An Analitical cache model

Christine Fricker, Philippe Robert

Rapports de Recherche

N°1496

Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

AN ANALYTICAL CACHE MODEL

Christine FRICKER
Philippe ROBERT

Septembre 1991

An analytical cache model

Christine Fricker, Philippe Robert

INRIA, domaine de Voluceau
B.P. 105, 78153 Le Chesnay Cédex, France
Christine.Fricker@inria.fr, Philippe.Robert@inria.fr

Abstract.

In this paper we study a mathematical model of the sequence of memory references generated by a program. This model is an extension of the continuous model proposed in [9]. In the context of an architecture with cache-memories, we derive analytical formulas for the miss ratio and the execution time per instruction. The parameters of these formulas are the cache size, the block size, the associativity and the locality of the program. The respective influences of these parameters on the performances of the architecture are analyzed and compared with the results based on trace-driven simulations. Based on this model, a random generator of traces is proposed to study more complex memory hierarchies and some transient program behaviors.

Étude d'un modèle analytique de mémoire cache

Christine Fricker, Philippe Robert

INRIA, domaine de Voluceau
B.P. 105, 78153 Le Chesnay Cédex, France
Christine.Fricker@inria.fr, Philippe.Robert@inria.fr

Résumé.

Nous étudions un modèle stochastique de la suite des références mémoires engendrées par un programme. Ce travail fait suite à [9] où un modèle continu avait été considéré. Dans le cadre d'une architecture avec mémoire cache, nous donnons les expressions analytiques du taux de défaut. Les paramètres de ces expressions sont la taille du cache, la taille des lignes de cache, l'associativité ainsi que la localité du programme. L'influence des divers paramètres sur les performances des architectures avec cache est étudiée et les conclusions comparées avec les résultats obtenus dans la littérature par des simulations à partir de traces de programmes. Nous proposons un générateur aléatoire de références-mémoire ssocié à ce modèle pour étudier des hiérarchies mémoire plus complexes ainsi que certains comportements transitoires de programmes.

1 Introduction

The last years have seen the growing importance of memory hierarchies in the design of computer architectures; as the CPU cycle time decreases, the access to the memory becomes more and more the main bottleneck for the performance of architectures. Despite of its crucial importance, the way of how the memory (and more generally some level of the memory hierarchy) is accessed, is not properly understood (if one remembers that simulations are still the main tool in this area). For example, if one can consider that the behavior of caches whose sizes are around 32KB is quite well understood (see [13]), it is not so clear for two level cache hierarchies (see [14]) or for larger caches (see [5]). For multi-processors architectures of cluster type (see [1], [18] and [12]), the complexity becomes quite unmanageable (problems of coherence, bursty accesses, bus contention,...). Our goal in this paper is propose a simple model of memory accesses which has the main properties that matter in a memory hierarchy. This model allows to derive analytical formulas for the performance of some architectures and to construct a random generator of memory accesses.

Up to now, the main tool to estimate the performance of memory hierarchies has been trace driven simulations. A set of programs is chosen and these programs are traced, i.e. as the program executes, the addresses and the nature of the references requested in main memory are stored. Then these traces can be sent into a simulator of some memory hierarchy. The well known drawbacks of this method are:

1. **Getting the traces**. Tracing a program is quite long, since the execution must be stopped after each instruction in order to get the characteristics of this one. It is commonly admitted (though quite empirically) that for cache sizes of the order of 32KB, some millions of traces are sufficient to simulate the behavior of a cache. An extension of this method has been proposed in [5] where a device is proposed to trace billions of references to simulate caches whose sizes are around some Megabytes. It seems quite unlikely that this method can be used in a standard way.

2. **The choice of the programs to trace**. Generally the set of programs being traced is shared between very regular programs like the ones

used in numerical analysis (Lawrence Livermore Loops, Linpacks...) and a set of more "random" programs likes pieces of the kernel, Unix binaries... The underlying idea is that a mixture of these programs will reasonably represent the "standard" behavior of an architecture. Despite this step cannot be avoided, it is quite difficult to understand the choices it entails on the architecture and the sensitivity of the set of programs on these choices. Because of this mixture of programs, even if a simple phenomenon determines some range of values for some parameter, it may not be apparent in the final result.

3. **The duration of the simulations**. Trace driven simulations must be repeated for *every* configuration of the architecture. If the memory hierarchy has several levels of caches (as most of the projects of architectures now), a complete simulation is almost out of reach.

The performance of an architecture can also be estimated through analytical models. They can give first estimates about some key parameters of the memory hierarchy. Extended surveys on this subject can be found in [17] and [2]. The main problem in this case is to agree on a mathematical model for the references generated by a program. The model must be analytically tractable and it must represent in a "reasonable" way the behavior of a program in a memory hierarchy. The independence reference model (IRM) used in the context of paging algorithms can be a possible candidate to represent the traces of a program. Most of the analytical studies on caches use more or less this model mainly because of its mathematical tractability. It assumes that the number of the successive lines of cache requested by the CPU form a sequence of independent random variables. If the IRM model gives a quite accurate picture of the classification of cache replacement algorithms (see [15]), it does not have the main properties that motivate the use of a cache in an architecture: Every line is referenced independently of the others so that almost no spatial or temporal locality of the references will be represented in this model. Moreover it assumes a fixed length of line and thus does not allow the analysis of the optimal line size.

An analytical description of the behavior of a cache has been proposed in [2] where the authors proposed a small set of parameters to evaluate the performance of a cache memory. These parameters include the notion of intrinsic interference (collisions within the cache of the blocks of the same programs).

The essential idea is that the behavior of a cache is due to a small number of phenomena such as regular collisions, start up effects, non-stationary behavior... Our study goes in this direction. Most of the mathematical models do not care sufficiently to the underlying phenomena that cause repeated misses in a cache. We propose a more precise description of a program which extends in some way the model of [2] and allows a more systematic study of the parameters that influence the behavior of a cache (through analytical formulas and the random generator proposed with the model). This model can also be used to study the performances of the different reordering strategies of scientific code such as blocked algorithms (for example, see [11] where the special case of memory accesses generated by a matrix multiplication is analyzed).

Our purpose in this paper is to give a simple stochastic model of the sequence of memory addresses requested by a program. Though simple, this model has the two major characteristics that motivate the use of cache : the temporal locality (some references are frequently accessed) and the spatial locality (some references are contiguous). The important point here, is that the randomness does not break the locality properties of the accesses as it was the case in the previous stochastic models. With this model, we derive the corresponding analytical expressions for the miss ratio. It is proved that the miss ratio can be expanded as a finite sum $\sum_{i=1}^{n} \frac{a_i}{C^i}$ where C is the cache size and a_i is a coefficient depending on the block size, the associativity and the locality of the program. Thus this model is mathematically tractable. The phenomena concerning the optimal block size, the effects of associativity,... are compared with the corresponding ones obtained by trace driven simulations (see [17]). The empirical formulas for the miss ratio proposed in the literature are discussed.

The analytical expressions of the cache performance are not the only consequences of the choice of a mathematical model for the memory references. Attached to this model, a generator of random traces is proposed. It is much quicker than usual trace driven simulations (though that these ones cannot be avoided, at least in the final steps of the design procedure) and it can give some insight in some areas where the analytical study is more difficult, like multiprogramming environment, start up behaviors, more complex hierarchies of caches...

Let us finish by recalling some definitions and elementary properties of cache memories. The main memory is divided into blocks of l words. The

block (cache line) is the basic unit of transfer between the main memory and the cache. When the address of a word is requested in the main memory, the block containing it is fetched and consequently, all the neighboring words in this block are also transferred.

