Near-Optimal Loop Tiling by means of Cache Miss Equations and Genetic Algorithms

Jaume Abella, Antonio González, Josep Llosa y Xavier Vera

Resumen— **The effectiveness of the memory hierarchy is critical for the performance of current processors. The performance of the memory hierarchy can be improved by means of program transformations such as loop tiling, which is a code transformation targeted to reduce capacity misses. This paper presents a novel systematic approach to perform near-optimal loop tiling based on an accurate data locality analysis (Cache Miss Equations) and a powerful technique to search the solution space that is based on a genetic algorithm. The results show that this approach can remove practically all capacity misses for all considered benchmarks. The reduction of replacement misses results in a decrease of the miss ratio that can be as significant as a factor of 7 for the matrix multiply kernel.**

Palabras clave— **Loop tiling, Cache miss equations, Genetic Algorithms, Code transformations.**

I. Introduction

MEMORY performance is critical for the performance of current computers. Memory is organized hierarchically in such a way that the upper levels are smaller and faster. The uppermost level typically has a very short latency (e.g. 1-2 cycles) but the latency of the lower levels may be a few orders of magnitude longer (e.g. main memory latency may be around 100 cycles). Thus, techniques to keep as much data as possible in the uppermost levels are key to performance.

In addition to the hardware organization, it is well known that the performance of the memory hierarchy is very sensitive to the particular memory reference patterns of each program. The reference patterns of a given program can be changed by means of transformations that do not alter the semantics of the program. These program transformations can modify the order in which some computations are performed or can simply change the data layout. *Loop Tiling* is an example of the former family of techniques. Loop tiling [1] is a technique based on a combination of strip-mining and loop interchange.

Loop tiling has a significant potential to remove cache misses. Loop tiling can remove most capacity misses by restructuring the loop and changing the order in which statements are executed. However, finding the optimal loop tiling for a given program is a very complex task, since the options are almost unlimited and exploring all of them is infeasible. For very simple programs, the programmer intuition may help but in general, a systematic approach that can be integrated into a compiler and can deal with any type of program is desirable. This systematic approach requires the support of a locality analysis method in order to assess the performance of different alternatives.

In this paper, we propose an automatic approach to perform loop tiling in numeric codes. It is based on a very accurate technique to analyze the locality of a program that is known as Cache Miss Equations (CMEs) and a genetic algorithm in order to search the solution space. The proposed genetic algorithm converges very fast, and although it does not guarantee that the optimal solution is found, we show that after loop tiling, the replacement miss ratio of the evaluated benchmarks is almost negligible. Therefore, for these programs the results are near-optimal. The rest of this paper is organized as follows. Section II overviews the locality analysis approach. Section III presents the loop tiling technique and its performance is evaluated in section IV. Section V summarizes the main conclusions of this work.

II. Memory Locality Analysis

To effectively transform a code in order to optimize memory performance, an effective memory locality analysis is required in order to assess the different alternatives. In this section, we describe the locality analysis technique used in this work to perform loop tiling.

In particular, we use Cache Miss Equations (CMEs) [2] to represent the cache behavior. Cache Miss Equations are a very accurate analytical model of the cache memory. They describe the cache behavior by means of diophantine equations, which allows us to use mathematical techniques to compute the locality of each memory reference. Unfortunately a direct solution of these equations is computationally intractable due to its NP nature. Statistical methods and some techniques based on polyhedra theory have been proposed to solve the equations in a reasonable amount of time [3], [4].

A. Overview of Cache Miss Equations

Cache Miss Equations are a set of equations[1] that represent all the potential cache misses for the references in a loop nest. They describe the precise relationship among the iteration space, array sizes, base addresses, and the cache parameters for a loop nest. This section presents an overview of the CMEs. For more details about CMEs see [2].

In order to generate CMEs, the *reuse vectors* [5] of all the references in a loop nest must be gener-

Departmento de Arquitectura de Computadores, Universitat Politècnica de Catalunya, Campus Nord, Barcelona (España).

