure 3: The snapshot of a cache after running Process i for time t.

3.2.3 *Individual, Dependent Footprint as a function of time*

We now compute the amount of a process' data at time t when the cache is not flushed at a context switch, i.e., the dependent case. To distinguish between the processes, a subscript i is used to represent Process i. For example, $x_i(t)$ represents the amount of Process i's data at time t.

The estimation of $x_i(t)$ is based on round-robin scheduling (See Figure 1 (b)) and the LRU replacement policy. Process i runs for a fixed length time quantum T_i. For simplicity, processes are assumed to be of infinite length so that there is no change in the scheduling. Also, the initial startup transient from an empty cache is ignored since it is negligible compared to the steady state.

To estimate the amount of a process' data at a given time, imagine the snapshot of a cache after executing Process i for time t as shown in Figure 3. Note that time is 0 at the beginning of the process' time quantum. In the figure, the blocks on the left side show recently used data, and blocks on the right side show old data. $P_{j,k}$ represents the data of Process j, and subscript k specifies the most recent time quantum when the data are referenced. From the figure, we can obtain $x_i(t)$ once we know the size of all $P_{j,k}$ blocks.

The size of each block can be estimated using the $x_i^{\phi}(t)$ curve from Equation 11, which is the amount of Process i's data when the process starts with an empty cache. Since $x_i^{\phi}(t)$ can also be thought of as the amount of data that are referenced from time 0 to time t, $x_i^{\phi}(T_i)$ is the amount of data that are referenced over one time quantum. Similarly, we can estimate the amount of data that are referenced over k recent time quanta to be $x_i^{\phi}(k \cdot T_i)$. As a result, the size

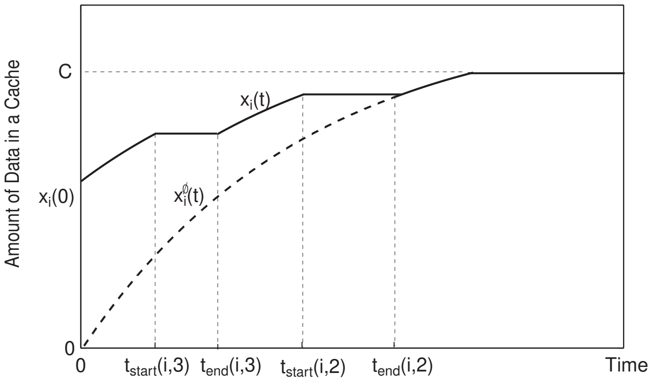

Figure 4: The relation between $x_i^\phi(t)$ and $x_i(t)$. $x_i(0)$ is the amount of Process i's data in the cache when a time quantum starts.

of Block $P_{j,k}$ can be written as

$$P_{j,k} = \begin{cases} x_j^\phi(t+(k-1)\cdot T_j) - x_j^\phi(t+(k-2)\cdot T_j) & \text{if } j \text{ is executing} \\ x_j^\phi(k\cdot T_j) - x_j^\phi((k-1)\cdot T_j) & \text{otherwise} \end{cases} \tag{12}$$

where we assume that $x_j^\phi(t) = 0$ if $t < 0$.

$x_i(t)$ is the sum of $P_{i,k}$ blocks that are inside the cache of size C in Figure 3. If we define $l_i(t)$ as the maximum integer value that satisfies the following inequality, then $l_i(t)+1$ represents how many $P_{i,k}$ blocks are in the cache.

$$\sum_{k=1}^{l_i(t)} \sum_{j=1}^{N} P_{j,k} = x_i^\phi(t+(l_i(t)-1)\cdot T_i) + \sum_{j=1, j\neq i}^{N} x_j^\phi(l_i(t)\cdot T_j) \leq C \tag{13}$$

where N is the number of processes. From $l_i(t)$ and Figure 3, the estimated value of $x_i(t)$ is

$$x_i(t) = \begin{cases} x_i^\phi(t+l_i(t)\cdot T_i) & \text{if } x_i^\phi(t+l_i(t)\cdot T_i) + \sum_{j=1,j\neq i}^{N} x_j^\phi(l_i(t)\cdot T_j) \leq C \\ C - \sum_{j=1,j\neq i}^{N} x_j^\phi(l_i(t)\cdot T_j) & \text{otherwise} \end{cases} \tag{14}$$

Figure 4 illustrates the relation between $x_i^\phi(t)$ and $x_i(t)$. In the figure $l_i(t)$ is assumed to be 2. Unlike the cache flushing case, a process can start with some of its data left in the cache. The amount of initial data $x_i(0)$ is given by Equation 14. If the least recently used (LRU) data in a cache does not belong to Process i, $x_i(t)$ increases the same as $x_i^\phi(t)$. However, if the LRU data belongs to Process i, $x_i(t)$ does not increase on a cache miss since Process i's block gets replaced.

