Stochastic analysis of cache thrashing

Analyse stochastique des fautes de cache

Xavier Gréhant

2010D008

février 2010

Département Informatique et Réseaux
Groupe S3 : Systèmes, Logiciels, Services

Telecom ParisTech

Stochastic Analysis of Cache Thrashing

Analyse Stochastique des Fautes de Cache

Xavier Gréhant

with Institut Telecom, Telecom ParisTech, CNRS UMR 5141, LTCI, and CERN openlab, and supported by HP Labs.

Abstract

Cache misses are central to processor performance. The analysis of memory access patterns is widely used for performance prediction under monotasking. The monotasking hypothesis gives a lower bound of the cache miss ratio. With multitasking, which is the rule on mainstream operating systems, additional misses are caused by context switches. We present a higher bound of the cache misses overhead from context switches, based on a stochastic analysis of how the cache warms up, i.e. fills up with useful data. We observe that the higher bound is as close as the lower bound to the actual cache miss ratio. The resulting segment covers the exact cache miss ratio in a finite time quantum, potentially representative of the whole execution.

Les fautes de cache sont déterminantes pour la performance des processeurs. L'analyse des patterns d'accès mémoire est largement utilisée pour la prédiction de performance en l'absence de partage du cache entre processus. Cette hypothèse permet d'obtenir une limite inférieure du ratio de fautes de cache. En présence de concurrence entre processus, ce qui est le cas dans les système d'exploitation habituels, un surplus de fautes de cache est du aux changements de contexte. Nous proposons une limite supérieure du nombre de fautes additionnelles dues aux changements de contexte, à partir d'une analyse stochastique de la façon dont le cache se "réchauffe", c'est-à-dire se remplit de données utiles au processus en cours. Nous observons que la limite supérieure est aussi proche que la limite inférieure de la valeur réelle. Le segment obtenu comprend la valeur exacte du ratio de fautes de cache dans un quantum fini, potentiellement représentatif de la totalité de l'execution.

Contents

Chapter 1

Introduction

Processor caches are normally shared by multiple processes. A cache stores a limited amount of memory for fast access. Processes scheduled on successive time quanta affect each other's performance by erasing each other's cached items. This produces extra *cache misses*, i.e. failures to fetch items from cache. The adverse effect on performance is called *cache thrashing*. It is in some cases overwhelming, although reputedly difficult to quantify.

Performance prediction is particularly relevant to application, compiler and platform developers, and to task schedulers. Developers evaluate cache performance with expensive simulations. Online task schedulers require a faster alternative to map tasks to heterogeneous platforms. Cache misses are to a great extent responsible for performance variability between heterogeneous platforms. However, former prediction algorithms make simplifying assumptions that question their applicability to real cases, as with multiple concurrent processes.

This paper presents a stochastic approach to cache misses prediction in presence of multiple concurrent processes time-sharing a fully associative LRU[1] *cache.*

The method gives a higher bound of the *cache miss rate*, i.e. the ratio of cache misses per memory access.

Section 2 positions this work with current research. Section 3 introduces useful concepts: *age, rank, span, propagation*. Section 4 shows their relationships with the expected number of cache misses in a time quantum. Section 5 presents their calculation based on the stack distance distribution of the process. Section 6 summarizes the algorithm and its complexity. Section 7 measures the cost of its hypotheses on a set of benchmarks.

[1] A cache with Least Recently Used replacement policy

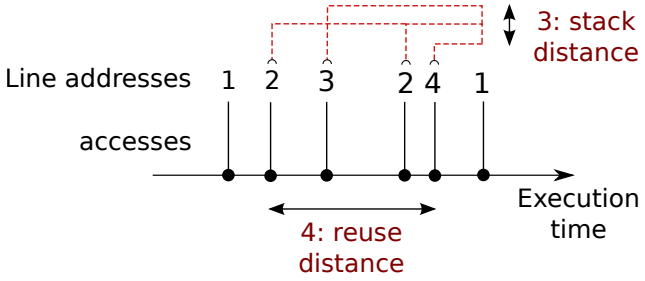

Figure 2.1: Stack and reuse distance

Chapter 2

Position of this work

This section positions the present work in its context. Section 2.1 introduces the metric that forms the basis of the analysis. Section 2.2 presents related work on cache simulation and performance prediction. Section 2.3 identifies our contributions.

2.1 Stack distance

In order to prepare for the complexity of the analysis, this section goes into more details than paper **??** on the definition of the metrics.

The method relies on the following cache design principles. Given that contiguous memory items are often accessed in a row, multiple contiguous items are fetched from memory at a time, in a *line* or *block*. Moreover, given that the same memory accesses often occur repeatedly, every line is stored on cache (is *cached*) when accessed, in order to be available on subsequent accesses. Several strategies may apply to determine the position of a new line on cache.

Associativity is a design choice. With a n-associative cache, a memory line can only be stored on n different positions based on its memory address. A fully-associative cache of size c lines is a c-associative cache. Fully-associative

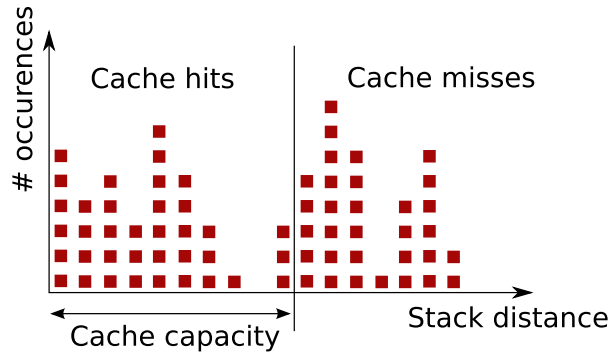

Figure 2.2: Misses under cache monopoly

caches make it possible to place a new line anywhere. When the cache is full, the replacement policy determines which line is selected for eviction.

On fully associative caches with Least Recently Used (LRU) replacement policy, every new line overwrites (*evicts*) the least recently used line, considered one of the least likely to be reused in the future.

The method is based on the estimation of its *stack distance* probability distribution. This distribution describes the temporal locality of memory accesses. An observation can be made for every memory access.

Definition 1 (Stack distance, adapted from [BD01])**.** *The* ***stack distance*** *of a memory access is the number of different lines accessed since the last reference to the requested line, present and initial references excluded. Stack distance is defined on* $\mathbb{N} \cup \{+\infty\}$*. A stack distance of* $+\infty$ *occurs on the first request of any line.*

A few authors call stack distance *circular sequence* [CGKS05]. Others call it *reuse distance* [SSkP$^+$07]. However, for most authors, reuse distance is incremented for every distinct access to the same line. On figure 2.1, the stack distance of the last access to line 1 is 3 while its reuse distance is 4 because line 2 was accessed twice between the two accesses to line 1.

Definition 2 (Reuse distance, adapted from [BH04])**.** *the* ***reuse distance*** *of a memory access is the number of memory references since the same line was previously accessed, present and initial references excluded. Reuse distance is defined on* $\mathbb{N} \cup \{+\infty\}$*. A reuse distance of* $+\infty$ *occurs on the first request of any line.*

Stack distance allows to predict the number of cache misses for a process that runs forever with a dedicated LRU, fully associative cache. An access is a miss if and only if its stack distance is greater than the cache capacity. Figure 2.2 represents a stack distance histogram. For each stack distance, the histogram stores the number of corresponding accesses. The miss rate appears on the histogram as the ratio of the number of misses by the number of hits.

There is indecision in current literature whether stack distance is a number of different *lines* accessed between two successive accesses to the same *line*, or a number of different *items* accessed between two successive accesses to the same *item*. Counting lines is useful for prediction but cannot be done by looking at the process only. Counting items can be done once for all for a process but is not directly useful for performance prediction. We deliberately choose the first alternative and assume that stack distance is known. Strictly speaking, it depends on line size and on spatial distribution of data on memory. To obtain stack distance from independent analysis of a process and a platform is a work in progress.

2.2 Previous work

Agarwal, Hennessy and Horowitz show in [AHH89] that the impact of multi-programming on cache misses is at least substantial and can be predominant. In [LGS+08], Liu et al. confirm this observation on recent benchmarks and processors.