[1]The term equation is loosely used to refer to a set of simultaneous equalities and inequalities.

```
parameter (N)
REAL a(N,N), b(N,N), c(N,N)
do i = 1, N
   do j = 1, N
      do k = 1, N
         a(i,j) = a(i,j) + b(i,k) * c(k,j)
      enddo
   enddo
enddo
```

Fig. 1. Matrix multiply algorithm

ated. *Reuse vectors* provide information about the potential reuses in the entire iteration space. Figure 1 shows the *matrix multiply* kernel. For instance, $\vec{r} = (0,0,1)$ is a reuse vector for reference *c(k,j)*, because the data accessed by this reference in a given iteration is potentially reused one iteration later (both are potentially mapped into the same cache line). In order to determine if these potential reuses are realized, CMEs are generated and studied (it is not necessary to solve the equations to find realized reuses).

For every reuse vector of a reference two types of CMEs are generated:

- **Compulsory equations.** Compulsory equations represent the first time a memory line is brought into the cache.
- **Replacement equations.** Given a reference, replacement equations represent the interferences with any other reference. For each pair of references (R_A and R_B), the following expression gives the condition that determines whether they are mapped onto the same cache set:

$$\begin{aligned} Cache\_Set(\vec{\imath})_{R_A} &= Cache\_Set(\vec{\jmath})_{R_B} \\ \vec{\jmath} &\in \mathcal{I} \end{aligned}$$

  where $\mathcal{I}$ represents the iteration points between $\vec{\imath}$ (the current one) and the iteration point from which R_A reuses. This condition is expanded into a set of equations for each reuse vector.

Due to its excesive computation cost, we use sampling techniques to study a subset of the iteration space instead of the whole iteration space [4]. The subset of points is selected using *Simple Random Sampling.*

The size of the sample is set according to the required width of the confidence interval and the desired confidence. We found experimentally that a confidence interval of width 0.1 and a 90% confidence is enough to obtain accurate miss ratios with a very small computing time (only 164 points of the iteration space must be explored). In this way, the miss ratio is computed as an interval of width 0.1 and the actual miss ratio belongs to this interval with a probability of 90%. The central point of this interval can be used as an estimation of the actual miss ratio.

B. CME for Multiple Convex Regions

CME are defined for iteration spaces that are single convex regions, but after tiling n dimensions the iteration space is the union of 2^n convex regions.

Figure 2 shows how the iteration space of a one-dimensional loop becomes a two-convex region iteration space after tiling. The shaded regions correspond to the different convex regions before and after tiling.

(a)	(b)

do i_1=1, U_1
  do i_2=1, U_2
    A(i_2,i_1) = B(i_1,i_2)
  enddo
enddo
	do ii_2=1, U_2, T_2
	  do i_1=1, U_1
	    do i_2=ii_2, min(ii_2+T_2-1, U_2)
	      A(i_2,i_1) = B(i_1,i_2)
	    enddo
	  enddo
	enddo

Fig. 3. 2D matrix transposition: (a) before tiling, (b) after tiling inner loop

There are different ways to solve this problem. The easiest option would be to use only one convex region that approximates the actual non-convex region. This convex region can be the smallest parallelepiped that includes all other convex regions (see figure 2 (c)) or alternatively, the region which does not include the last iteration of every tiled loop where the tile size is not a divisor of the upper bound (see figure 2 (d)). Both options have drawbacks. The first option includes in the convex regions points outside the iteration space, while the second option does not include points belonging to the iteration space.

Because of this, we have decided to implement a more accurate solution. The CME implementation has been modified to deal with multiple convex regions by defining the equations for every convex region and solving for every analyzed point the equations corresponding to the convex region in which the point is contained. Let n be the number of convex regions of a loop after tiling. Every compulsory equation should be defined for each convex region, so the number of compulsory equations is increased by a factor of n. For each reuse vector, we have to generate a set of replacement equations for each convex region. In addition, we have to generate a set of equations for every pair of convex regions that reflect the potential reuse between different regions. Due to this, the number of replacement equations is increased by a factor of n^2.

III. Loop Tiling

In this section we present our proposal to perform loop tiling. Tiling [1] is a transformation which combines strip-mining [1] with loop interchange to form small tiles [1] of loop iterations in order to increase the data locality. In this section T_i and U_i stand for the tile size and upper bound respectively of loop i in the original loop nest. Figure 3 shows an example of a loop before (a) and after (b) loop tiling.