An a-associative cache of size C is a cache divided into C/a sets, each of them having the same number a of blocks. If the blocks are b bits wide, when an address is requested by the CPU, its value shifted from b, modulo C/a gives the number of the set where it should be. The address is then compared with the addresses of the lines within this set to determine if it is already in the cache (hit) or absent (miss). In the case where there is a miss, the replacement policy determines the block to be erased by the block fetched in the main memory. The triple (C, a, l) defines the basic parameters of a cache organization. When $a = 1$, the cache is called direct-mapped. In a direct-mapped cache, an address has only one possible location.

2 Motivation of the model

Let us begin with the simple loop of scalar product (Kernel-3 of Lawrence Livermore)

```
DO  3  I= 1,N
       Q=Q+X[I]*Y[I]
    3  CONTINUE.
```

Every time I is incremented, the following parts of the main memory are accessed :

1. **The text** : The addresses of all the program instructions like LOAD, STORE,... These addresses are constantly referenced at every iteration during of the loop.

2. **The data** : In this case the arrays X and Y. The addresses fetched are incremented by one unit (the size of an integer or of a double) after each loop. The successive addresses fetched in this region are contiguous.

3. **The stack** : The state of the current process. From our point of view, it will be related to the text part.

A slice of the sequence of addresses fetched during two iterations of the above loop:

iteration n° 1	iteration n° 2	
4194952	4194952	
2147474444	2147474444	
2147474440	2147474440	
2147474440	2147474440	
2147474440	2147474440	
2147474440	2147474440	
2147474468	2147474472	⟵ `X[I]`
2147476468	2147476472	⟵ `Y[I]`
2147474444	2147474444	
2147474444	2147474444	
2147474436	2147474436	
2147474436	2147474436	
2147474440	2147474440	
2147474440	2147474440	

The area referenced in the main memory by the program can be split into elementary regions $(I_j)_j$ where I_j is some interval (i.e. a set of contiguous addresses) of the main memory. In the case of our loop, these intervals are reduced to one word, except for `X` and `Y` which are respectively $[ADDX, ADDX + 4 \times N]$ and $[ADDY, ADDY + 4 \times N]$, with $ADDX = 2147474468$ and $ADDY = 2147476468$. At every iteration, one address is referenced in each of these intervals.

In the same way, to define our model, we will assume that the main memory can be decomposed into K elementary regions $I_p = [B_p, B_p + L_p]$, $p = 1, \ldots, K$. If w is the size of a word, we define $A_p(k) = B_p + (kw \text{ modulo } [\frac{L_p}{w}]w)$ for $k \geq 0$ and $p \in \{1, \ldots, K\}$. Then for any k, $A_p(k) \in I_p$, A_p will be called an elementary access and $(A_p(k))_{k \geq 0}$ will be the sequence of the successive addresses requested by this access. Thus each elementary access requires successively all the words of its elementary region and then restarts at the beginning of this region and so on. As in our previous loop where $w = 4$, our model of program will be an interleaving of these K elementary accesses. The more straightforward way is to interleave them cyclically, that is the

sequence of addresses requested by the program is:

$$A_1(1), A_2(0), \ldots, A_K(0), A_1(w), \ldots, A_K(w), \ldots, A_K((n-1)w), A_1(nw), \ldots, A_K(nw), A_1((n+1)w), \ldots$$

However, to avoid problems with (intractable) periodic schemes in the analytical study, we consider the slightly different interleaving:

For $1 \leq p \leq K$ and $0 \leq i \leq [\frac{L_p}{w}]$, put the mark (p, i) on all the points $(iw + nL_p)_{n \geq 0}$ of the positive half line. We have then a non decreasing sequence of points on the real line (if there is an equality $iw + nL_p = i'w + n'L_{p'}$, then we consider that $iw + nL_p$ is before $i' + n'L_{p'}$ if $p < p'$). Now we can define the sequence of addresses of our model of program. If the n-th point has the mark (q, j) then the n-th address requested by the program will be $B_q + jw$. In the case (not considered here) where the L_p are multiple of w, it is easily seen that we obtain our cyclic interleaving. If we look only at the addresses in the region I_p and discard all the others, then we get the sequence of addresses $(A_p(k))_{k \geq 0}$ as expected.

The elementary region of the main memory accessed by A_p is $[B_p, B_p + L_p]$. The starting point B_p is some random variable. Its exact distribution is not important as we will see, provided the cache sizes analyzed are small compared to the size of the main memory. The second variable L_p will be the locality of A_p. If L_p is quite small (some bytes), then A_p will be a sequence of repeated accessed to a small number of addresses (temporal locality property). Otherwise, if L_p is quite large, A_p will be typically the sequence of accesses to a vector or a matrix (spatial locality property) and L_p is the size of the array (vector or matrix). We will assume that the L_p $(1 \leq p \leq K)$ are random variables with the common distribution H which will be our locality parameter. For example, if we look at our simple loop kernel-3 , there are 5 elementary regions and H would be the following distribution: with probability $\frac{3}{5}$, L_p is one word large and with probability $\frac{2}{5}$, L_p is N words large.

A random generator

The associated random generator is easy to implement. The following function **RandTrac**(T) returns K*T addresses with locality H:

RandTrac(T)

```
Initiate K random variables L[1],...,L[K] with distribution H
while index < T
```

```
k = 1
while k < K
writeoutput( B[p] + (index * w) modulo L[p] )
k = k + 1
endwhile
index = index + 1
endwhile
```

In the following, we will study analytically the stationary behavior of the cache, i.e. look at the miss ratio when $T \to +\infty$ in **RandTrac**. An analytical study of start-up period and multiprogramming effects seems much harder. Nevertheless, their effects can be analyzed (through successive calls to **RandTrac**) quite easily with this random generator and a cache simulator[1], and much more quickly than with the usual procedure using trace-driven simulations. The function **RandTrac** can also be modified to consider a more refined description of the program. Usually, an elementary access A_p will be either a READ, WRITE or an instruction fetch during the whole program. For example, in the loop kernel-3, all the accesses to `X` and `Y` are READ. For that, one just has to modify the function in the following way:
`writeoutput( tag[p], B[p] + (index * w) modulo L[p] )`
where `tag[p]` is either R, W or I.

Validity of the model

In this paper, we focus on the main phenomena that cause misses in a cache and on the mathematical modeling of locality properties of programs. We will pay little attention to the predictive aspects of the model, i.e. for a given program, to find the appropriate parameters such that, for any memory hierarchy the difference between miss ratios calculated with the model and real miss ratios is small. Nevertheless it is not difficult to remark that, because of our choice, most of the programs used in numerical analysis fit in that model (Linpacks, most of the Lawrence Livermore loops,...). In [11], the authors study cache performance to optimize blocked algorithms. The program analyzed in this paper is a matrix multiplication. From the point of view of memory accesses, it can be represented in our model, as a program

[1] For the simulations performed with this generator, we used M. D. Hill's cache simulator Dinero III.

with two elementary regions of size B and B^2 respectively, where B is the blocking factor (restricted to this special case, our analysis is close to what is done in [11]). In [2] the authors propose to measure some parameters of the program to derive the miss ratio of a given cache architecture. Though the authors do not consider an analytical description of a program, it is nevertheless quite interesting to see the analogue of their parameters in our model. Let us denote by $P(I)$ the region of the cache that the elementary region I will occupy (projection of I in the cache). According to the terminology of [2], the set $\bigcup_{i \neq j} P(I_i) \cap P(I_j)$ is the potential collision set. The feature included in our model is the coupling between the frequencies of collisions and the collision set itself; the sizes of the intervals and their places in the cache will determine the frequencies as we will see. In [2], the frequencies and the potential collision set are set independently. In our model the locality is included by the interleaving of elementary accesses. In [2] it is done in a comparable way as in the Partial Markov Reference Model (see [4]): If a line is required then, with some probability this line will be again requested with some probability and this for a while (run length), otherwise a new line is drawn at random.