A LRU, fully associative cache is comparable to a FIFO (first in first out) queue with perturbations. In the absence of memory reads, the first line written (first in the stack) is the first evicted (first out). Perturbations are caused by memory accesses that reorder line rankings. After every access, the accessed line is ranked first and the others are shifted down in the rankings. In this perspective, the present work is related to queuing theory as described in [All90]. However, queueing theory is concerned with the time spent in the queue - the queueing delay, while the present work is concerned with the probability that a line is in the cache when the program needs to access it.

Strictly speaking, a *stack* is a LIFO (last in first out) queue. Caches resemble stacks because of the programming artifact that the most recently used lines (last in) generally have a high probability to be re-used soon (first "out"). This observation motivates the use of the LRU replacement policy, that dictates that the least recently used line in cache is the first line evicted, i.e. overwritten by a new entry.

Gecsei, Slutz, and Traiger introduced stack distances in 1970 [GST70]. In [GAFN94], Grimsrud et al. show that stack distances describes a typical memory access pattern better than other models still in use today. In 1999, Brehob and Enbody define the *stack distance of a reference as the depth in the stack from which it was fetched*; and use it for cache misses prediction [BE99]. They exhibit prediction errors of a few percents when running a single process at a time.

In [SDR01], Suh, Devadas and Rudolph calculate the average miss probability on a finite time quantum, from the given miss probability as a known, convex function of the number of cached lines. More recently, in [LGS+08], Liu et al. use a Markov model to simulate the effect of cache thrashing. A three-states Markov chain describes the last step to obtain a given distribution of data in cache. Recursion on transition probabilities yields a simple cache simulation

algorithm. To the best of our knowledge, these two works are representative for the few attempts to predict or simulate context switch misses. Among these references, other works in cache performance prediction are restricted to the monotasking case, where the cache is dedicated to a single process.

Stack distance histograms are often used as tasks *signatures*, i.e. to store memory access patterns. They can be computed at compile time, by analyzing loops, and under some assumptions on the regularity of the references within a loop, as is done in [CP03]. An alternative is to monitor memory accesses at runtime on a short time period, and suppose that the extracted stack distance histogram is representative of the full run [MMC04]. In [GJ08], we fit *stack distance* to known probability distributions to reduce the size of the task signature to a few parameters and the complexity of the miss prediction to a constant complexity, with measured risks on accuracy.

Results are often obtained for LRU, fully associative caches and said to extrapolate to other types of cache. The impact of replacement policy is measured for example in [GS06]. The impact of cache associativity is detailed in [HS89]. In [LZ09], Liu et al. generalize to *associative* caches the conditions on stack distance under which an access yields a cache miss.

2.3 Assumptions and contributions

The present work builds on the assumption that stack distance observations are *independent, identically distributed*. To the best of our knowledge, this assumption is always done for non cycle accurate cache performance prediction and simulation. It underlies for example the use of stack distance histograms. However, it removes some information contained in a stack distance trace. For example, it hides the fact that because of program loops, successive accesses often exhibit regular stack distance patterns. An alternative would be to consider stack distance as a stochastic process, at the expense of a probably more complex model.

In addition, the stack distance distribution Σ of the process of interest is supposedly known, and representative of an identified phase of the execution. Its cumulative distribution function $F(k) = \mathbb{P}(\Sigma \geq k)$ is given at any point. The computational complexity of the proposed algorithm is the number of calls to F.

The method gives the first higher bound of the cache miss ratio in presence of context switches. It complements prediction methods such as [LZ09] that do not account for context switches and, therefore, present a lower bound of the cache miss ratio in multitasking. It is consistent with other analysis of cache thrashing ([SDR01] and [LGS$^+$08]). However, by contrast with [SDR01], it does not require prior knowledge or assumptions on the miss probability function, and by contrast with [LGS$^+$08], it results in a prediction algorithm faster than a simulator.

We prove that the computation can be done with the initial approximation of the singular values of a bidiagonal matrix, followed by a computation with

cubic complexity of the cache size. We also indicate, but we fail to prove in the general case, an exact method, quadratic of the cache size.

A detailed terminology is developed to elaborate from the observation of stochastic processes. It cements a theoretical frame for the development of more accurate or faster algorithms.

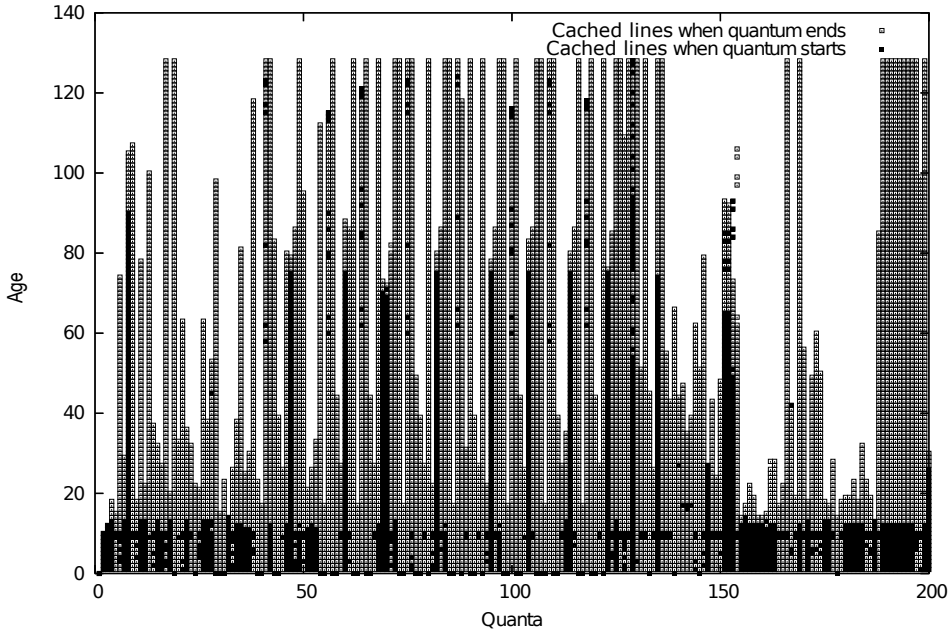

Figure 3.1: Cached lines with their age at the beginning and the end of successive time quanta.

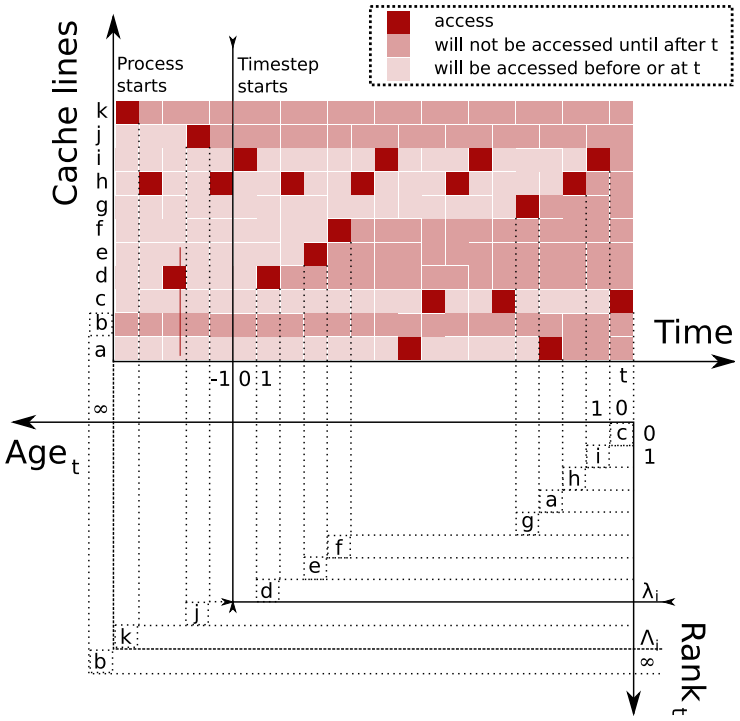

Figure 3.2: Relationship between time, age, rank and span.