Define $t_{start}(j,k)$ as the time when the k^{th} MRU block of Process j ($P_{j,k}$) becomes the LRU part of a cache, and $t_{end}(j,k)$ as the time when $P_{j,k}$ gets completely replaced from the cache (See Figure 3). $t_{start}(j,k)$ and $t_{end}(j,k)$ specify the flat segments in Figure 4 and can be estimated from the following equations that are based on Equation 12.

$$x_j^\phi(t_{start}(j,k)+(k-1)\cdot T_j) + \sum_{p=1,p\neq j}^{N} x_p^\phi((k-1)\cdot T_p) = C. \tag{15}$$

$$x_j^\phi(t_{end}(j,k)+(k-2)\cdot T_j) + \sum_{p=1,p\neq j}^{N} x_p^\phi((k-1)\cdot T_p) = C. \tag{16}$$

$t_{start}(j, l_j(t)+1)$ would be zero if Equation 15 is satisfied when $t_{start}(j, l_j(t)+1)$ is negative, which means that the $P(j, l_j(t)+1)$ block is already the LRU part of the cache at the beginning of a time quantum.

3.2.4 *Overall Miss-rate*

This section presents the overall miss-rate calculation. When a cache uses virtual address tags and gets flushed for every context switch, each process starts a time quantum with an empty cache. In this case, the miss-rate of a process can be estimated from the results of Section 3.2.1 and 3.2.2. From Equation 2 and 11, the miss-rate for Process i can be written by

$$\text{miss-rate}_i^\phi = \frac{1}{T_i}\int_0^{T_i} m_i(MIN[M_i^{-1}(t+M_i(0)), C])dt. \tag{17}$$

If a cache uses physical address tags or has a process' ID with virtual address tags, it does not have to be flushed at a context switch. In this case, the amount of data $x_i(t)$ is estimated in Section 3.2.3. The miss-rate for Process i can be written by

$$\text{miss-rate}_i = \frac{1}{T_i}\int_0^{T_i} m_i(x_i(t))dt \tag{18}$$

where $x_i(t)$ is given by Equation 14.

For actual calculation of the miss-rate, $t_{start}(j,k)$ and $t_{end}(j,k)$ from Equation 15 and 16 can be used. Since $t_{start}(j,k)$ and $t_{end}(j,k)$ specify the flat segments in Figure 4, the miss-rate of Process i can be rewritten by

$$\begin{aligned} \text{miss-rate}_i = \frac{1}{T_i}\{ & \int_0^{T_i'} m_i(MIN[M_i^{-1}(t+M_i(x_i(0))), C])dt \\ & + \sum_{k=d_i}^{l_i(t)+1} m_i(x_i^\phi(t_{start}(i,k)+(k-1)\cdot T_i)) \\ & \cdot (MIN[t_{end}(i,k), T_i] - t_{start}(i,k))\} \end{aligned} \tag{19}$$

where d_i is the minimum integer value that satisfies $t_{start}(i,d_i) < T_i$. T_i' is the time that Process i actually grows.

$$T_i' = T_i - \sum_{k=d_i}^{l_i(t)+1} (MIN[t_{end}(i,k), T_i] - t_{start}(i,k)). \tag{20}$$

As shown above, calculating a miss-rate could be complicated if we do not flush a cache at a context switch. If we assume that the executing process' data left in a cache is all in the most recently used part of the cache, we can use the