3 The characterization of a program by its working set

Definition 1 *The working set of a program at time t is the set of addresses referenced by the program up to time t.*

We will denote by $w(t)$ the cardinality of the working set at time t. The function $t \rightarrow w(t)$ is non decreasing. Its rate of growth is one of the usual quantitative characteristics of the behavior of a program (if not the only one with the miss ratio). For the IRM model, it is related in various ways to the miss ratio (see [7]).

Our purpose in this section is to compare the IRM model and our model from this point of view. For that, we will take a simple example, namely the repeated accesses to the successive components of a vector of size N. For the IRM reference model, the probability of fetching the i^{th} component of this vector is $\frac{1}{N}$. Straightforward calculations show that the expected value

of $w(t)$ for the IRM model is

$$N(1-(1-\frac{1}{N})^t),$$

which is always less than $\min\{t, N\}$, the real cardinality of the working set for this example.

It is a constant feature of the IRM model that it often underestimates the size of the working set. Our example might look simple. Nevertheless it is quite typical of what the IRM model will never represent correctly: A steady growth of $w(t)$ during some periods of the execution of a program. In our opinion, this is the reason for which the spatial locality of programs is not accurately represented by the IRM model.

4 An analytical expression of the miss ratio

In this section, we denote by w, l and C the respective sizes of a word, a line and the cache. The cache is represented by the interval $[0, C]$. We will say the word kw (resp. the line kl) for a word (resp. a line) whose address in the direct mapped cache is kw (resp. kl). Therefore the cache is the union of the words $kw,\ \ 0 \leq k < C/w$ as well as the union of the lines $kl,\ \ 0 \leq k < C/l$.

An address of the main memory shifted from the number of bits of a block gives the number of the line of this address. If the cache is a-associative, this number of line modulo C/a will give the number of the set where it should be located if it is in the cache. For simplicity, until Proposition 5, we assume that our cache is direct mapped.

In our model, the t-th word that the elementary access j will require, should be located in the cache at $X_j(tw)$,

$$X_j(tw) = (U_j + (tw \text{ modulo } L_j)) \text{ modulo } C,$$

where $U_j = B_j$ modulo C. We will assume that the beginnings of the elementary regions U_j are independent and uniformly distributed on the discrete set of words $\{kw,\ 0 \leq k \leq \frac{C}{w} - 1\}$. This is a reasonable assumption as long as the cache size is not too large compared to the size of the main memory (see [2])

The elementary interval I_j is mapped onto $P(I_j) = \{kw, \ldots, (K-1)w\}$ in the cache, with $k \leq K$ and the access will require successively the $K-k$ words $kw, \ldots, (K-1)w$. If, for some line ml in this region $\{kw, \ldots, (K-1)w\}$, there is a miss when access A_j reaches the words which are mapped in line ml, then the whole line has to be fetched from the main memory. This line will have to be in the cache at each of the $\frac{l}{w}$ accesses of the words of this line i.e. $ml, ..., (m+1)l - w$ in order to avoid misses.

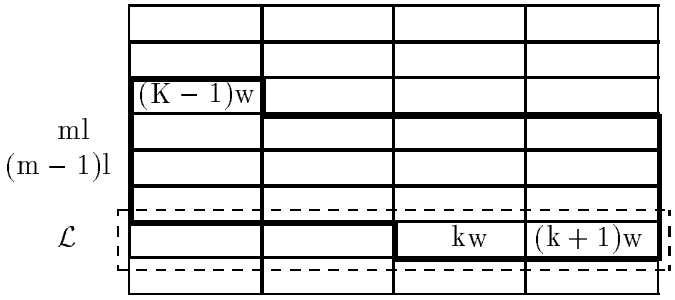


Figure 1: *Location of an elementary region in the cache*

Let us look at the instants when the elementary access A_j requires some fixed line $\mathcal{L}$ in the cache. In the case where line $\mathcal{L}$ does not intersect the region $\{kw, \ldots, (K-1)w\}$, this elementary access never requires it. However, if the line is required, the accesses to $\mathcal{L}$ are bursty: We denote by t the first time the first word of $\mathcal{L}$ is requested. The instants $t+w,\ t+2w, ..., t+(q-1)w$ are also instants when the access requires line $\mathcal{L}$. Here q is the number of words in $\mathcal{L} \bigcap \{kw, \ldots, (K-1)w\}$. The integer q is thus the number of accesses at line $\mathcal{L}$ per burst. After these consecutive accesses, there is a gap until time $t + L_j$ when this line is again referenced by the access A_j.

Definition 2 *For a fixed line $\mathcal{L}$ and a given elementary access X, we will say that an access of type 2 occurs if X requires a word of $\mathcal{L}$ and the previous access of X was already in $\mathcal{L}$. All the other accesses will be of type 1.*

In our example, the accesses to the word kw are of type 1 and those to word $(k+1)w$ of type 2 (see figure 1).

The direct mapped cache has the advantage of simplicity for the search of an address. There is only one line to check in the cache to know whether the line has been flushed by another access or not. It has nevertheless the

following drawback that, if two different accesses require alternatively the same line of the cache, then each word accessed within the line will cause a miss. For example, take $w = 1,\ l = 4$, two elementary memory regions a and b and the following sequence of accesses for a direct-mapped cache:

time	0	1	2	3	4	5	6	7	8	9	10	11	12	13
access	a	b	a	b	a	b	a	b	a	b	a	b	a	b
cache address	4	3	5	4	6	5	7	6	8	7	9	8	4	3
M() or H(it)	M	M	H	M	M	M	M	M	M	H	H	M	M	H

At time $t = 5, 7$, the line 4 of 4 words is accessed by b and by a at time $t = 4, 6$ and at each of these instants, the owner of the line alternates between a and b entailing a miss at each word accessed. If the cache is associative and the set size is two lines, two different lines can be present in the same set at the same time, so that the above misses disappear. The same phenomenon can occur in a two way set associative cache, however it will require that three accesses access alternatively the same set, which is more unlikely.

Notice that the misses that disappear are misses on accesses which follow an access in the same line (accesses of type 2) and that the misses at time $t = 0, 3$ which are the misses at the entrance in the line 4 (accesses of type 1) do not disappear for this reason. As for the accesses, the misses are of two types : the misses of type 1, the misses which eventually occur at the entrance in a line (at time $t = 0, 3$ in our example) and the misses of type 2, the other misses, which eventually occur when the previous word accessed by a given interval in the same line, also called internal misses in the line. From this point of view, the main difference between direct-mapped and associative caches is that the internal misses disappear if the number of elementary accesses in the same set at the same time is less or equal to the set associativity.