A. Model

The target of our work is to obtain the values of the variables T_i that minimize the number of misses. Note that loop tiling changes only the order in which the original iteration space is traversed, so the number of compulsory misses before and after tiling remains constant. Because of this, our work focuses on

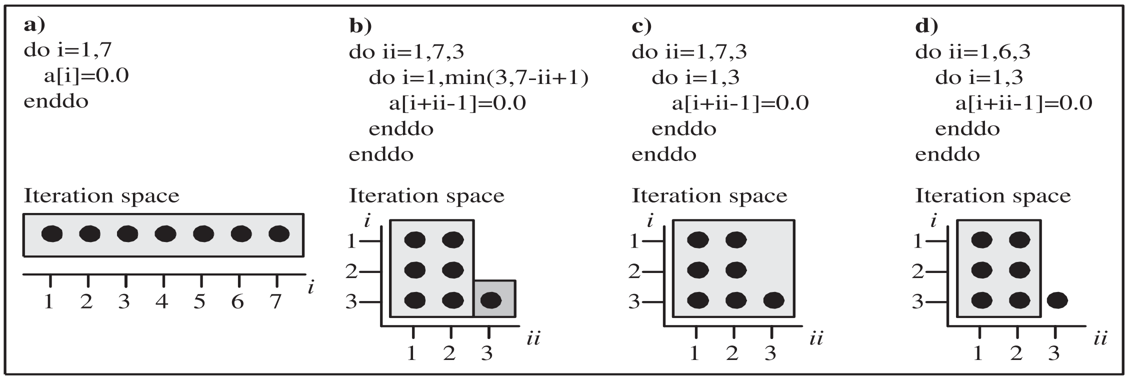

Fig. 2. Example of iteration space: (a) before tiling, (b) after tiling, (c) bigger region, (d) smaller region

minimizing the number of replacement misses, which include both capacity and conflict misses.

Let f be the function that represents the number of replacement misses for each possible value of the tile size variables:

$$f \quad \longmapsto \quad \#Replacement\ Misses \qquad (1)$$

$$f : \underbrace{[1, U_1]}_{T_1} \times \ldots \times \underbrace{[1, U_k]}_{T_k} \longrightarrow \mathbb{Z}$$

Our problem can be expressed as follows:

$$MIN \quad f(T_1, \ldots, T_k)$$

$$1 \leq T_i \leq U_i$$

$$i = 1 \ldots k$$

where f is called the *objective function*.

The objective function consists of the CMEs generated in a parameterized way, where the parameters are the tile sizes.

Since f is a pseudo-polynomial function, the relationship between loop tiling and the number of misses is nonlinear. T_i can take only integer values, thus, our problem can be seen as a nonlinear integer optimization (NLP) one.

Many researchers have studied NLP. A well studied case is the one where the constraints are linear (named *linearly constrained optimization*). A special case is when the objective function is entirely linear; this is called Linear Programming (LP). Algorithms to solve both real (e.g simplex) and integer functions (e.g branch & bound scheme) can be found in the literature.

One of the challenges in NLP is that some problems exhibit local minima and search algorithms can be stuck at them. Algorithms that propose to overcome this problem are named *Global Optimization*. Real functions have been studied deeply. Unfortunately, integer functions are hard to optimize. There are some studies based on {0,1} valued integer functions, but in general, this is a hard and time-consuming problem. Hence, the use of heuristics is necessary. Tabu search [6] obtains promising theoretical results, but only partial implementations have been reported so far. On the other hand, simulated annealing [7] and genetic algorithms [8] have been used for years with very good results.

Our proposal is based on the use of a genetic algorithm to optimize function f.

```
ALGORITHM:
Supply a population P_0
i=1
while (not finish)
   P_i=Selection(P_{i-1})
   P_i=Reproduce(P_i)
   i=i+1
end
```

Fig. 4. Simple Genetic Algorithm

B. Genetic Algorithm

Algorithms for function optimization are generally limited to convex regular functions. However, there are lots of functions that are not continuous, non differentiable or multi-modal. It is common to solve this problem by means of stochastic sampling. Whereas traditional search techniques use characteristics of the problem to determine the next sampling point (e.g Gradient), stochastic methods use non-deterministic decision rules.