Chapter 3

Metrics

This section introduces metrics to model the state of a cache as seen by a process in competition with other time-shared processes. Memory accesses increment a measure of *time* relative to the process. At a given time, *age* and *rank* characterize cache lines, and *span*, characterizes the process by its number of cached lines. *Propagation* measures consistency between two time quanta.

3.1 Age

This section defines *time* and a line's *age* to describe usage recency, and connects age to reuse distance: the reuse distance of a hit is the age of the requested line.

Definition 3 (Time)**.** *The **time** of a memory access is the number of previous memory accesses by the process since the beginning of the time quantum. Time is defined on* $\mathbb{Z} \cup \{+\infty, -\infty\}$*.* $+\infty$ *is the time of a non existent future access, and* $-\infty$ *is the time of a non existent past access.*

Definition 4 (Duration)**.** *The **duration** of a period is the number of memory accesses of the process on this period. Duration is defined on* $\mathbb{N}$*.*

Definition 5 (Age)**.** *The **age** of a line is the time minus the time of its last request (fig. 3.2). Age is defined on* $\mathbb{N} \cup \{+\infty\}$*.*

Theorem 1. *An access* i *of reuse distance* $r \in \mathbb{N}$ *is a hit on the line of age* r *at* $i-1$ *if it exists and a miss otherwise.*

Proof. If i is the time of the access of reuse distance r, the previous access to the same line is at time $i-r-1$. Therefore the age of the requested line at $i-1$ is r. □

3.2 Rank

This section defines *rank*, that indexes lines according to their age, and connects rank to stack distance: the stack distance of a hit is the rank of the requested line.

Definition 6 (Rank)**.** *A line's **rank** is the number of younger lines (fig. 3.2).*

Definition 7 (Cache capacity)**.** *The cache **capacity** is the number of different lines that the process is allowed to have in cache at a given time.*

Lemma 1 (Necessary condition for eviction)**.** *If* C *is the cache capacity, a line of rank* r *at* i *can be evicted at* $i+1$ *only if*

$$r = C - 1$$

Proof. LRU replacement policy ensures that r is the maximum rank. Full associativity ensures that the allocated cache is fully occupied before eviction. Therefore, the maximum rank is $C-1$. □

Theorem 2. *An access* i *of stack distance* s *hits the line of rank* s *at* $i-1$ *if it exists and is a miss otherwise.*

Proof. Suppose access i is a hit. Let s be its stack distance, l the line hit, and x its rank l at access $i-1$. There were s different lines $l_1, \ldots, l_s$ accessed since last access n_0 to the same line l, x of which have not been evicted at $i-1$. Therefore, $x \leq s$.

Ad absurdum, suppose $x < s$. In this case $\exists k \in [1, s]$ such that l_k was accessed at time i_2 and evicted a time i_3 with $i_0 < i_2 < i_3 < i$. Therefore, if $r_{k,3}$ is the rank of l_k at time $i_3 - 1$, $r_{k,3} < x$. In addition, since i_3 is an eviction and i is not, from lemma 1, $r_{k,3} > x$. By contradiction, $x = s$. Therefore, if i is a hit, the requested line has rank s.

By contraposition, if there is no line of rank s, n is a miss. □

3.3 Span

This section introduces the *span*, the number of lines that have been accessed by the process, and its evolution in the quantum.

Definition 8 (Active line)**.** *A cached line of age a is **active** at time i if $a \leq i$, i.e. if it has been accessed in the time quantum.*

Definition 9 (Active span)**.** *The **active span** is the number of active lines (fig. 3.2).*

Lemma 2 (Active span and rank)**.** *If r_{max} is the maximum rank of active lines and λ is the active span,*

$$\lambda = r_{max} + 1$$

Proof. Let l be the active line of maximum rank r_{max}. All $\lambda - 1$ other active lines are younger, therefore $r_{max} \geq \lambda - 1$. All r_{max} younger lines are also active, therefore $\lambda \geq r_{max} + 1$. □

Definition 10 (Maximum span)**.** *With C the cache capacity and Σ the stack distance distribution of the process (only the finite values), the **maximum span** c is defined by*

$$c = \min(C, \max \Sigma + 1)$$

Theorem 3 (Convergence of active span)**.** *In a time quantum, the active span is monotonic increasing and converges into the maximum span almost surely.*

Proof. Let c be the maximum span, and $\forall i \in \mathbb{N}$, λ_i the active span at time i. The sequence $(\lambda_i)_{i \in \mathbb{N}} \in \mathbb{N}^{\mathbb{N}}$ gives the evolution of active span.

Let $i \in \mathbb{N}$.

- If access $i + 1$ requests a line active at i, $\lambda_{i+1} = \lambda_i$.
- If access $i + 1$ requests a line not active at i, $\lambda_{i+1} > \lambda_i$.

Therefore, (λ_i) is monotonous increasing.

We show that $\forall i \in \mathbb{N}$, $\lambda_i \leq \max \Sigma + 1$ by recursion on time. $\lambda_0 = 1 \leq \max \Sigma + 1$. We suppose that $\lambda_i \leq \max \Sigma + 1$ and examine every case for λ_{i+1}.

- If $i + 1$ requests a line active at i, $\lambda_{i+1} = \lambda_i$ and $\lambda_i \leq \max \Sigma + 1$ by hypothesis. Therefore, $\lambda_{i+1} \leq \max \Sigma + 1$.
- If $i + 1$ requests a line not active at i, let s be its stack distance at i.

- If $i+1$ is a hit, let l be the requested line and r its rank at i. From theorem 2, $s = r$. Since l is not active at i, l is older than all active lines. If r_{max} is the maximum rank of active lines at i, $r > r_{max}$. From lemma 2, $r_{max} = \lambda_i - 1$ and therefore $r \geq \lambda_i$. Since $r = s$ and $s \leq \max \Sigma$, $\lambda_i \leq \max \Sigma$ and therefore $\lambda_{i+1} \leq \max \Sigma + 1$.
- If $i+1$ is a miss, from theorem 2, there is no line of rank s at i. We show that $\lambda_i \leq s$ *ad absurdum*. Suppose that $\lambda_i > s$. From lemma 2, there is a line l of rank $r_l = \lambda_i - 1$. Therefore, $r_l \geq s$. Since there is no line of rank s, $r_l > s$. By definition 6, there are r_l lines younger than l, and the $(s+1)$th youngest is of rank s. By contradiction, $\lambda_i \leq s$ and therefore $\lambda_{i+1} \leq s + 1$.

We showed that $\forall i \in \mathbb{N},\ \lambda_i \leq \max \Sigma + 1$. In addition, $\forall i \in \mathbb{N},\ \lambda_i \leq C$ by definition 9. By the monotone convergence theorem, (λ_i) converges and $\lim_{k\to\infty} \lambda_k \in [0, c]$.

Let $\lambda \in [0, c-1]$ and $i \in \mathbb{N}$ such that $\lambda_i = \lambda$. By theorem 2, an access of stack distance c is a miss, and by lemma 1, it does not yield an eviction. Therefore,

$$\forall n \in \mathbb{N}, \mathbb{P}(\lambda_{i+n} = \lambda) \leq (1 - \mathbb{P}(\Sigma = c))^n$$

By taking the limit:

$$\mathbb{P}\left(\lim_{k\to\infty} \lambda_k = \lambda\right) = 0$$

It remains that:

$$\mathbb{P}\left(\lim_{k\to\infty} \lambda_k = c\right) = 1$$

□

Definition 11 (Time to fill)**.** *The* ***time to fill*** *is the time at which the active span reaches the maximum span. If λ_i is the active span at time i and Δ is the time to fill:*

$$\Delta = \min\{i \in [1, +\infty] | \lambda_i = c\}$$

The time to fill may be greater than the quantum duration. In this case the active span does not reach the maximum span during the quantum.

Figure 3.3 represents memory accesses in a single, isolated time quantum. The x coordinate enumerates accesses in chronological order. The y coordinate shows the stack distance of each access with a vertical bar and the active span with a horizontal line.