Remarks

a) In the particular case where the length of the elementary accesses is less than $\frac{C}{a} - l$, i.e. that every elementary access require at most one line of each set in an associative cache, at most a accesses can be present in the same set. Thus no miss occur in this set as long as there are less than a accesses in the set. Hence in this case, the gain due to associativity is positive, increasing with associativity.

b) The following simple example shows that associativity can, in some cases, decrease the performances of the cache. Take a 2-associative cache with 4 lines of 4 words of size 1. The replacement policy is LRU.

time	0	1	2	3	4	5	6	7	8
access	a	b	c	a	b	c	a	b	c
main memory address	0	16	8	0	16	8	0	16	8
direct mapped	M	M	M	M	M	H	M	M	H
two-associative	M	M	M	M	M	M	M	M	M

The idea is that if more than a elementary intervals are mapped in the same set, misses occur in the a-associative cache, at each entrance in a line (to simplify, let us assume that the elementary regions L_j have the same length) In a direct-mapped cache, the elementary accesses associated are split in the different lines of the set, causing perhaps no misses, for example if one is requiring a line and all the others are requiring one of the other lines.

Despite this example is somewhat artificial, it shows however that one cannot expect a relation between direct mapped and associative valid for any kind of program. We will see that in our model, the miss ratio for direct mapped is always greater than for a two-way-associative.

We will assume that the L_j's have a density function h. The j-th access requires a line $\mathcal{L}$ at instants consisting in a sequence of bursts. The number of accesses per burst conditioned on the fact that the j-th access requires $\mathcal{L}$ is q_j. The random variable (L_j, q_j) has a density function denoted g.

Definition 3 *The miss ratio at time t due to accesses of type $i = 1, 2$ is the number of misses of type i up to time t divided by the total number of memory references up to time t. The stationary miss ratio is the limit of the miss ratio at time t when t tends to infinity and is denoted by D_i.*

We first begin by the case of a direct mapped cache, we assume that the word is the unit of length. We assume that $h(x) = 0$ for $x \geq C - l$, i.e. $L_j + l \leq C$ for any j. We will denote $a \wedge b = \min\{a, b\}$.

Proposition 4 *The stationary miss rate D of a program of K elementary accesses in a direct mapped cache of size C with line size l can be decomposed into $D_1 + D_2$ with*

$$D_1 = \sum_q \int \frac{1}{q} \left(1 - \left(1 - \sum_r \int \frac{l+y-1}{C} (1 \wedge \frac{x-q+r}{y})\, g(y,r)dy \right)^{K-1} \right) g(x,q)dx \quad (1)$$

where D_1 is the miss rate for the accesses of the first words in a line of the cache, and

$$D_2 = \left(\sum_q \int (1 - \frac{1}{q})\, g(x,q)dx \right) \left(1 - \left(1 - \sum_q \int \frac{l+y-1}{C} (1 \wedge \frac{q}{y})\, g(y,q)dy \right)^{K-1} \right) \quad (2)$$

where D_2 is the miss rate for the accesses which follow an access in the same line, $h(x)$ is the density of the size of an elementary region, and $g(x,q)$ is the probability density of (L_1, q_1) on $[0, C] \times I\!N$ defined by

$$\sum_q \int F(x,q)g(x,q)dx =$$

$$\sum_{j=1}^{l-1} \int_{j-1}^{j} h(x)\, (\frac{l+1-j}{l+j-1}\, F(x,j) + \sum_{i=1}^{j-1} \frac{2}{l+j-1}\, F(x,i))\, dx$$

$$+ \sum_{j=l}^{C} \int_{j-1}^{j} h(x)\, (\sum_{i=1}^{l-1} \frac{2}{l+j-1} F(x,i) + \frac{j+1-l}{l+j-1}\, F(x,l))\, dx.$$

Proof:
See the Appendix. ■

Remarks

a) Usually the density of the length of the elementary accesses $h(x)$ will be concentrated around a small number of values (16B, 64B, 256B,...). If we consider this number of addresses N as the parameter, the complexity of formulas (1) and (2) is respectively N^2 and N. Hence numerical results are fairly easy to obtain (see Section 5). In the IRM model, for a program with N different addresses, the miss ratio associated has a complexity of at least of the order $N^{\frac{C}{l}}$ and is thus quite difficult to derive in practice (see [3]). In section 5, using these formulas, we will study the behavior of a cache architecture with the locality of the program $h(x)$ as a parameter.

b) When the line size is small compared to the cache size, the continuous analogue of the above formulas can be found in [9].

In the case of an associative cache, for simplicity we assume that the replacement policy is LRU and that $h(x) = 0$ for $x \geq \frac{C}{a} - l$ i.e. $L_j + l \leq \frac{C}{a}$ for any j.

Proposition 5 *The stationary miss ratio D of a program of $n > a$ elementary accesses in a a associative cache of size C with lines of size l (and words of size 1) can be decomposed in $D_1 + D_2$ where D_1 is the miss ratio due to the accesses to the first word in a line of the cache given by*

$$D_1 = \sum_{1 \leq q \leq l} \int g(x,q) \tfrac{1}{q} \times$$
$$\left(1 - \sum_{i=1}^{a} c_i \binom{a-i}{n-1} \left(1 - \sum_r \int a \tfrac{(l+y-1)}{C} (1 \wedge \tfrac{x-q+r}{y}) g(y,r) dy\right)^{n-1-a+i}\right) dx \quad (3)$$

and D_2 is the miss ratio due to the accesses which follow an access in the same line given by

$$D_2 = \left(\sum_q \int (1 - \tfrac{1}{q})\, g(x,q) dx\right) \times$$
$$\left(1 - \sum_{i=1}^{a} c_i \binom{a-i}{n-1} \left(1 - \sum_q \int \tfrac{(l+x-1)}{C} (1 \wedge \tfrac{q}{x})\, g(x,q) dx\right)^{n-1-a+i}\right) \quad (4)$$

where g is the density function of (L_1, q_1) defined by

$$\sum_q \int F(x,q) g(x,q) dx =$$

$$\sum_{j=1}^{l-1} \int_{j-1}^{j} h(x) \left(\frac{l+1-j}{l+j-1} F(x,j) + \sum_{i=1}^{j-1} \frac{2}{l+j-1} F(x,i)\right) dx$$

$$+ \sum_{j=l}^{\frac{C}{a}} \int_{j-1}^{j} h(x) \left(\sum_{i=1}^{l-1} \frac{2}{l+j-1} F(x,i) + \frac{j+1-l}{l+j-1} F(x,l)\right) dx$$

and the c_i $(1 \leq i \leq a)$, which depend also on n and a, are given by the recursion formula

$$c_1 = 1,\ c_l = 1 - \sum_{i=1}^{l-1} c_i \binom{l-i}{n-a+l-1} \qquad (2 \leq l \leq a)\ . \tag{5}$$

Proof:
See the Appendix. ■

Remarks

a) Proposition 4 can be seen as a special case of Proposition 5 when a is equal to 1 (the coefficients $(c_i)_{i \geq 1}$ do not play any role in the direct mapped case).

b) The assumption on the density h can be removed, the method to derive an expression for the miss ratio does not change, but then, the instants when the j-th elementary access requires line $\mathcal{L}$ induce a periodic structure with bursts which is more complicated.

5 Results

In this section we study the influence of the main architectural parameters on the miss ratio: line size, cache size and associativity. We will compare our results on dependence on cache size, on optimal line size, on changes due to associativity to the previous ones, obtained in the literature by trace driven simulations. Here, we have an additional parameter for the program: the locality. This last parameter was not introduced in the previous studies where the traces corresponds to benchmarks which are various mixtures of programs assumed to be representative of all the programs used.