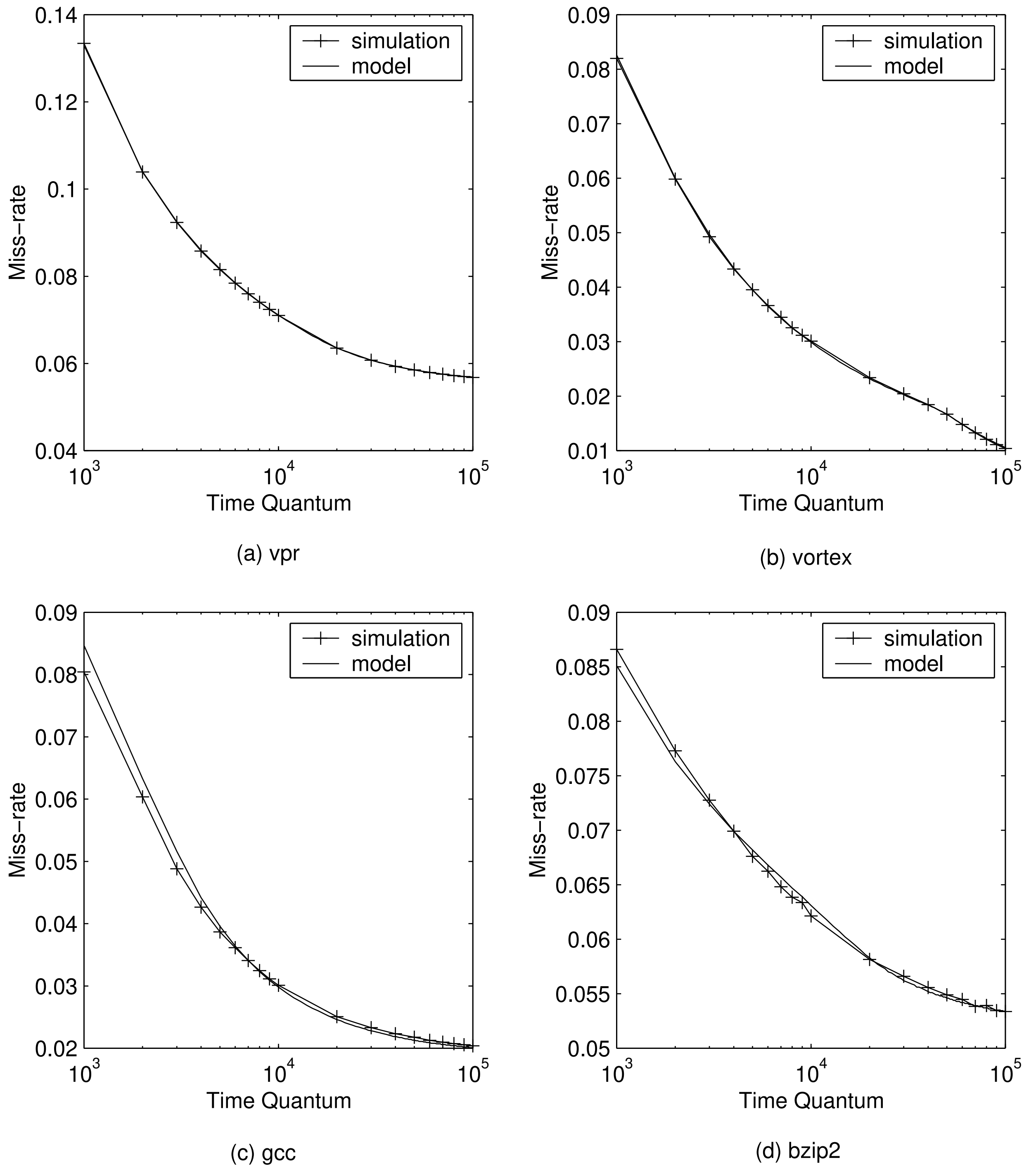

Figure 5: The result of the cache model for cache flushing cases. (a) `vpr`. (b) `vortex`. (c) `gcc`. (d) `bzip2`.

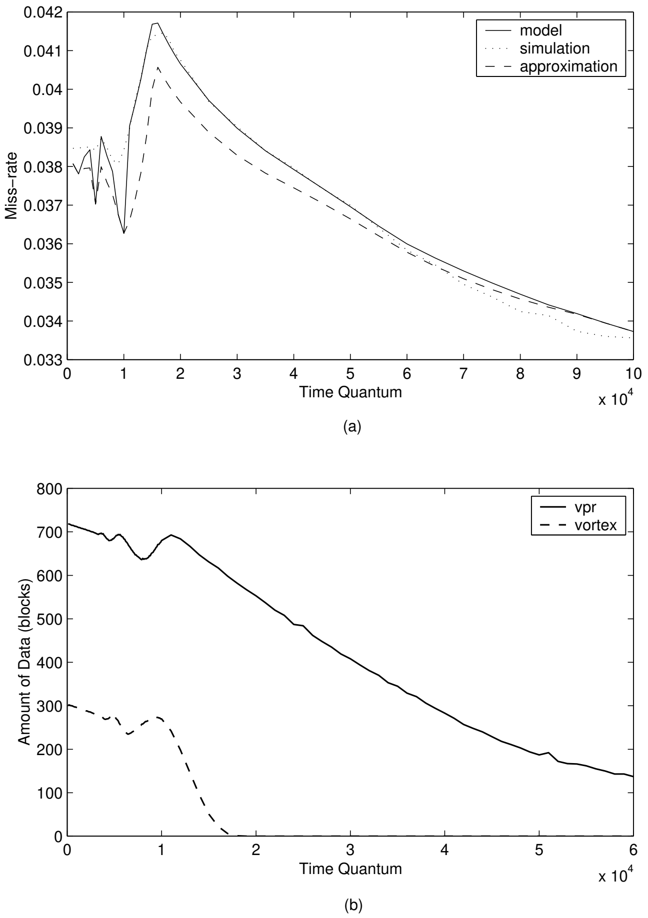

Figure 6: The result of the cache model when two processes (vpr, vortex) are sharing a cache (32 KB fully-associative). (a) the overall miss-rate. (b) the initial amount of data $x_i(0)$.

equation for estimating the amount of data starting with an empty cache. Therefore, the calculation can be much simplified as follows,

$$\overline{\text{miss-rate}_i} = \frac{1}{T_i}\int_0^{T_i} m_i(MIN[M_i^{-1}(t + M_i(x_i(0))), C])dt \quad (21)$$

where $x_i(0)$ is estimated from Equation 14. The effect of this approximation is evaluated in the experiment section (cf. Section 3.3).

Once we calculate the miss-rate of each process, the overall miss-rate is straightforwardly calculated from those miss-rates.

$$\text{Overall miss-rate} = \frac{\sum_{i=1}^{N} \text{miss-rate}_i \cdot T_i}{\sum_{i=1}^{N} T_i} \quad (22)$$

3.3 Experimental Verification

Our cache model can be validated by comparing estimated miss-rate predictions with simulation results. Several combinations of benchmarks are modeled and simulated for various time quanta. First, we simulate cases when a cache gets flushed at every context switch, and compare the results with the model's estimation. Cases without cache flushing are also tested. For the cases without cache flushing, both the complete model (Equation 19) and the approximation (Equation 21) are used to estimate the overall miss-rate. Based on the simulation results, the error caused by the approximation is discussed.