3.4 Propagation

This section introduces *propagation* to measure cache consistency between two time quanta.

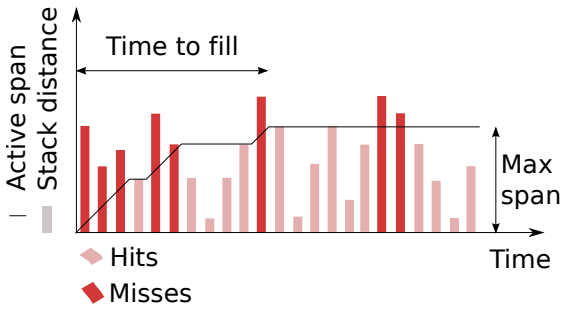

Figure 3.3: Memory accesses in a single, isolated time quantum.

Definition 12 (Propagation)**.** *The* ***propagation*** *is the proportion of cached lines that are not erased between two time quanta of the process of interest.*

The evaluation of propagation is not detailed in this paper.

Definition 13 (Propagated line)**.** *A cached line of age a is* ***propagated*** *at time i if $a > i$, i.e. if it was last accessed by the process of interest in a previous time quantum and not accessed ever since.*

On a common operating system, many services run together with user processes. Even with a single user process, the propagation between its time quanta is low, as shown by figure 3.4.

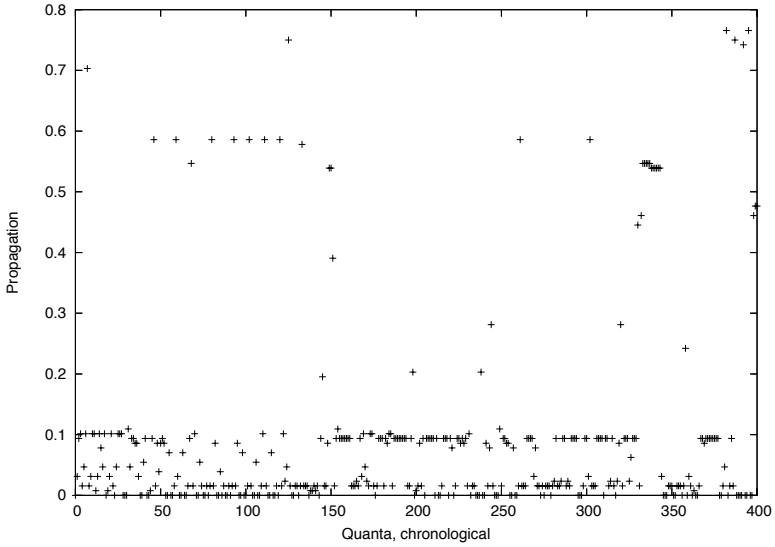

Figure 3.4: Propagation observed between successive quanta of a process running "alone" on Linux.

Chapter 4

Cache misses

This section shows how the probability of occurrence of cache misses in a time quantum is linked to the cumulative distribution function of stack distance and the metrics defined in section 3.

4.1 A typology of cache misses

Lines cached by the process were last accessed either in the current time quantum or in a previous time quantum. They are either active or propagated. The request of a line which is neither active nor propagated is a miss.

Definition 14. *A **blind access** is the request of a non active line.*

Definition 15. *A **blind hit** is the request of a propagated line.*

Lemma 3. *If M is the set of misses over a time period, B_A the set of blind accesses and B_H the set of blind hits,*

$$M = B_A \backslash B_H$$

with $B_A \backslash B_H = \{x \in B_A | x \notin B_H\}$.

Proof. A miss is the request of a line which is not in cache. Active lines are in cache, therefore a miss is a request of a non active line, i.e. $M \subset B_A$ (1). Propagated lines are in cache, therefore the request of a propagated line is not a miss, i.e. $M \cap B_H = \emptyset$ (2). (1) and (2) yield $M \subset B_A \backslash B_H$.

Conversely, a line which is neither active nor propagated is not in cache. Therefore, its request is a miss, i.e. $B_A \backslash B_H \subset M$. □

4.2 Blind accesses count

Theorem 4. *Let λ_i be the active span at i and s_{i+1} the stack distance of $i+1$. $i+1$ is a blind access if and only if*

$$s_{i+1} \geq \lambda_i$$

Proof. By lemma 2, all active lines at i have ranks in $[0, \lambda_i - 1]$. Then theorem 2 applies. □

Corollary 1. *If Σ is the stack distance distribution, i is an access and λ_i is the active span at i, the probability that $i+1$ is a blind access is*

$$\mathbb{P}(i+1 \in B_A) = \mathbb{P}(\Sigma \geq \lambda_i)$$

Corollary 2. *We write $[0,i]$ the $i+1$ first accesses of the time quantum, $[0,i] \cap B_A$ the corresponding blind accesses, λ_k the active span at k, Σ the stack distance of the process, c the maximum span, Δ the time to fill. The expected number of blind accesses on $[0,i]$ is:*

$$\begin{aligned} &E\left[|[0,i] \cap B_A|\right] \\ &= E[\lambda_i] + (i - E[\Delta | \Delta < i])\mathbb{P}(\Delta < i)\mathbb{P}(\Sigma \geq c) \end{aligned}$$

Proof. We use the indicator variable $\mathbf{1}_X$. $\mathbf{1}_X = 1$ in the event X and 0 otherwise. For all $k \in [0,i]$, let s_k be the stack distance at access k. From corollary 1:

$$\begin{aligned} E[|[0,i] \cap B_A|] &= E\left[1 + \sum_{k=1}^{i} \mathbf{1}_{s_k \geq \lambda_{k-1}}\right] \\ &= E\left[1 + \sum_{k=1}^{\min(i,\Delta)} \mathbf{1}_{s_k \geq \lambda_{k-1}}\right] \\ &\quad + E\left[\mathbf{1}_{i>\Delta} \sum_{k=\Delta+1}^{i} \mathbf{1}_{s_k \geq \lambda_{k-1}}\right] \end{aligned}$$

By theorem 3, $k < \Delta \Leftrightarrow \lambda_k < c$ and $k \geq \Delta \Leftrightarrow \lambda_k = c$.

$$\begin{aligned} &E\left[\mathbf{1}_{i>\Delta} \sum_{k=\Delta+1}^{i} \mathbf{1}_{s_k \geq \lambda_{k-1}}\right] \\ &= E\left[\mathbf{1}_{i>\Delta} \sum_{k=\Delta+1}^{i} \mathbf{1}_{s_k \geq c}\right] \\ &= E\left[\mathbf{1}_{i>\Delta} \sum_{k=\Delta+1}^{i} 1\right] \mathbb{P}(\Sigma \geq c) \\ &= E\left[\mathbf{1}_{i>\Delta}(i - \Delta + 1)\right] \mathbb{P}(\Sigma \geq c) \\ &= (i - E[\Delta | \Delta < i])\mathbb{P}(\Delta < i)\mathbb{P}(\Sigma \geq c) \end{aligned}$$

By recursion on i, we show that

$$1 + \sum_{k=1}^{\min(i,\Delta)} \mathbf{1}_{s_k \geq \lambda_{k-1}} = \lambda_i$$

If $i = 1$, $\lambda_0 = 1$. Suppose that the hypothesis is true for i, we examine case $i+1$. If $i+1 \leq \Delta$

$$\begin{aligned} 1 + \sum_{k=1}^{\min(i,\Delta)} \mathbf{1}_{s_k \geq \lambda_{k-1}} &= \lambda_i + \mathbf{1}_{s_k \geq \lambda_{k-1}} \\ &= \lambda_{i+1} \end{aligned}$$

If $i + 1 > \Delta$

$$\begin{aligned} 1 + \sum_{k=1}^{\min(i,\Delta)} \mathbf{1}_{s_k \geq \lambda_{k-1}} &= \lambda_i \\ &= \lambda_{i+1} \end{aligned}$$

□

4.3 Blind hits count

Theorem 5. *Let $i \in \mathbb{N}$. If $i+1$ is a request of line l with stack distance s, λ_i the active span at i and Λ_i the total span at i,*

$$\lambda_i \leq s < \Lambda_i \Leftrightarrow l \text{ is propagated and not active at } i$$