In the applications of Propositions 4 and 5 we took three examples for h, corresponding to the programs 1, 2 and 3, with an increasing locality, and the number of elementary accesses K equal to 16 or 32. Below the size entry refers to the length (in words) of the elementary regions of the program:

program 1	size	1	2	8	64	
	proportion	5/16	5/16	2/8	1/8	
program 2	size	1	2	3	16	310
	proportion	0.16	0.16	0.16	0.4	0.1

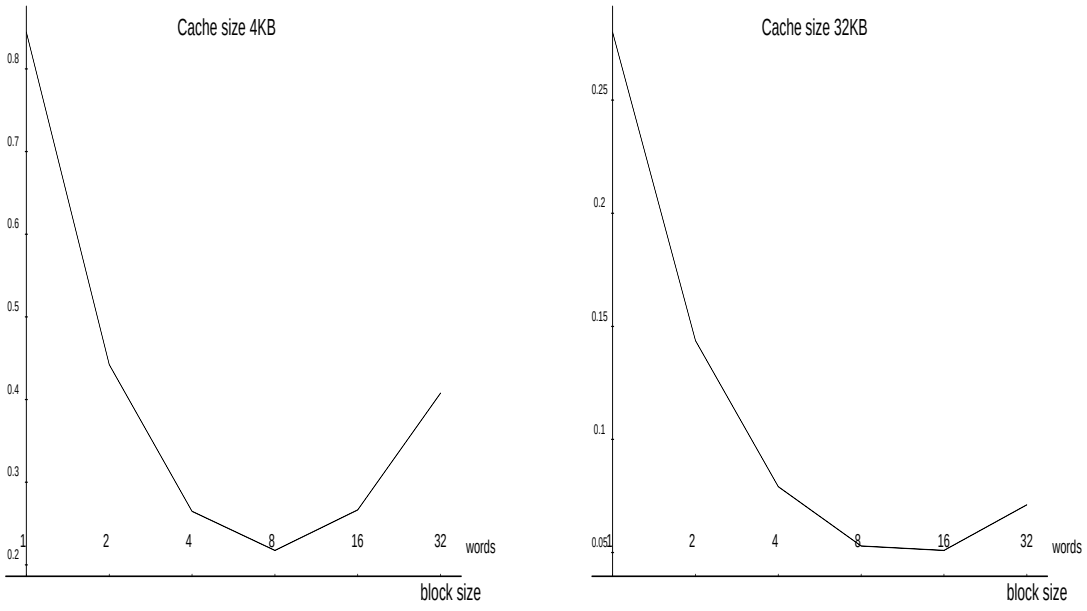
program 3	size	16	256
	proportion	1/6	5/6

5.1 Influence of the line size

Despite of the importance of the line size in the architecture of a cache, the analytic expressions of the miss ratio including this parameter are quite rare in the litterature. In [16], an empirical function has been proposed to estimate the influence of line size on the miss ratio. As the author mentions it, it is not justified by any theory, but it seems to fit reasonably well with some data.

a) The optimal line size.
There is an optimal line size, which corresponds to the minimum of the miss ratio, as a function of the line size. Different parameters change this optimal value :

- The optimal line size is relatively invariant, although increasing with cache size. For example (figure 2), for a given locality (program 3), the optimal line size varies from 8 words for a direct-mapped cache of 4 KB to 16 words for a direct-mapped cache of 256 KB.



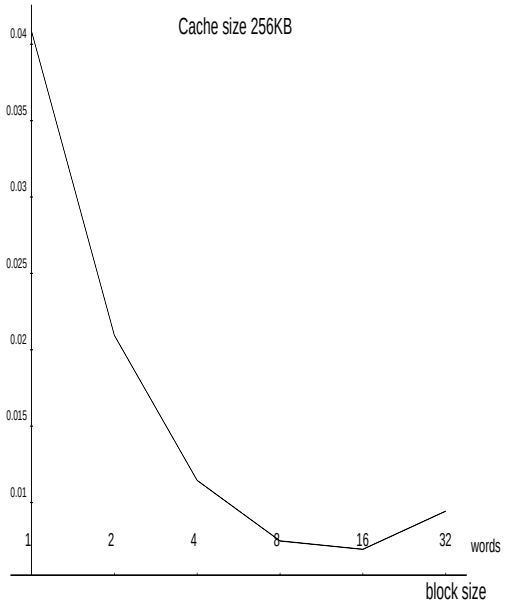


Figure 2: *Optimal block size for the miss ratio of a direct mapped cache*

- The optimal line size is slightly increasing with spatial locality. For example (figure 3), for a direct mapped cache of 4KB, the optimal line size varies from 2 words (program 1) to 8 words (program 3). From trace-driven simulations, it has been observed that the optimal line size is relatively invariant with the set of programs chosen. If we look at our examples, this phenomenon is quite apparent. For example, in program 3, most of the accesses are 256 words long, but the optimal line size is nevertheless quite small: 8 words.

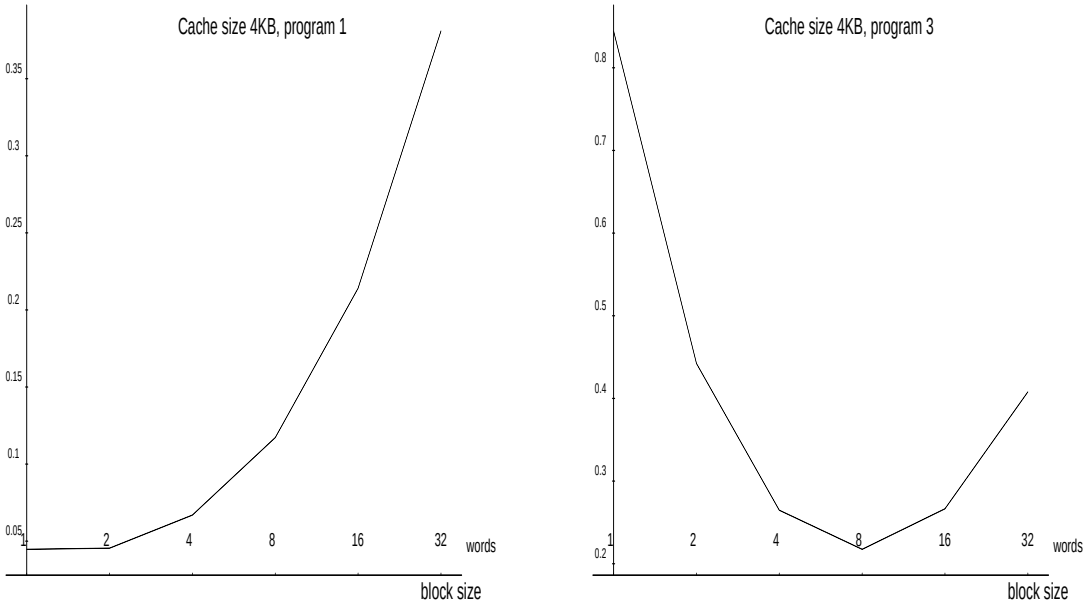


Figure 3: *Influence of locality on the optimal block size*

- The optimal line size is increasing with associativity. For example (figure 4), for a cache of 16KB and a given locality (program 2), the optimal line size varies from 2 words (direct mapped cache) to 8 words (4-associative cache).

  To estimate the effect of associativity, we will use as in [10] the spread miss ratio which is the function $S : \ n \rightarrow \frac{D(n)-D(2n)}{D(2n)}$ where $D(n)$ is the miss ratio of an n-associative cache.