3.3.1 Cache Flushing Case

The results of the cache model and simulations are shown in Figure 5 in cases when a process starts its time quantum with an empty cache. Four benchmarks from SPEC CPU2000 [7], which are `vpr`, `vortex`, `gcc` and `bzip2`, are tested. The cache is a 32-KB fully-associative cache with 32-Byte blocks. The miss-rate of a process is plotted as a function of the length of a time quantum, and shows a good agreement between the model's estimation and the simulation result.

As inputs to the cache model, the average miss-rate of each process has been obtained from simulations. Each process has been simulated for 25 million memory references, and the miss-rates of the process for various cache size have been recorded. The simulation results were also obtained by simulating benchmarks for 25 million memory references with flushing a cache every T memory references. As the result shows, the average miss-rate works very well.

3.3.2 General Case

Figure 6 shows the result of the cache model when two processes are sharing a cache. The two benchmarks are `vpr` and `vortex` from SPEC CPU2000, and the cache is a 32-KB fully-associative cache with 32-Byte blocks. The overall miss-rates are shown in Figure 6 (a). As shown in the figure, the miss-rate estimated by the model shows a good agreement with the results of the simulations.

The figure also shows an interesting fact that a certain range of time quanta could be very problematic for cache performance. For short time quanta, the overall miss-rate is relatively small. For very long time quanta, context switches do not matter since a process spends most of its time in the steady state. However, medium time quanta could severely degrade cache miss-rates as shown in the figure. This problem occurs when a time quantum is long enough to pollute the cache but not long enough to compensate for the misses caused by context switches. The problem becomes clear in Figure 6 (b). The figure shows the amount of data left in the cache at the beginning of a time quantum. Comparing Figure 6 (a) and (b), we can see that the problem occurs when the initial amount of data rapidly decreases.

The error caused by our approximation (Equation 21) method can be seen in Figure 6. In the approximation, we assume that the data left in the cache at the beginning of a time quantum are all in the MRU region of the cache. In reality, however, the data left in the cache could be the LRU cache blocks and get replaced before other process' blocks in the cache, although the current process's data are likely to be accessed in the time quantum. As a result, the approximated miss-rate is lower than the simulation result when the initial amount of data is not zero.

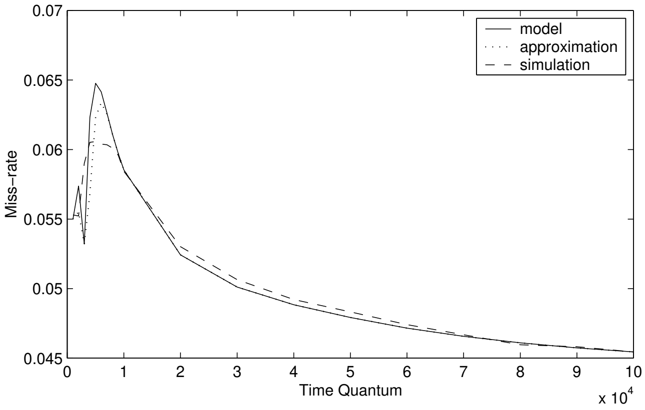

Figure 7: The overall miss-rate when four processes (`vpr`, `vortex`, `gcc`, `bzip2`) are sharing a cache (32 KB, fully-associative).

A four-process case is also tested in Figure 7. Two more benchmarks, `gcc` and `bzip2`, from SPEC CPU2000 [7] are added to `vpr` and `vortex`, and the same cache configuration is used as the two process case. The figure also shows a very close agreement between the miss-rate estimated by the cache model and the miss-rate from simulations. The problematic time quanta and the effect of the approximation have changed. Since there are more processes polluting the cache as compared to the two process case, a process experiences an empty cache in shorter time quanta. As a result, the problematic time quanta become shorter. On the other hand, the effect of the approximation is less harmful in this case. This is because the error in one process' miss-rate becomes less important as we have more processes.

4. CACHE PARTITIONING

This section shows how the analytical cache model can be used to dynamically partition the cache. A partitioned cache allocates cache space to particular processes. This space is dedicated to the process and cannot be used to satisfy cache misses by other processes. Using trace-driven simulations, we compare partitioning with the normal LRU. The partitioning is based on the fully-associative cache model. However, simulation results demonstrate that this implementation works for both fully-associative caches and set-associative caches.

4.1 Recording Memory Reference Patterns

The miss-rate curves for each process are generated off-line. We record the miss-rate curve for each process to represent its memory reference pattern. For various cache sizes, a single process cache simulator is applied to each process. This information can be reused for any combination of processes as long as the cache configuration is the same[2].

To incorporate the dynamically changing behavior of a process, a set of miss-rate curves, one for each time period, are produced. At run-time, the miss-rate curve is mapped to the appropriate time quantum.