Genetic Algorithms (GAs) are a particular type of stochastic methods that have been used to solve hard problems with objective functions that do not meet the properties required by traditional methods [8]. These algorithms search in the solution space of a function simulating the Nature-based process of evolution, that is, the survival of the fittest. Usually, the fittest individuals tend to reproduce more than the inferior individuals, and they survive to the next generation propagating the best genes.

GAs simulate the evolution of a population. Figure 4 shows the simplest GA. It starts from a random generated population, and it makes the population evolve by means of basic genetic operators (selection, mutation and crossover) [8] applied to individuals of the current population, to produce an improved next generation.

C. Genetic Algorithm Parameters

The use of GAs requires the determination of the following issues: chromosome representation, selection function, genetic operators, the creation of the initial population and the termination criteria.

In our context, each individual represents a particular tiling solution, which is identified by the particular values of the tile size variables T_i. Individuals are made up of a set of chromosomes, each one being associated to a tile size variable. Each chromosome is made up of a sequence of genes, each gene being represented by a digit of a certain alphabet. We have experimentaly observed that using the alphabet {00, 01, 10, 11} produces good results.

The function to transform the chromosome values into tile sizes is not the identity function. Tile size T_i can take any value in the range $[1 \ldots U_i]$. On the other hand a chromosome is represented by a sequence of genes encoded in a binary representation. Thus, each chromosome will be represented by a value in the range $[0 \ldots 2^k\text{-}1]$ where k is $\lceil log_2 U_i \rceil$. If k is an odd number, k is increased in 1 because of the alphabet we have used to represent genes. Thus, there are more values in the representation range for a chromosome than possible tile size values. Therefore, we need a function to map values $[0 \ldots 2^k\text{-}1]$ into the range $[1 \ldots U_i]$.

Let g be the function that represents the tile size for each possible value of a chromosome:

$$g : [0 \ldots 2^k - 1] \longrightarrow [1 \ldots U_i]$$

where

$$k = \lceil log_2 U_i \rceil \qquad (+1 \text{ if odd})$$
$$x \in [0 \ldots 2^k\text{-}1]$$

$$g(x) = \lfloor \frac{x * (U_i - 1)}{2^k - 1} \rfloor + 1 \tag{2}$$

It can be deduced that every possible tile size has at least one representation.

Example. Let us assume a two-dimensional nested loop where the upper bounds of the loops are 10 and 100 respectively. Thus, $\lceil log_2 10 \rceil = 4$ and $\lceil log_2 100 \rceil = 7$, so we have $k_1 = 4$ and $k_2 = 8$ respectively. Therefore, the first chromosome is represented by 2 genes, and the second one by 4 genes. For instance, the value 12 (1100) and 74 (01001010) correspond to the tile sizes 8 ($g_1(12)$=8) and 29 ($g_2(74)$=29) respectively, and are represented by the following genes:

$$\underbrace{\underbrace{\underbrace{11}_{gene_{1_0}=3}\;\underbrace{00}_{gene_{1_1}=0}}_{chromosome_1}\;\underbrace{\underbrace{01}_{gene_{2_0}=1}\;\underbrace{00}_{gene_{2_1}=0}\;\underbrace{10}_{gene_{2_2}=2}\;\underbrace{10}_{gene_{2_3}=2}}_{chromosome_2}}_{individual}$$

Genetic operators provide the basic search mechanism by creating new solutions based on the solutions that exist. The *selection* of individuals to produce successive generations plays an extremely important role. A common selection approach assigns probability of selection to each individual depending on its fitness. Individuals with higher fitness have a higher probability of contributing one or more offsprings to the next generation. We have adopted one of the selection schemes that gives better results, which is known as *remainder stochastic selection without replacement* [8]. Let us call N the size of the population (number of individuals). This selection scheme consists in choosing N individuals from the N individuals of the previous generation. In this selection process a given individual can be chosen more than once. The chosen individuals are grouped forming pairs and crossover is applied to each pair with a given probability. In the case they do not crossover, both individuals are added to the new population (see Figure 5).