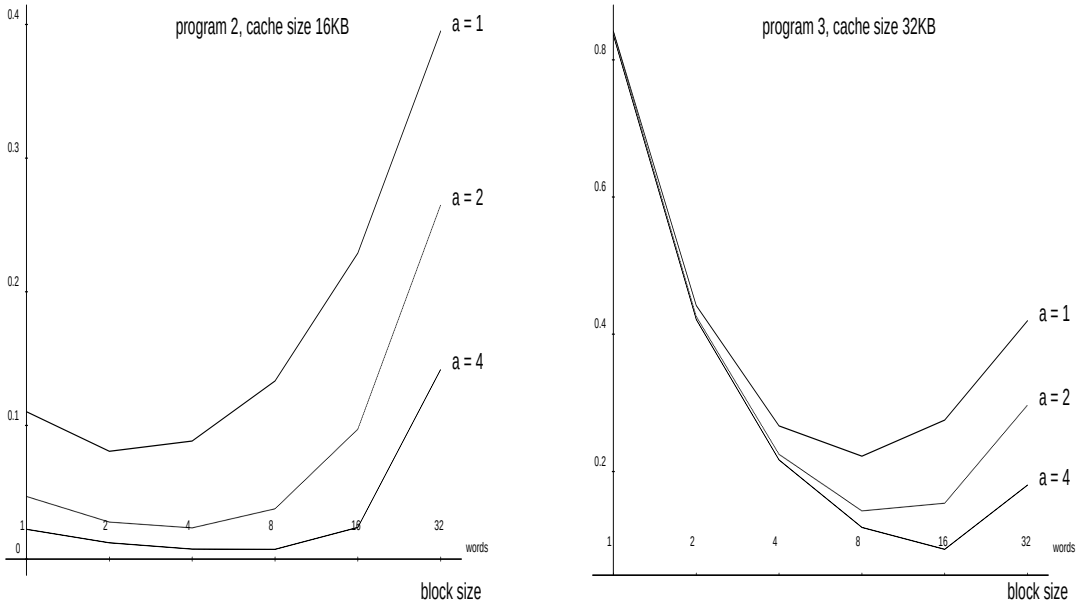


Figure 4: *Miss ratios for associativity 1,2,4*

b) Influence of line size on miss ratio spreads.
The previous observations (see [10]) by trace-driven simulations seem to show that miss ratio spreads are positively correlated with line size. Our study (see figure 4) agree with these observations for the values of line size examined by [10] lines of 4, 8 and 16 words, also for smaller values of line size (lines of 1 and 2 words), but no more for larger line sizes.

c) The optimal line size for execution time.
If we use the execution time per reference as the metric instead of the miss ratio then the optimal line is smaller than the optimal line for the miss ratio (see [14]). The time to transfer a line from the main memory to the cache can be expressed as $la + l \times tr$, where la is the latency for the access to the first word and tr the transfer time per word. Hence, large line sizes induce a larger penalty on the execution time than on the miss ratio. In the example of figure 5, for program 3, the latency is 360 and the transfer time 60ns/word (see [16]); the cache size is 32KB:

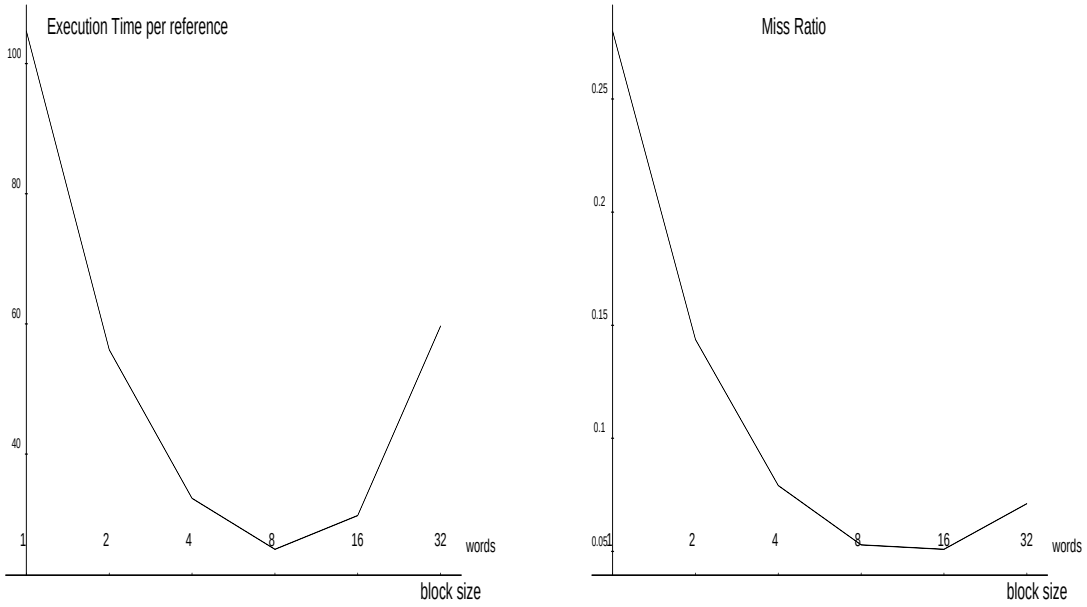


Figure 5: *Execution time and Miss ratio*

5.2 Influence of cache size

In one of the first studies ([6]), Chow empirically assumes a power function of the form $D = AC^B$ for the miss ratio where A and B are constants, without analytical basis or experimental validation. Smith ([17]) validates the approximation for a given set of results for a certain range of parameters, choosing appropriate values of constants. According to Propositions 4 and 5, the miss ratio can be expanded as $\sum_{i=1}^{n} \frac{a_i}{C^i}$. Hence, for large cache sizes, miss ratio can be approximated to $\frac{a_1}{C}$ which implies that the spread miss ratio is constant for large cache sizes (see [10]).

5.3 Influence of associativity

We will examine the changes on miss ratio due to associativity. The motivation for set associativity is two fold : the expected reduction in the miss ratio over a direct-mapped cache of the same size, and a large set size imposed by the virtual memory constraints (see [14] for details).

a) The miss ratio is reduced by associativity.
We have seen in Section 4 that there are some examples where the direct-mapped cache performs better than an associative cache. In fact, analytical expressions cannot be compared easily for fixed starting points of elementary accesses. Using expressions for miss ratios in Proposition 5, it can be proved that miss ratios for a 2-associative cache are less than for a direct-mapped cache with the same cache and program parameters i.e. C, l, h and K. For associativity greater than two, the comparison becomes quite tedious. Hence, for our model, a 2-associative cache performs better in average than a direct-mapped cache.

b) Miss ratio spreads with more restricted associativity are larger.
From trace-driven simulations, it appears that if the change from direct-mapped to two way set associativity is significant, smaller improvements are seen for set sizes above two (see [14] and [10]). In our model, because of the regularity of the accesses, the values of spread miss ratios are generally larger that the usual values encountered in the literature. Nevertheless the phenomena remain the same for the examples we considered: The gain from direct mapped to 2-associative is larger than 2-associative to 4-associative (see figure 4).

Conclusion. Our analytical results generally agree with the results from trace-driven simulations on miss ratio and execution time: existence of an optimal line size, smaller optimal line size for execution time than for miss ratio, relative invariance of the optimal line size varying the cache size or the locality, reduction of the miss ratio due to associativity. Nevertheless the influence of locality is more tractable with our model and the range of parameters explored can be extended.

6 Appendix

Proof of Propositions 4 and 5 :

We denote by a the associativity and $\mathcal{E}$ the set of a lines

$$0, \frac{C}{a}, \ldots, \frac{(a-1)C}{a}$$

where the first line 0 is denoted $\mathcal{L}$.