[2]Note that for our fully-associative model, only the cache block size matters

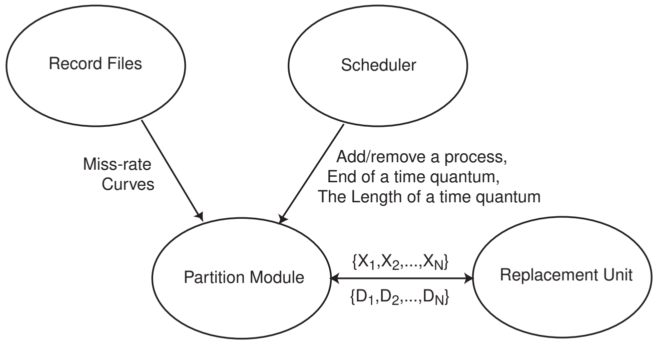

Figure 8: The implementation of on-line cache partitioning.

4.2 The Partitioning Scheme

The overall flow of the partitioning scheme can be viewed as a set of four modules: off-line recording, scheduler information, allocation, and replacement (Figure 8). The scheduler provides the partition module with the set of executing processes and their start/end times. The partition module uses the miss-rate information for the processes to calculate cache partitions at the end of each time quantum. Finally, the replacement unit maps these partitions to the appropriate parts of the cache.

The partition module decides the number of cache blocks that should be dedicated to a process (D_i). The D_i most recently used cache blocks of Process i are kept in the cache over other process' time quanta, and Process i starts its time quantum with those cache blocks in the cache. During its own time quantum, Process i can use all cache blocks that are not reserved for other processes ($S = C - \sum_{j=1,j\neq i}^{N} D_j$).

In addition to LRU information, our replacement decision depends on the number of cache blocks that currently belong to each process (X_i), that is, the number of cache lines in the cache that currently contain memory of that process. The LRU cache block of an active process (i) is chosen if its allocation is larger than its current use ($D_i + S \geq X_i$). Otherwise, the LRU cache block of a dormant overallocated process is chosen. For set-associative caches, there may be no cache block of the desired process in the set. In this case, the LRU cache block of the set is replaced.

For set-associative caches, the fully-associative replacement policy may result in replacing recently used data to keep useless data. Imagine the case when a process starts to heavily access two or more addresses that happen to be mapped to the same set. If the process already has many cache blocks in other sets, our partitioning will allocate only a few cache blocks in the accessed set for the process, causing lots of conflict misses. To solve this problem, we can use better mapping functions [22, 6] or a victim cache [8].

When a Process i first starts, D_i is set to zero since there is no cache block that belongs to the process. At the end