Proof. This implication is straightforward:

$$l \text{ is propagated and not active at } i \Rightarrow \lambda_i \leq s < \Lambda_i$$

We prove that:

$$\lambda_i \leq s < \Lambda_i \Rightarrow l \text{ is propagated and not active at } i$$

Since $s < \Lambda_i$, there is a cached line of rank s at i. From theorem 2, this line is l. Since its rank $s \geq \lambda_i$, l is not active. However, l is in cache, and therefore, l is propagated. □

Corollary 3. *Let $i \in \mathbb{N}$. If $i+1$ is an access of stack distance s, λ_i the active span at i, Λ_i the total span at i, the probability that $i+1$ is a blind hit is:*

$$\mathbb{P}(i+1 \in B_H) = \mathbb{P}(s < \Lambda_i)\mathbb{P}(s \geq \lambda_i)$$

And for $i = 0$, with ρ the propagation:

$$\mathbb{P}(0 \in B_H) = \rho\mathbb{P}(s < \Lambda_{-1})$$

Proof. The general case follows from theorem 5. We show the case $i = 0$. Access 0 is blind by definition. Supposing that $\rho = 1$, From theorem 2 it is a hit if and only if the line l of rank s at -1 is still in cache before the access. l is in cache at -1 with probability $\mathbb{P}(s < \Lambda_{-1})$ and l is still in cache before access 0 with probability $\rho\mathbb{P}(s < \Lambda_{-1})$. □

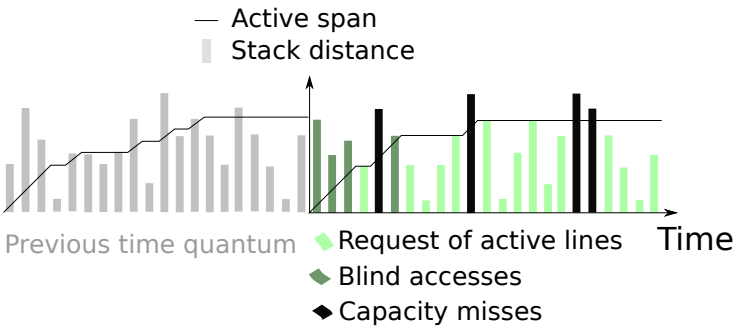

Figure 4.1: Accesses in successive time quanta

Corollary 4. *For $i \in \mathbb{N}$, the expected number of blind hits on $[0, i]$ is*

$$E\left[|[0,i] \cap B_H|\right] = \rho\mathbb{P}(s < \Lambda_{-1}) + \sum_{s=1}^{c}\sum_{k=0}^{i-1}\mathbb{P}(\Sigma = s)\mathbb{P}(\Lambda_k > s)\mathbb{P}(\lambda_k \leq s)$$

Proof. From corollary 3,

$$E\left[|[0,i] \cap B_H|\right] = \rho\mathbb{P}(s < \Lambda_{-1}) + \sum_{k=1}^{i}\mathbb{P}(\Sigma < \Lambda_{k-1})\mathbb{P}(\Sigma \geq \lambda_{k-1})$$

□

Figure 4.1 shows accesses in non-isolated time quanta. The x axis is time. On the y axis, vertical bars represent stack distances and the horizontal line represents active span. Positive x's are accesses in current time quantum, and negative x's (left panel) are accesses in previous time quantum of the same process. On current time quantum, accesses of stack distance greater than cache capacity (also known as **capacity misses** in [HP06]) are shown with dark bars. Blind accesses are shown with medium-light bars. Depending on propagation, some are hits and others are misses. Requests of active lines are shown with light bars. These are hits.

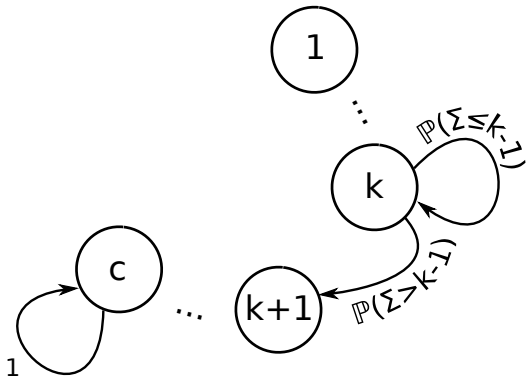

Figure 5.1: State diagram of active span indexed by time $(\lambda_i)_{i\in\mathbb{N}}$

Chapter 5

Stochastic analysis

The metrics defined and used in previous sections are related to stack distance. This section explains how. They are based on a Markov process. The stack distance distribution Σ appears in transition probabilities.

5.1 Active span

Theorem 6. *The active span indexed by time in a quantum, $(\lambda_i)_{i\in\mathbb{N}}$, is a Markov chain with the following transitions:*

- $\forall k \in [0, c-1]$, $\forall i \in \mathbb{N}$,

$$\mathbb{P}(\lambda_{i+1} = k | \lambda_i = k) = \mathbb{P}(\Sigma \leq k-1)$$
$$\mathbb{P}(\lambda_{i+1} = s+1 | \lambda_i = s) = \mathbb{P}(\Sigma > k-1)$$

- $\forall i \in \mathbb{N}$, $\mathbb{P}(\lambda_{i+1} = c | \lambda_i = c) = 1$

Proof. Let $i \in \mathbb{N}$. From theorem 4, $i+1$ is a blind access with probability $\mathbb{P}(\Sigma \leq \lambda_i - 1)$. While the active span is not maximal, blind accesses increment it (proof of theorem 3). Theorem 3 also states that when the active span is maximal, it remains constant. □

Figure 5.1 shows the state diagram of $(\lambda_i)_{i\in\mathbb{N}}$.

Let P_f be the transition matrix of $(\lambda_i)_{i\in\mathbb{N}}$. Diagonal elements are supposed non zero. However, they can be chosen as small as necessary.

We write $F_k = \mathbb{P}(\Sigma \leq k)$ and $\bar{F}_k = \mathbb{P}(\Sigma > k)$

$$P_f = \begin{pmatrix} F_0 & \bar{F}_0 & & \\ & \ddots & \ddots & \\ & & F_{c-2} & \bar{F}_{c-2} \\ & & & F_{c-1} \end{pmatrix} = \begin{pmatrix} T_f & \mathbf{T_f^0} \\ \mathbf{0} & 1 \end{pmatrix}$$

Since $c \geq \max \Sigma + 1$, $F_c = \mathbb{P}(\Sigma \leq c) = 1$. This defines the $1 \times (c-1)$ vector $\mathbf{T_f^0}$ and the $(c-1) \times (c-1)$ substochastic matrix T_f.

Since P_f is triangular, P_f is diagonalizable and its eigenvalues are its diagonal values. Therefore, there is a passage matrix A_{P_f} such that $A_{P_f}^{-1} P_f A_{P_f} = D_{P_f}$ with:

$$D_{P_f} = \begin{pmatrix} F_0 & & \\ & \ddots & \\ & & F_{c-1} \end{pmatrix}$$

Since A_{P_f} is invertible, $\forall n \in \mathbb{N}$, ${P_f}^n = A_{P_f} {D_{P_f}}^n {A_{P_f}}^{-1}$.

Let $\boldsymbol{\tau_f}$ be the initial distribution of $(\lambda_i)_{i\in\mathbb{N}}$, i.e. the list of probabilities of each state at access 0. Since $\mathbb{P}(\lambda_0 = 0) = 1$ and $\mathbb{P}(\lambda_0 > 0) = 0$, $\boldsymbol{\tau_f} = [1, 0, \ldots, 0]$.