We denote by $N_{j,\mathcal{E}}$ the process (i.e. the sequence of times) of the accesses to set $\mathcal{E}$ by the elementary access j, i.e. $N_{j,\mathcal{E}}[t,t'[$ is the number of accesses in $\mathcal{E}$ by j between t and t'. Similarly $N^1_{j,\mathcal{E}}$ is the process of the accesses of type 1 to set $\mathcal{E}$.

We recall that, provided that each elementary region length is less than $\frac{C}{a} - l$, each elementary access requires at most one line of set $\mathcal{E}$. For this reason, the $j-th$ access has a miss to access $\mathcal{E}$ at time t, if and only if the number of elementary accesses different from j requiring $\mathcal{E}$ between the last time before t where j requires $\mathcal{E}$ and time t is greater than a, the number of lines in $\mathcal{E}$.

Let us take the example of the accesses of type 1, in order to derive D_1. The sequence of accesses of type 1 of j requiring line $\mathcal{L}$ is $(t^1_j + m\ L_j)_{m\geq 0}$ where t^1_j is the first of these instants. Hence, if j requires line $\mathcal{L}$, the number of misses of type 1 is

$$m^1_j(t) = \sum_{m, t^1_j+mL_j\leq t} 1_{\{\exists A\subset\{1,\ldots,n\}, j\notin A,\ card\ A=a, \forall k\in A\ N_{k,\mathcal{E}}[T_{j,\mathcal{L}}(t^1_j+mL_j), t^1_j+mL_j[\neq 0\}}$$

where $T_{j,\mathcal{L}}(t)$ is the last time before t that access j requires line $\mathcal{L}$. Straightforward manipulations give

$$\begin{aligned} m^1_j(t) &= \sum_{m, t^1_j+mL_j\leq t} 1 - 1_{\{\forall A\subset\{1,\ldots,n\}, j\notin A,\ card\ A=a, \exists k\in A, N_{k,\mathcal{E}}[T_{j,\mathcal{L}}(t^1_j+mL_j), t^1_j+mL_j[=0\}} \\ &= \sum_{m, t^1_j+mL_j\leq t} 1 - \prod_{\substack{A\subset\{1,\ldots,n\} \\ j\notin A, cardA=a}} 1_{\{\exists k\in A, T_{k,\mathcal{E}}(t^1_j+mL_j)<T_{j,\mathcal{L}}(t^1_j+mL_j)\}}. \end{aligned}$$

It is then easy to see that the last point of $N_{k,\mathcal{E}}$ before $t^1_j + mL_j$ to be possibly before $T_{j,\mathcal{L}}(t^1_j + mL_j)$ is $T^1_{k,\mathcal{E}}(t^1_j + mL_j) + (q_k - 1)$ if $T^1_{k,\mathcal{E}}(t)$ is the last point of $N^1_{k,\mathcal{E}}$ before t and $T_{j,\mathcal{L}}(t^1_j+mL_j)$ is simply $t^1_j+(m-1)L_j+q_j-1$. Thus

$$m^1_j(t) = \sum_{m=1}^{l^1_j(t)} 1 - \prod_{\substack{A\subset\{1,\ldots,n\} \\ j\notin A, cardA=a}} 1_{\{\exists k\in A, t^1_j+mL_j-T^1_{k,\mathcal{E}}(t^1_j+mL_j)>L_j-q_j+q_k\}}$$

where $l_j^1(t) = \text{card}\ \{m \geq 1, t_j^1 + mL_j \leq t\}$ is the number of accesses of type 1 for j in line $\mathcal{L}$ before t.
If $(t_j^1 + mL_j - t_k^1)\ mod\ L_k \geq L_j - q_j + q_k$ then, for some integer a,

$$t_k^1 + aL_k + L_j - q_j + q_k < t_j^1 + mL_j < t_k^1 + (a+1)L_k,$$

which means that

$$t_j^1 + mL_j - T_{k,\mathcal{E}}^1(t_j^1 + mL_j) > L_j - q_j + q_k.$$

This yields that

$$m_j^1(t) = \sum_{k=0}^{l_j^1(t)} 1 - \prod_{\substack{A \subset \{1,\ldots,n\} \\ j \notin A,\ card\ A = a}} 1_{\{\exists k \in A,\ (\frac{t_j^1 + mL_j - t_k^1}{L_k})\ mod\ 1 \geq \frac{L_j - q_j + q_k}{L_k}\}}. \qquad (6)$$

We denote by α the vector $((\alpha_j)_{1 \leq i \leq n, i \neq j})$ where $\alpha_i = \frac{L_j}{L_i}$ and by T the translation on the torus $[0,1]^{n-1}$ defined by $T((x_i)_i) = (((x_i + \alpha_i)\ mod\ 1)_{1 \leq i \leq n, i \neq j})$.

Hence, $m_j^1(t)$ can be rewritten as

$$m_j^1(t) = \sum_{m=0}^{l_j^1(t)} f(T^m(y_j))$$

where

$$y_j = \left(\frac{t_j^1 - t_1^1}{L_1}, \ldots, \frac{t_j^1 - t_n^1}{L_n}\right),\ T^m = \underbrace{T \circ \ldots \circ T}_{m\ times}$$

and

$$f(x) = 1 - \prod_{\substack{A \subset \{1,\ldots,n\} \\ j \notin A,\ card\ A = a}} 1_{\bigcup_{k \in A}\{x_k \geq \frac{L_j - q_j + q_k}{L_k}\}}.$$

A classical theorem of ergodic theory on the invariant measures of translations of the torus (see Cornfeld [8] for example) ensures that

$$\lim_{l_1 \to +\infty} \frac{1}{l_1} \sum_{k=1}^{l_1} f(T^k(x)) = \int_{[0,1]^{n-1}} f(u)\, du_1 \ldots du_{n-1},$$

Lebesgue almost surely on $[0,1]^{n-1}$, provided that the $\alpha_1, \ldots, \alpha_{n-1}$ are rationally independent, i.e. if for some integers $k_0, \ldots, k_{n-1}$ the relation $k_0 + \sum_{i=1}^{n-1} k_i \alpha_i = 0$ holds, then $k_i = 0$ for $i = 0, \ldots, n-1$ (the vectors $1, \alpha_1, \ldots, \alpha_{n-1}$ are free in the $\mathbb{Z}$-module $\mathbb{R}$). This property is true in our case where the L_j, $j = 1, \ldots, n$ are almost surely rationally independent because their distribution is absolutely continuous with respect to the Lebesgue measure.

Now, in order to calculate the right-hand side of (6), we need some algebra. We denote by B_k the set $\{x_k \geq \frac{L_j - q_j + q_k}{L_k}\}$. We will prove that

$$\prod_{\substack{A \subset \{1,\ldots,n\} \\ j \notin A,\, card\ A = a}} 1_{\bigcup_{k \in A} B_k} = \sum_{i=1}^{a} c_i \sum_{j \in A,\, card\ A = a-i} \prod_{k \notin A \cup \{j\}} 1_{B_k} \tag{7}$$

where the c_i are given by the equation (5). For this, one verifies that, if we denote by m the number of k different from j such that $1_{B_k} = 0$, then

$$\sum_{\substack{A \subset \{1,\ldots,n\} \\ j \notin A,\, card A = a-i}} \prod_{k \notin A \cup \{j\}} 1_{B_k} = \begin{cases} \binom{a-i-m}{n-m-1} & \text{if } 0 \leq m \leq a-i \\ 0 & \text{if } m > a-i\,. \end{cases} \tag{8}$$

Then, it is easy to see that equation (7) is equivalent to the system

$$1 = \sum_{i=1}^{a-m} c_i \binom{a-i-n}{n-m-1} \quad (0 \leq m < a)\,. \tag{9}$$

Indeed, if $0 \leq m < a$, then equation (7) yields (9). Conversely, if (9) is verified, this gives the result in the case $m < a$ and when $m \geq a$, equation (7) is true because its left-hand side and all the terms of its right-hand side are 0.