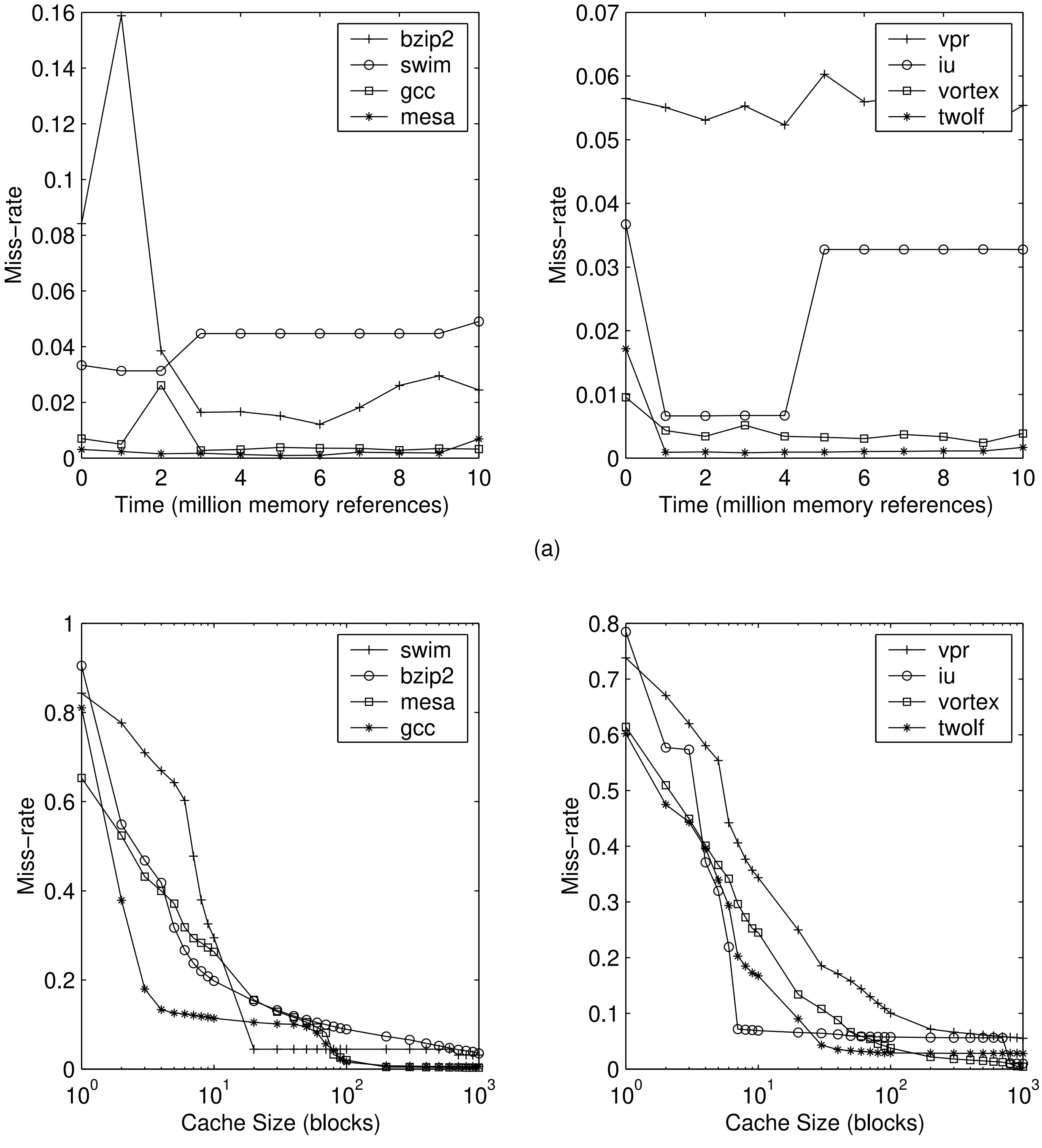

Figure 9: The characteristics of the benchmarks. (a) The change of a miss-rate over time. (b) The miss-rate as a function of the cache size.

of Process i's time quantum, the partition module updates the information such as the miss-rate curve($m_i(x)$) and the time quantum(T_i). If there is any change, D_i is also updated based on the cache model.

A cache partition specifies the amount of data in the cache at the beginning of a process' time quantum (D_i), and the maximum cache space the process can use ($C - \sum_{j=1,j\neq i}^{N} D_j$). Therefore, the number of misses for a process over one time quantum can be estimated from Equation 21:

$$\text{miss}_i = \int_0^{T_i} m_i(MIN[M_i^{-1}(t + M_i(D_i)), C - \sum_{j=1,j\neq i}^{N} D_j])dt \tag{23}$$

where C is cache size, and N is the number of processes sharing the cache.

The new value of D_i is the integer, in the range $[0, X_i]$, that minimizes the total number of misses that is given by the following quantity:

$$\sum_{p=1}^{N} \int_0^{T_p} m_p(MIN[M_p^{-1}(t + M_p(D_p)), C - \sum_{q=1,q\neq p}^{N} D_q])dt. \tag{24}$$

4.3 Experimental Verification

The case of eight processes sharing a 32-KB cache is simulated to evaluate model-based partitioning. Seven benchmarks (`bzip2`, `gcc`, `swim`, `mesa`, `vortex`, `vpr`, `twolf`) are from SPEC CPU2000 [7], and one (the image understanding program (`iu`)) is from a data intensive systems benchmark suite [15]. The overall miss-rate with partitioning is compared to the miss-rate only using the normal LRU replacement policy.

The simulations are carried out for fifty million memory references for each time quantum. Processes are scheduled in a round-robin fashion with the fixed number of memory references per time quantum. Also, the number of memory references per time quantum is assumed to be the same for the all eight processes. Finally, two record cycles (P), of ten million and one hundred thousand memory references, respectively, are used for the model-based partitioning. The record cycle represents how often the miss-rate curve is recorded for the off-line profiling. Therefore, a shorter record cycle implies more detailed information about a process' memory reference pattern.

The characteristics of the benchmarks are illustrated in Figure 9. Figure 9 (a) shows the change of a miss-rate over time. The x-axis represents simulation time. The y-axis represents the average miss-rate over one million memory references at a given time. As shown in the figure, `bzip2`, `gcc`, `swim` and `iu` show abrupt changes in their miss-rate, whereas other benchmarks have very uniform miss-rate characteristics over time. Figure 9 (b) illustrates the miss-rate as a function of the cache size. For a 32-KB fully-associative cache, benchmarks show miss-rates between 1% and 5%.

4.3.1 *Fully-Associative Result*

The results of cache partitioning for a fully-associative cache are shown in Figure 10. In Figure 10 (a), the miss-rates are averaged over 50 million memory references and shown for various time quanta. As discussed in the cache model, the normal LRU replacement policy is problematic for a certain range of time quanta. In this case, the overall miss-rate increases dramatically for time quanta between one thousand and ten thousand memory references. For this problematic region, the model-based partitioning improves the cache miss-rate by lowering it from 4.6% to 3.4%, which is about a 25% improvement. For short time quanta, the relative improvement is about 7%. For very long time quanta, the model-based partitioning shows the exact same result as the normal LRU replacement policy. In general, it is shown by the figure that the model-based partitioning always performs at least as well as or better than the normal LRU replacement policy. Also, the partitioning with a short record cycle performs better than the partitioning with a long record cycle.