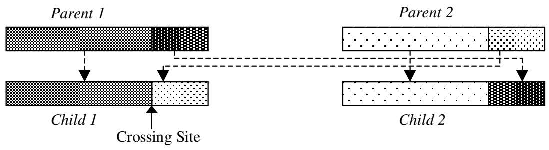

Fig. 5. Schematic of simple crossover

```
ALGORITHM:
finish := false
iters := 0
while (not finish)
  if (iters<15)
    iters = iters + 1
    create next generation
  else if (iters>=15 and iters<25)
    if (not converge())
      iters = iters + 1
      create next generation
    else finish := true
    endif
  else finish := true
  endif
endwhile
```

Fig. 6. Genetic Algorithm used to perform loop tiling

Crossover takes two individuals and produces two new individuals merging the genetic material in a random point (named cross site). *Mutation* is applied after crossover to each individual with a given probability. Mutation changes one individual to produce a new one by flipping some of its genes. Both crossover probability and mutation probability have to be determined empirically, and are related to the size of the population.

The GA must be provided with an initial population (see Figure 4) that is created randomly. GAs moves from generation to generation, and the usual termination criterion is the number of generations, although other criteria can be used [8].

Our experiments show that an initial population of size equal to 30, with crossover probability of 0.9 and a mutation probability of 0.001, gives near-optimal results in most cases after 15 generations. In the rest of the cases, near-optimal results are obtained after a number of generations between 15 and 25. Figure 6 shows the algorithm to decide the number of generations required before the optimal tile search

Kernel	Program	Nested loops
T2D		2
T3DJIK		3
T3DIKJ		3
JACOBI3D		3
MATMUL		3
MM	LIVER.	3
ADI	LIVER.	2
ADD	NAS	4
BTRIX	NAS	3
VPENTA1	NAS	2
VPENTA2	NAS	2
DPSSB	BIHAR	3
DPSSF	BIHAR	3
DRADBG1	BIHAR	3
DRADBG2	BIHAR	3
DRADFG1	BIHAR	3
DRADFG2	BIHAR	3

TABLA I
EVALUATED KERNELS

Kernel	8KB		
	Original	Padding	Padding + tiling
ADD	60.2%	59.8%	0.5%
BTRIX	50.1%	0.2%	0.2%
VPENTA1	78.3%	52.4%	0.0%
VPENTA2	86.0%	11.9%	0.0%
ADI_1000	26.2%	12.3%	4.1%
ADI_2000	25.7%	12.4%	3.4%

Kernel	32KB		
	Original	Padding	Padding + tiling
ADD	60.2%	59.8%	0.0%
BTRIX	34.1%	0.0%	0.0%
VPENTA1	78.1%	32.9%	0.0%
VPENTA2	86.0%	11.3%	0.0%

TABLA II
MISS RATIO FOR SOME EVALUATED KERNELS BEFORE PADDING AND TILING, AFTER PADDING AND AFTER PADDING AND TILING

Cache sizes	Replacement miss ratio		
	<1%	<2%	<5%
8KB	56.4%	79.5%	100.0%
32KB	90.2%	97.6%	100.0%

TABLA III
REPLACEMENT MISS RATIOS AFTER TILING OF ALL KERNELS EXCEPTING THOSE IN TABLE II

stops, where *converge()* is a function to decide when the population is homogeneous enough. In our case we consider that a population converges when the best individual has a difference of replacement misses smaller than 2% with respect to the population average of its generation. We have observed in the evaluated loops that this convergence criterion is only achieved if the population is close to the optimal.

IV. PERFORMANCE EVALUATION

This section evaluates the proposed loop tiling approach. Examples of its use will be given, as well as its accuracy.

A. Experimental Framework

CMEs have been implemented for fortran codes through the Polaris Compiler [9] and the Ictineo library [10]. These libraries allow us to obtain all the compile-time information needed to generate the equations. The evaluation of CMEs has been implemented in C++ using our own polyhedra representation [3]. Due to CMEs restrictions, only perfectly nested loops in which the array subscript expressions are affine functions of the induction variables are analyzed [2].