Eventually, (9) is a linear system, associated to a triangular matrix with diagonal elements 1. Thus, there is a unique solution $(c_i, 1 \leq i \leq a)$ for the system (9) given by,

$$c_l = 1 - \sum_{i=1}^{l-1} c_i \binom{l-i}{n-a+l-1} \quad (1 \leq l \leq a)$$

which is (5) in Proposition 5.

Hence, rewriting function f using equation (7) and gathering the results, we obtain that for fixed L_j, q_j $(1 \leq j \leq n)$,

$$\lim_{t \to +\infty} \frac{m_j^1(t)}{l_j^1(t)} = 1 - \sum_{i=1}^{a} c_i \sum_{\substack{A \subset \{1,\ldots,n\} \\ j \notin A, cardA = a-i}} \prod_{k \notin A \cup \{j\}} a_{k,j}. \tag{10}$$

where

$$\begin{aligned} a_{k,j} &= 1_{q_k=0} + 1_{q_k \neq 0} \int_0^1 1_{x_k \geq \frac{L_j - q_j + q_k}{L_k}} dx_k \\ &= 1_{q_k=0} + 1_{q_k \neq 0} (1 - \frac{L_j - q_j + q_k}{L_k})^+ \end{aligned}$$

where b^+ is the positive part of b $(\max(0, b))$.

Then, using that

$$P(q_k \neq 0) = \frac{L_k + l - 1}{\frac{C}{a}},$$

integrating with respect to q_k, $k \neq j$, we obtain after some transformations that

$$a_{k,j} = 1 - \frac{L_k + l - 1}{\frac{C}{a}} E\left(1 \wedge \frac{L_j - q_j + q_k}{L_k} / q_j, q_k \neq 0\right).$$

Using that $\lim_{t \to +\infty} \frac{l_j^1(t)}{l_j(t)} = \frac{1}{q_j}$ where $l_j^1(t)$ is the total number of accesses of j to set $\mathcal{E}$, integrating again, we get for $D_1 = \frac{1}{n} \sum_{j=1}^{n} \lim_{t \to +\infty} \frac{m_j^1(t)}{l_j(t)}$ that

$$D_1 = E\left(\frac{1}{q_1}(1 - \sum_{i=1}^{a} c_i \binom{n-1}{a-i} I^{n-a+i-1})\right)$$

where

$$I = E\left(1 - \frac{L_2 + l - 1}{\frac{C}{a}} E\left(1 \wedge \frac{L_1 - q_1 + q_2}{L_2} / q_2 \neq 0\right) / q_1\right).$$

This gives the first part of our proposition. For the misses of type 2, the proof follows exactly the same lines, starting from the following expression for the number of misses of type 2 up to time t,

$$m_j^2(t) = \sum_{\substack{m \geq 1, 1 \leq m' \leq q_j, \\ t_j^1 + mL_j + m' \leq t}} 1_{\{\exists \mathrm{A} \subset \{1,\ldots,\mathrm{n}\}, \mathrm{j} \notin \mathrm{A},\ \mathrm{card\ A=a}, \forall \mathrm{k} \in \mathrm{A}\ \mathrm{N_{k,\mathcal{E}}[T_{j,\mathcal{L}}(t_j^1+mL_j+m'), t_j^1+mL_j+m'[\neq 0\}}$$

which gives

$$m_j^2(t) = \sum_{m=1}^{l_j^2(t)} 1 - \prod_{\substack{A \subset \{1,\ldots,n\} \\ j \notin A,\ card A = a}} 1_{\{\exists k \in A, t_j^1 + mL_j + m' - T^1_{k,\mathcal{E}}(t_j^1 + mL_j + m') > q_k\}}$$

where $l_j^2(t) = \text{card } \{m \geq 1, , 1 \leq m' \leq q_j, t_j^1 + mL_j \leq t\}$ is the number of accesses of type 2 for j in line $\mathcal{L}$ before t.

References

[1] Agarwal, A., Beng-Hong, L., Kranz, D., and Kubiatowicz, J. APRIL: A processor architecture for multiprocessing. In *Proc. of the 17th Annual Symposium on Computer Architecture* (New York NY (USA), May 1990), IEEE, pp. 104–114.

[2] Agarwal, A., Horowitz, M., and Hennessy, J. An analytical cache model. *ACM Transactions on Computer Systems 7*, 2 (May 1989), 184–215.

[3] Aven, O. I., Coffman, E. G., and Kogan, Y. A. *Stochastic Analysis of computer storage*. Reidel, Amsterdam, 1987.

[4] Aven, O. I., and Kogan, Y. A. Brief paper stochastic control of paging in a two-level computer memory. *Automatica 11* (1975), 309–313.

[5] Borg, A., Kessler, R. E., Lazana, G., and Wall, D. W. Long address traces from risc machines: Generation and analysis. Tech. Rep. 89/14, DEC Western Research Laboratory, Palo Alto, California, Sept. 1989.

[6] Chow, P., and Horowitz, M. Architectural tradeoffs in the design of mips-x. In *Proc. of the 13th Annual Int. Symp. on Computer Architecture* (University Stanford, 1987), ACM, pp. 300–308.

[7] Coffman, E. G., and Denning, P. J. *Operating Systems Theory.* Prentice-Hall, 1973.

[8] Cornfeld, I. P., Fomin, S. V., and Sinai, Y. G. *Ergodic Theory*, vol. 245 of *Grundlehren der mathematischen Wissenschaften.* Springer Verlag, 1982.

[9] Fricker, C., and Robert, P. Memory reference model for the cache memories analysis. In *Performance'90: International Symposium on Computer Performance* (Edinburgh, Sept. 1990).

[10] Hill, M., and Smith, A. Evaluating associativity in CPU caches. *IEEE Transactions on Computers 38*, 12 (Dec. 1989), 1612–1630.

[11] Lam, M., Rothberg, E., and Wolf, M. The cache performance and optimizations of blocked algorithms. Tech. rep., Stanford University, 1991.

[12] Lenoski, D., Laudon, J., Gharachorloo, K., Gupta, A., and Hennessy, J. The directory-based cache coherence protocol for the DASH multiprocessor. In *Proc. of the 17th Annual Symposium on Computer Architecture* (New York, May 1990), IEEE, pp. 148–160.

[13] Przybylski, S., Horowitz, M., and Hennessy, J. Performances tradeoffs in cache design. In *Proc. of the 15th Annual IEEE Int. Symp. on Computer Architecture* (1988). IEEE.

[14] Przybylski, S., Horowitz, M., and Hennessy, J. Characteristics of performance-optimal multi-level cache hierarchies. In *Proc. of the 15th Annual Int. Symp. on Computer Architecture* (1989), ACM.

[15] Rao, G. S. Performance analysis of cache memories. *Journal of the ACM 25*, 3 (July 1978), 378–395.

[16] Smith, A. Line (block) size choice for CPU cache memories. *IEEE Transactions on Computers 36*, 9 (Sept. 1987), 1063–1075.

[17] Smith, A. J. Cache memories. *ACM Computing Surveys 14*, 3 (Sept. 1982), 473–530.

[18] Torrellas, J., Weil, T., and Hennessy, J. L. Analysis of some architectural and programming tradeoffs in a large scale shared memory multiprocessor, Feb. 1989.