Corollary 5. *The expected active span is written:*

$$\begin{aligned} E[\lambda_i] &= \sum_{k=1}^{c} k\mathbb{P}(\lambda_i = k) \\ &= \boldsymbol{\tau_f} {P_f}^i [1, \ldots, c]^t \end{aligned}$$

5.2 Time to fill

Lemma 4. *$(\lambda_i)_{i\in\mathbb{N}}$ is terminating and c is an absorbing state.*

Proof. This is a direct consequence of theorem 3. It can also be seen on P_f. $\forall i \in [0, c-1], 0 \leq F_i < 1$ and $F_c = 1$ therefore,

$$\exists x \in \mathbb{Q} | \lim_{n\to\infty} {P_f}^n = \begin{pmatrix} 0 & & & 0 \\ & \ddots & & \vdots \\ & & 0 & 0 \\ 0 & \ldots & 0 & x \end{pmatrix}$$

Since P_f is stochastic $[0, \ldots, 0, 1]P_f^n\mathbf{1} = 1$. By taking the limit, $x = 1$.

As a consequence,

$$\lim_{n\to\infty} {D_{P_f}}^n = \begin{pmatrix} 0 & & & 0 \\ & \ddots & & \vdots \\ & & 0 & 0 \\ 0 & \ldots & 0 & 1 \end{pmatrix} \text{ with exponential speed}$$

□

Theorem 7. *The time to fill, Δ, is a discrete phase type distribution.*

Proof. Δ takes values in $\mathbb{N}$. $(\lambda_i)_{i\in\mathbb{N}}$ is a terminating Markov chain with finitely many states. The maximum span, c, is its absorbing state. Δ is the first passage time to c, therefore, Δ, is a discrete phase type distribution. □

Corollary 6. *The f_Δ be the probability mass function and F_Δ the cumulative distribution function of Δ, i.e. $\forall k \in \mathbb{N}$,*

$$f_\Delta(k) = \mathbb{P}(\Delta = k)$$
$$F_\Delta(k) = \mathbb{P}(\Delta \leq k)$$

As a characterization of a discrete phase type distribution:

$$f_\Delta(k) = \boldsymbol{\tau_f} {T_f}^{k-1} \mathbf{T_f^0}$$
$$F_\Delta(k) = 1 - \boldsymbol{\tau_f} {T_f}^{k} \mathbf{1}$$

Corollary 7. *If Δ is the time to fill, $\forall k \in \mathbb{N}$, P_f the transition matrix of active span, T_f its sub-stochastic matrix, and $T_f = A_{T_f} D_{T_f} A_{T_f}^{-1}$ the diagonalization of T_f, The expected value of Δ under the condition that $\Delta \leq i$, is:*

$$E[\Delta|\Delta \leq i] = \boldsymbol{\tau_f} A_{T_f} \left(\sum_{k=0}^{i} k {D_{T_f}}^{k-1} \right) A_{T_f}^{-1} \mathbf{T_f^0}$$

Proof. Let $f_\Delta(k) = \mathbb{P}(\Delta = k)$.

$$\begin{aligned} E[\Delta|\Delta \leq i] &= \sum_{k=0}^{i} k f_\Delta(k) \\ &= \sum_{k=0}^{i} k \boldsymbol{\tau_f} {T_f}^{k-1} \mathbf{T_f^0} \\ &= \boldsymbol{\tau_f} \left(\sum_{k=0}^{i} k {T_f}^{k-1} \right) \mathbf{T_f^0} \\ &= \boldsymbol{\tau_f} A_{T_f} \left(\sum_{k=0}^{i} k {D_{T_f}}^{k-1} \right) A_{T_f}^{-1} \mathbf{T_f^0} \end{aligned}$$

□

Chapter 6

Algorithm and complexity

The following expression gives the number of cache misses in a time quantum based on knowledge of the stack distance distribution, the number of accesses in the quantum, and the assumption that no line was propagated from previous quantum.

Theorem 8. *Let n be the duration of a time quantum, λ_n the active span at n, Δ the time to fill, and Σ the stack distance distribution. We write P_f the transition matrix of the active span, T_f its sub-stochastic matrix, $\boldsymbol{\tau_f} = [1, 0, \ldots, 0]$ of size c, $\mathbf{T_f^0} = [0, \ldots, 0, \bar{F}(c-2)]^t$ of size $c-1$, $\mathbf{1} = [1, \ldots, 1]^t$ of size $c-1$, and $\forall k \in \mathbb{N}$, $F(k) = \mathbb{P}(\Sigma \leq c)$ and $\bar{F}(k) = \mathbb{P}(\Sigma > k)$.*

In the absence of propagation, the number of cache misses in the quantum is:

$$\begin{aligned} M &= E[\lambda_n] + (n - E[\Delta|\Delta < n])\mathbb{P}(\Delta < n)\mathbb{P}(\Sigma \geq c) \\ &= \boldsymbol{\tau_f} {P_f}^n [1, \ldots, c]^t \\ &+ \left(n - \sum_{k=0}^{n} k \boldsymbol{\tau_f} {T_f}^{k-1} \mathbf{T_f^0} \right) (1 - \boldsymbol{\tau_f} {T_f}^n \mathbf{1})(1 - F(c-1)) \end{aligned}$$

In addition, the computational complexity is the complexity of the implicit-shifted QR algorithm [DK90] on P_f and T_f, and the rest of the computation is done in $O(c^3)$.

Proof. From corollary 2, M is the number of blind accesses in the time quantum. Since there is no propagation, there is no blind hit by definition. Therefore, from lemma 3, M is the number of cache misses in the time quantum.

From corollary 5,

$$E[\lambda_n] = \boldsymbol{\tau_f} {P_f}^n [1, \ldots, c]^t$$

From corollary 6,

$$\mathbb{P}(\Delta < n) = 1 - \boldsymbol{\tau_f} {T_f}^n \mathbf{1}$$

From corollary 7,

$$E[\Delta|\Delta < n] = \boldsymbol{\tau_f} A_{T_f} \left(\sum_{k=0}^{n} k {D_{T_f}}^{k-1} \right) A_{T_f}^{-1} \mathbf{T_f^0}$$

P_f and T_f are bidiagonal. The implicit-shifted QR algorithm can be used to approximate their singular values.

Once P_f and T_f are diagonalized, P_f^n and T_f^n are computed in $O(c^3)$. Therefore, $E[\lambda_n]$, $\mathbb{P}(\Delta < n)$ and $E[\Delta|\Delta < n]$ are computed in $O(c^3)$. □

The QR algorithm approximates the singular value decomposition of a bidiagonal matrix. It is iterative, and new iterations are done until enough precision is reached. In fact, we suspect that use of the QR algorithm is not necessary and that the overall complexity is in fact in $O(c^2)$. This is due to the fact that the transition matrix of active span P_f and its sub-stochastic matrix T_f are bidiagonal stochastic. The following presents this faster alternative.

We consider a bidiagonal right-stochastic matrix M_n of size n and we write its diagonal elements $F_0 \dots F_{n-1}$ and $\forall k \in [0, n-1]$, $\bar{F}_k = 1 - F_k$.

$$M_n = \begin{pmatrix} F_0 & \bar{F}_0 & & \\ & \ddots & \ddots & \\ & & F_{n-2} & \bar{F}_{n-2} \\ & & & F_{n-1} \end{pmatrix}$$

If $\forall k \in [0 \dots n-1]$, $F_k = \mathbb{P}(\Sigma \leq k)$, then $P_f = M_c$ and $T_f = M_{c-1}$. We write the diagonalization of M_n:

$$M_n = A_n D_n A_n^{-1}$$

with D_n diagonal and A_n invertible.

A pattern systematically appears in A_n and A_n^{-1}, for every n. If $A_{n\,i,j}$ is the element at row i and column j of A_n, and ${A_n}^{-1}{}_{i,j}$ is the element at row i and column j of A_n^{-1}, $\forall (i,j) \in [0, n-1]^2$, we observe that:

$$A_{n\,i,j} = \prod_{k=i}^{j-1} \frac{\bar{F}_k}{F_j - F_k} \delta_{i \leq j}$$

$${A_n}^{-1}{}_{i,j} = \frac{\prod_{k=i}^{j-1} \bar{F}_k}{\prod_{k=i+1}^{j} F_i - F_k} \delta_{i \leq j}$$

δ is the Kronecker symbol, i.e. $\delta_X = 1$ if X is true, 0 otherwise.