The loop nests considered are some kernels from different programs (NAS[2], BIHAR[3], LIVERMORE) and some frequently used kernels [11] (see Table I). These loops have been chosen because they exhibit high number of capacity misses. Results for different cache architectures are reported.

B. Performance Evaluation for some Kernels

In this section we present the replacement miss ratio of different direct-mapped caches (8KB and 32KB) before and after applying loop tiling for all the benchmarks of the table I. Results are shown in Figures 7 and 8, where the number following the program name corresponds to the problem size.

[2]Numerical Aerospace Simulation Facility
[3]Biharmonic Partial differential equations solver

It can be seen that for most programs practically all replacement misses are removed after tiling, which implies that near-optimal solutions have been found for these programs. However, for some kernels (ADD, BTRIX, VPENTA1 and VPENTA2) the replacement miss ratio obtained after tiling is still quite high for all cache sizes. For some others (ADI_1000 and ADI_2000) replacement misses after tiling are only significant for a 8KB cache. We have analyzed these cases carefully and we have observed that most of the remaining replacement misses are due to conflicts and they cannot be removed by loop tiling. For these programs we have investigated the combination of padding and tiling techniques. Padding parameters are obtained in a similar way to tiling ones. They are introduced in the CMEs and a GA is used to find near-optimal solutions. Details can be found in [12].

Table II shows the replacement miss ratios obtained for those kernels where the replacement miss ratio is still high after loop tiling. Column 2 shows the original replacement miss ratio; column 3 lists the replacement miss ratio after padding; and column 4 shows the replacement miss ratio after padding and tiling are applied sequentially in this order. In the future we plan to study the application of padding and tiling techniques in a single step, trying to find the padding and tiling parameters at the same time. This can in general produce better results than optimizing each part separately. The replacement miss ratios after padding and tiling are practically null for all benchmarks.

Table III shows the percentage of kernels (not considering those that appear in table II), which have a replacement miss ratio lower than 1%, 2% and 5% respectively after tiling. For all these kernels and

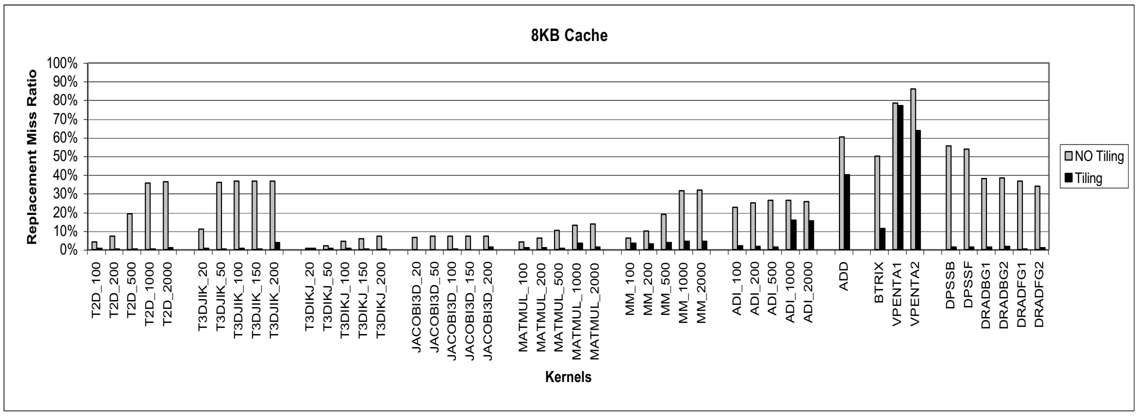

Fig. 7. Replacement miss ratio before and after loop tiling for a 8KB cache.