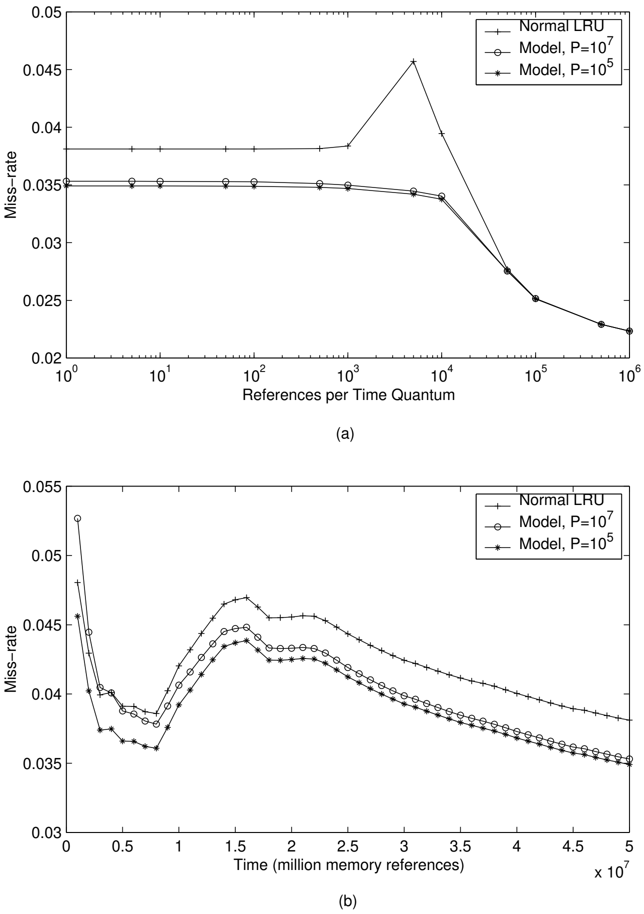

Figure 10: The results of the model-based cache partitioning for a fully-associative cache when eight processes (`bzip2`, `gcc`, `swim`, `mesa`, `vortex`, `vpr`, `twolf`, `iu`) are sharing the cache (32 KB, fully associative). (a) the average miss-rate for various time quanta. (b) the change of the miss-rate over time with ten memory references per time quantum.

In our example of a 32-KB cache with eight processes (Fig-

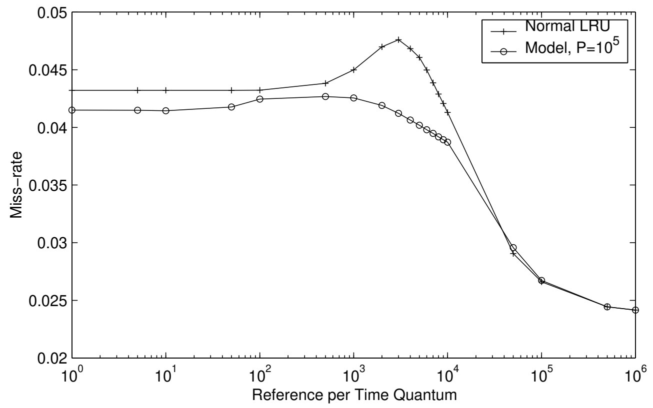

Figure 11: The results of the model-based cache partitioning for a set-associative cache when eight processes (bzip2, gcc, swim, mesa, vortex, vpr, twolf, iu) are sharing the cache (32 KB, 8-way associative).

ure 10), the problematic time quanta are in the order of a thousand memory references, which is very short for modern microprocessors. As a result, only systems with very fast context switching, such as simultaneous multi-threading machines [24, 11, 4], can be improved for this cache size and workload. However, longer time quanta become problematic if a cache is larger. Therefore, conventional time-shared systems with very high clock frequency can also be improved by the same technique if a cache is large.

Figure 10 (b) shows the change of a miss-rate over time rather than an average miss-rate over the entire simulation. It is clear from the figure how the short record cycle helps partitioning. In the figure, the model-based partitioning with the long record cycle ($P = 10^7$) performs worse than LRU at the beginning of a simulation, even though it outperforms the normal LRU replacement policy overall. This is because the model-based partitioning has only one average miss-rate curve for a process. As shown in Figure 9, some benchmarks such as bzip2 and gcc have a very different miss-rate at the beginning. Therefore, the average miss-rate curves for those benchmarks do not work at the beginning of the simulation, which results in worse performance than the normal LRU replacement policy. The model-based partitioning with the short record cycle ($P = 10^5$), on the other hand, always outperforms the normal LRU replacement policy. In this case, the model has correct miss-rate curves for all the time quanta, and partitions the cache properly even for the beginning of processes.

4.3.2 Set-Associative Result

The result of cache partitioning for a set-associative cache is shown in Figure 11. The same set of benchmarks are simulated with a 32-KB 8-way set-associative cache, and the same miss-rate curves generated for a 32-KB fully-associative cache are used. In this case, a 16 entry victim cache is added. In the figure, the model-based partitioning improves the miss-rate about 4% for short time quanta and up to 15% for mid-range time quanta. The figure demonstrates that the model-based partitioning mechanism works reasonably well for set-associative caches.