Therefore, we express $\boldsymbol{\tau}_n A_n D {A_n}^{-1}$ where D is diagonal with diagonal elements d_s, $s \in [0, n-1]$, and $\boldsymbol{\tau_n} = [1, 0, ..., 0]$ of size n.

Let $v = \boldsymbol{\tau_n} A_n D {A_n}^{-1}$ and v_j the element at index j in line v.

$$\begin{aligned}
v_j &= \sum_{s=0}^{j} \left(\prod_{k=0}^{s-1} \frac{\bar{F}_k}{F_s - F_k} \right) \delta_{0 \leq s} d_s \frac{\prod_{k=s}^{j-1} \bar{F}_k}{\prod_{k=s+1}^{j} F_s - F_k} \delta_{s \leq j} \\
&= \sum_{s=0}^{j} d_s \frac{\prod_{k=0}^{j-1} \bar{F}_k}{\prod_{\substack{k=0 \\ k \neq s}}^{j} F_s - F_k} \\
&= \left(\prod_{k=0}^{j-1} \bar{F}_k \right) \sum_{s=0}^{j} d_s \prod_{\substack{k=0 \\ k \neq s}}^{j} (F_s - F_k)^{-1}
\end{aligned}$$

Since all members of the expression of the number of cache misses in theorem 8 are on this form, the complexity of the algorithm is in $O(c^2)$ for every size c of bidiagonal stochastic matrix for which the pattern is observed, which we suspect is all $\mathbb{N}$. However, we have not been able to prove it.

Chapter 7

Experimental validation

The ratio of cache misses on a single time quantum is representative of the whole run if all quanta have same number of memory accesses, i.e. duration. Figure 7.1 shows the duration of successive time quanta for an execution of the Unix command *ls*. We observe a wide variability on a logarithmic scale. However, the effect of the hypothesis that the time quantum duration is constant is minor compared to the effect of hypotheses on propagation.

The following measurements are taken with Simics, a full system micro-architecture simulator described in [MCE+02]. The simulated platform is a x86 processor with a LRU, fully associative cache, running a Linux operating system. Measurements are taken for different benchmarks running "alone", i.e. only in competition with the default background services of the operating system.

Simics returns the actual number of cache misses in the whole run. In addition, it allows to monitor memory accesses, and it provides access to cache contents at any step in the execution. We use this information to monitor the propagation at every time quantum, and we simulate the same memory accesses on synthetic time quanta with modified propagation and duration. The goal is to replicate the effect of the simplifying hypotheses made by the algorithm of this paper, in comparison with other hypotheses made by existing or hypothetical algorithms.

The instrumentation of Simics and the simulation of synthetic time quanta required some programming. The code is written in Python using test-driven development. It is available under Artistic License 2 on a public repository[1].

Figures 7.2 to 7.6 show the relative number of cache misses under different assumptions, with regards to the actual number of cache misses.

1. **Warm cache** is the common hypothesis of prediction algorithms that do not account for the effect of context switches. A cache miss occurs for every stack distance greater than the cache capacity.

2. **Full propagation** means that all lines are propagated between two quanta

[1] code.google.com/p/mtc-project

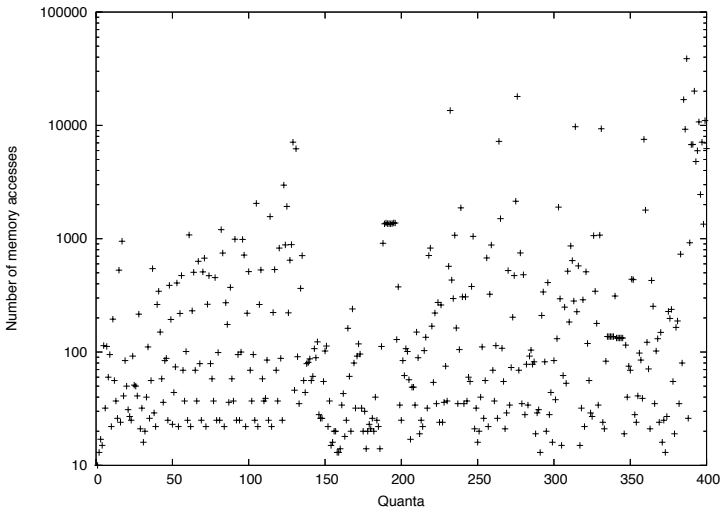

Figure 7.1: Number of memory accesses (= duration) of successive time quanta.