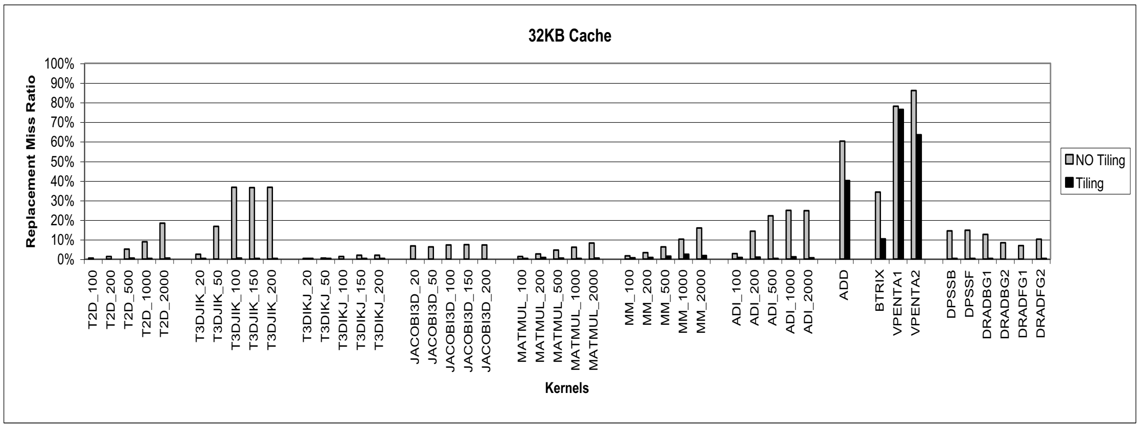

Fig. 8. Replacement miss ratio before and after loop tiling for a 32KB cache.

all cache sizes the replacement miss ratios are lower than 5%.

V. Conclusions

Cache memory performance is critical for the efficient execution of numerical applications. Loop tiling is a program transformation that reduce capacity misses. In this work, we have proposed the combination of genetic algorithms and a very accurate model of the cache behavior in order to perform near-optimal loop tiling. The cache model is based on a very fast solver of the Cache Miss Equations. The proposed technique can deal with any perfectly nested loop.

The evaluation of the proposed technique shows that the resulting tiling significantly reduces the miss ratio. For instance, for a 8KB direct-mapped cache, we can reduce the replacement miss ratio of the 3D matrix transpostion (N=100) from 36.7% to 0.6% and the replacement miss ratio of the Dpssb kernel from 55.5% to 1.25%. We have shown that the proposed loop tiling technique practically removes all capacity misses for all the loops that have been analyzed.

Acknowledgements

This work has been supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIC2001-0995-C02-01 and the ESPRIT Project MHAOTEU (EP24942).

Referencias

[1] Susan L.Graham David F.Bacon and Olivier J.Sharp. Compiler transformations for high-performance computing. Technical report, University of California, 1994.

[2] Somnath Ghosh, Margaret Martonosi, and Sharad Malik. Cache miss equations: an analytical representation of cache misses. In *ICS97*, 1997.

[3] Nerina Bermudo, Xavier Vera, Antonio González, and Josep Llosa. An efficient solver for cache miss equations. In *IEEE ISPASS*, 2000.

[4] Xavier Vera, Josep Llosa, Antonio González, and Carlos Ciuraneta. A fast implementation of cache miss equations. In *8th International Workshop on Compilers for Parallel Computers (CPC)*, January 2000.

[5] Michael E. Wolf and Monica S. Lam. A data locality optimizing algorithm. In *ACM SIGPLAN91*, 1991.

[6] Glover and Laguna. *Tabu search.* Kluwer, 1997.

[7] Kirkpatrick, Gelatt, and Vecchi. Optimization by simulated annealing. *Science 220*, 1983.

[8] D.E. Goldberg. *Genetic algorithms in search, optimizations and machine learning.* Addison-Wesley, 1989.

[9] David Padua et al. *Polaris developer's document*, 1994.

[10] Eduard Ayguadé et al. A uniform internal representation for high-level and instruction-level transformations. Technical Report UPC-DAC-95-02, Universitat Politècnica de Catalunya, 1995.

[11] J.Abella, A.González, J.Llosa, and X.Vera. Near-optimal loop tiling by means of cache miss equations and genetic algorithms. Technical Report UPC-DAC-2001-20, Universitat Politècnica de Catalunya, 2001.

[12] Xavier Vera, Antonio González, and Josep Llosa. Near-optimal padding for removing conflict misses. Technical Report UPC-DAC-2000-71, Universitat Politècnica de Catalunya, November 2000.