5. CONCLUSION

An analytical cache model to estimate overall miss-rate when multiple processes are sharing a cache has been presented. The model obtains the information about each process from its miss-rate curve, and combines it with parameters that define the cache configuration and schedule of processes. Interference among processes under the LRU replacement policy is quickly estimated for any cache size and any time quantum, and the estimated miss-rate is very accurate. A more important result is that the model provides not only the overall miss-rate but also a very good understanding of the effect of context switching. For example, the model clearly shows that the LRU replacement policy is problematic for mid-range time quanta because the policy replaces the blocks of least recently executed process that are more likely to be accessed in the near future.

The analytical model has been applied to the cache partitioning problem. A model-based partitioning method has been implemented and verified by simulations. Miss-rate curves are recorded off-line and partitioning is performed on-line according to the combination of processes that are executing. Even though we have used an off-line profiling method to obtain miss-rate curves, it should not be hard to approximate the miss-rate curve on-line using a miss-rate monitoring technique. Therefore, a fully on-line cache partitioning method can be developed based on the model.

Only the cache partitioning problem has been studied in this paper. However, as shown by the study of cache partitioning, our model can be applied to any cache optimization problem that is related to the problem of context switching. For example, it can be used to determine the best combination of processes that can be run on each processor of a multi-processor system. Also, the model is useful to identify areas in which further research in improving cache performance would be fruitful since it can easily provide the maximum improvement we can expect in the area.

6. ACKNOWLEDGMENTS

Funding for this work is provided in part by the Defense Advanced Research Projects Agency under the Air Force Research Lab contract F30602-99-2-0511, titled “Malleable Caches for Data-Intensive Computing”. Thanks also to E. Peserico, D. Chiou, and D. Chen for their comments on the cache model.

7. REFERENCES

[1] A. Agarwal, M. Horowitz, and J. Hennessy. An analytical cache model. *ACM Transactions on Computer Systems*, 7(2), May 1989.

[2] Compaq. Compaq AlphaStation family.

[3] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, John & Sons, Incorporated, Mar. 1991.

[4] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, 17(5), 1997.

[5] C. Freeburn. The hewlett packard PA-RISC 8500 processor. Technical report, Hewlett Packard Laboratories, Oct. 1998.

[6] A. González, M. Valero, N. Topham, and J. M. Parcerisa. Eliminating cache conflict misses through XOR-based placement functions. In *the 1997 international conference on Supercomputing*, 1997.

[7] J. L. Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. *IEEE Computer*, July 2000.

[8] N. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *the 17th Annual International Symposium on Computer Architecture*, 1990.

[9] D. B. Kirk. Process dependent static cache partitioning for real-time systems. In *Real-Time Systems Symposium*, 1988.

[10] H. Kwak, B. Lee, A. R. Hurson, S.-H. Yoon, and W.-J. Hahn. Effects of multithreading on cache performance. *IEEE Transactions on Computers*, 48(2), Feb. 1999.

[11] J. L. Lo, J. S. Emer, H. M. Levy, R. L. Stamm, D. M. Tullsen, and S. J. Eggers. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems*, 15, 1997.

[12] P. Magnusson and B. Werner. Efficient memory simulation in SimICS. In *28th Annual Simulation Symposium*, 1995.

[13] MIPS Technologies, Inc. *MIPS R10000 Microprocessor User's Manual*, 1996.

[14] J. C. Mogul and A. Borg. The effect of context switches on cache performance. In *the fourth international conference on Architectural support for programming languages and operating systems*, 1991.

[15] J. Muoz. *Data-Intensive Systems Benchmark Suite Analysis and Specification.* http://www.aaec.com/projectweb/dis, June 1999.

[16] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer system simulation: The SimOS approach. *IEEE Parallel & Distributed Technology*, 1995.

[17] M. S. Squillante and E. D. Lazowska. Using processor-cache affinity information in shared-momory multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 4(2), Feb. 1993.

[18] H. S. Stone, J. Turek, and J. L. Wolf. Optimal partitioning of cache memory. *IEEE Transactions on Computers*, 41(9), Sept. 1992.

[19] G. E. Suh and L. Rudolph. Set-associative cache models for time-shared systems. Technical Report CSG Memo 433, Massachusetts Institute of Technology, 2001.

[20] D. Thiébaut and H. S. Stone. Footprints in the cache. *ACM Transactions on Computer Systems*, 5(4), Nov. 1987.

[21] D. Thiébaut, H. S. Stone, and J. L. Wolf. Improving disk cache hit-ratios through cache partitioning. *IEEE Transactions on Computers*, 41(6), June 1992.

[22] N. Topham and A. González. Randomized cache placement for eleminating conflicts. *IEEE Transactions on Computers*, 48(2), Feb. 1999.

[23] J. Torrellas, A. Tucker, and A. Gupta. Benefits of cache-affinity scheduling in shared-memory multiprocessors: A summary. In *the 1993 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, 1993.

[24] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *22nd Annual International Symposium on Computer Architecture*, 1995.

[25] R. A. Uhlig and T. N. Mudge. Trace-driven memory simulation: A survey. *ACM Computing Surveys*, 29(2), June 1997.

[26] M. Zagha, B. Larson, S. Turner, and M. Itzkowitz. Performance analysis using the MIPS R1000. In *Supercomputing'96*, 1996.