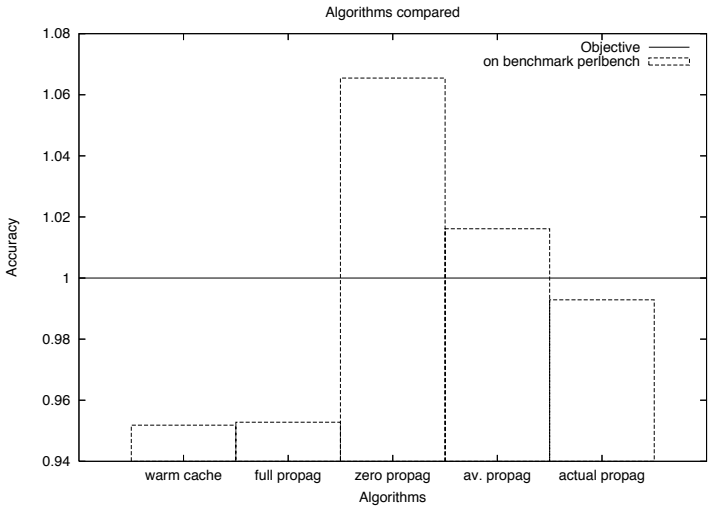

Figure 7.2: perlbench

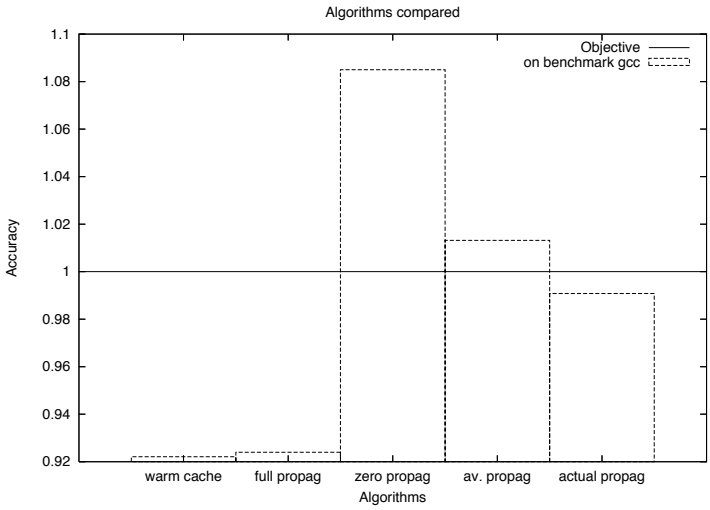

Figure 7.3: gcc

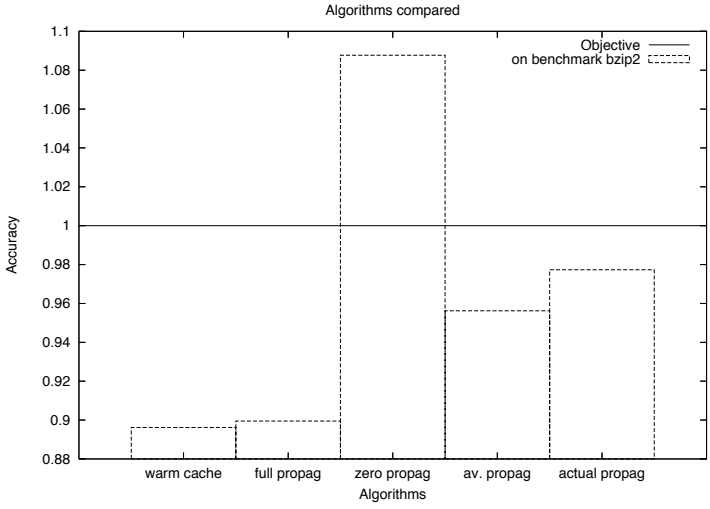

Figure 7.4: bzip2

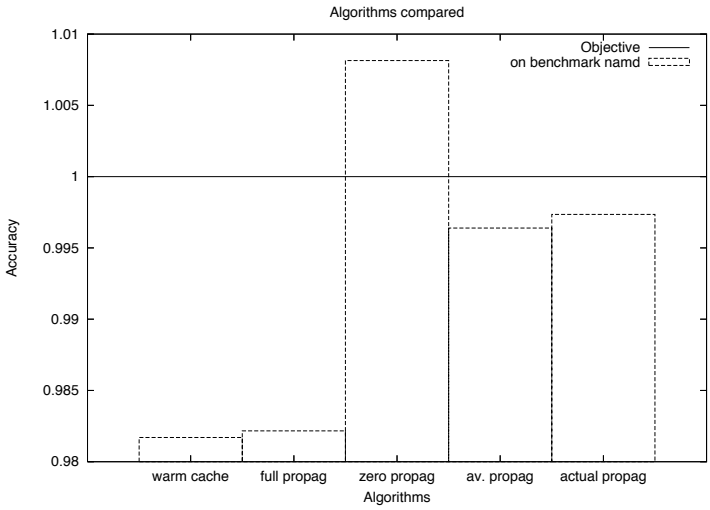

Figure 7.5: namd

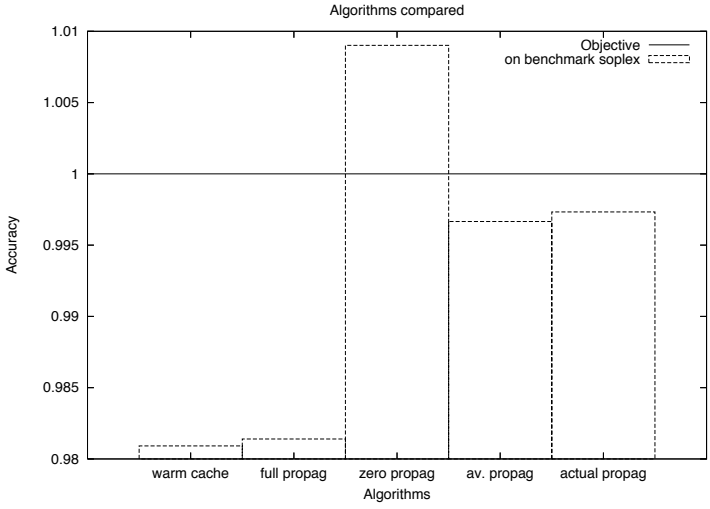

Figure 7.6: soplex

of the same process. By contrast with *warm cache*, a cache miss is counted at every first access to a memory line.

3. **Actual propagation** only differs from the observation by the duration of the time quanta. It is set constant, and equal to the average actual duration.

4. **Average propagation** also sets quantum duration as the average of the actual value. In addition, it differs from the observation by the propagation, which is set constant and equal to the average actual propagation.

5. **Zero propagation** is the result of the calculation proposed by theorem 8. It supposes that the time quantum duration is constant and the propagation is zero, i.e. it counts a cache miss for every blind access.

Figures 7.2 to 7.6 confirm that the expression of theorem 8 gives a higher bound of the cache miss ratio, and that the monotasking assumption gives a lower bound. In addition, we observe the distance to the actual value is qualitatively the same for the higher and the lower bounds. The lower bound, however, remains much faster to compute, as shown in [GJ08].

Chapter 8

Conclusion

Since the advent of multitasking, context switches have been known to generate cache misses. However, prior to this work, there has been no mean other than simulation to predict their impact from the observation of memory access patterns. The apparent complexity of the analysis justifies this blank with regards to the coverage of the monotasking case, and sustains the opinion that caches are unpredictable.

Still, the process by which cache warms up at the beginning of a quantum exhibits convenient properties for analysis. Prior algorithms consider that an access of stack distance greater than the cache capacity is a miss. This is true but it only gives a lower bound of the cache miss ratio. In fact, an access of stack distance greater than the number of lines cached since the beginning of the quantum is a possible miss, unless the requested line was written in a previous quantum and not erased by other processes inbetween. Considering that no line is kept between quanta gives a higher bound of the cache miss ratio. We observe in exeriments that higher and lower bounds are equally distant to the actual value.

Bibliography

[AHH89] A. Agarwal, J. Hennessy, and M. Horowitz. An analytical cache model. *ACM Trans. Comput. Syst.*, 7(2):184–215, 1989.

[All90] Arnold O. Allen. *Probability, statistics, and queueing theory with computer science applications*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[BD01] K. Beyls and E.H. D'Hollander. Reuse distance as a metric for cache behavior. In T. Gonzalez, editor, *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 617–622, Anaheim, California, USA, 8 2001. IASTED.

[BE99] Mark Brehob and Richard Enbody. An analytical model of locality and caching. Technical report, Michigan State University, Dept of Computer Science and Engineering, 1999.

[BH04] E. Berg and E. Hagersten. Statcache: a probabilistic approach to efficient and accurate data locality analysis. In *ISPASS '04: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 20–27, Washington, DC, USA, 2004. IEEE Computer Society.

[CGKS05] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture. HPCA-11.*, pages 340–351, Feb. 2005.

[CP03] Calin Cascaval and David A. Padua. Estimating cache misses and locality using stack distances. In *ICS*, pages 150–159, 2003.

[DK90] James Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Stat. Comput.*, 11(5):873–912, 1990.

[GAFN94] K. Grimsrud, J. Archibald, R. Frost, and B. Nelson. On the accuracy of memory reference models. In *Proceedings of the 7th international conference on Computer performance evaluation : mod-*

elling techniques and tools, pages 369–388, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[GJ08] Xavier Grehant and Sverre Jarp. Lightweight cache analysis for cache-aware scheduling on heterogeneous clusters. In *Proceedings of PDPTA'08, WorldComp'08*, Las Vegas, 7 2008.

[GS06] Fei Guo and Yan Solihin. An analytical model for cache replacement policy performance. In *SIGMETRICS '06/Performance '06: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 228–239, New York, NY, USA, 2006. ACM.

[GST70] J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Syst. J.*, 9(2):78–117, 1970.

[HP06] John Hennessy and David Patterson. *Computer Architecture - A Quantitative Approach.* Morgan Kaufmann, 1990,1996,2003,2006.

[HS89] M. D. Hill and A. J. Smith. Evaluating associativity in cpu caches. *IEEE Trans. Comput.*, 38(12):1612–1630, 1989.

[LGS+08] Fang Liu, Fei Guo, Yan Solihin, Seongbeom Kim, and Abdulaziz Eker. Characterizing and modeling the behavior of context switch misses. In *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 91–101, New York, NY, USA, 2008. ACM.

[LZ09] Yu Liu and Wei Zhang. Exploiting stack distance to estimate worst-case data cache performance. In *SAC*, pages 1979–1983, 2009.

[MCE+02] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, Feb 2002.

[MMC04] Gabriel Marin and John Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. *SIGMETRICS Perform. Eval. Rev.*, 32(1):2–13, 2004.

[SDR01] G. Edward Suh, Srinivas Devadas, and Larry Rudolph. Analytical cache models with applications to cache partitioning. In *ICS '01: Proceedings of the 15th international conference on Supercomputing*, pages 1–12, New York, NY, USA, 2001. ACM.

[SSkP+07] Xudong Shi, Feiqi Su, Jih kwon Peir, Ye Xia, and Zhen Yang. Cmp cache performance projection: accessibility vs. capacity. *SIGARCH Comput. Archit. News*, 35(1):13–20, 2007.

Dépôt légal : 2010 – 1er trimestre
Imprimé à Télécom ParisTech – Paris
ISSN 0751-1345 ENST D (Paris) (France 1983-9999)

Télécom ParisTech

Institut TELECOM - membre de ParisTech

46, rue Barrault - 75634 Paris Cedex 13 - Tél. + 33 (0)1 45 81 77 77 - www.telecom-paristech.frfr

Département INFRES