Polyhedral Modeling and Analysis of Memory Access Profiles

Philippe Clauss and Bénédicte Kenmei
ICPS/LSIIT, Université Louis Pasteur, Strasbourg
Pôle API, Bd Sébastien Brant
67400 Illkirch - France
{clauss, kenmei}@icps.u-strasbg.fr

Abstract

In this paper, we propose to model memory access profile information as loop nests exhibiting useful characteristics on the memory behavior, such as periodicity, linearly linked memory access patterns and repetitions. It is shown that static analysis methods as the polytope model approach can then apply onto the generated nested-loop representations. Moreover, the modeling loop nests can themselves be instrumented and run in order to generate further useful information that can also be modeled and analyzed.

1 Introduction

Managing the memory behavior of a program is crucial in reaching good performance, real-time constraints compliance, low system cost, or low energy consumption. But with the ever growing complexity of computer architectures, it is getting more and more difficult to ensure a deterministic execution behavior. Also developers use complex data and control structures, dynamic memory allocation, that make static analysis of the source code inadequate. Dynamic analysis approaches have been proposed in recent works that try to extract general behavior properties of a program either by analyzing the program during its execution, or by analyzing traces resulting from code instrumentation. Most of the proposed approaches focus on the computation of statistical quantities or coarse grain hot spots detections [8].

In this work, we use a behavior modeling approach based on a linear and periodic interpolation of a program execution trace presented in [4]. It considers traces as lists of memory addresses accessed during some program runs. Such lists can then be considered as time-series whose contained values are associated to a chronological time index. Our modeling approach allows to represent successive memory accesses as iterations of a sequence of loop nests whose indices define a multi-dimensional time-space. Each loop is associated to a program phase where memory accesses match access patterns linked by a function over the loop indices. From this representation of the program memory behavior, much analysis information can be computed and graphically represented by using the well-known *Polytope Model* of nested loops [6]. Hence the polytope model, classically used for static analysis of "Fortran-like" loop nests in programs, is used here for the analysis of the memory access profile of a program. It allows some representative visualizations and computations of some memory behavior characteristics. Graphic representations are built using multi-dimensional polyhedra and thus facilitating the understanding of a program memory behavior.

At our knowledge, the closest previous related work is the FORAY-GEN approach presented in [7]. Pointer access profiles are used as input to generate loops accessing arrays through affine access functions. However, their approach is less general than ours due to three main reasons. First, only affine access functions are generated, and memory accesses not following such a behavior are evicted. Second, the depth of the generated loop nest is predetermined by inserting some checkpoints between loop instructions in the source code. Third, we are able to identify successive phases of accessed memory locations each being modeled differently by different loops of the same depth. The FORAY-GEN approach does not handle such a modeling.

This paper is organized as follows. The periodic-linear model, already presented in [4], is recalled in the next section. It is shown in section 3 how this model can be advantageously used to analyze the memory profile of a program. Conclusions and perspectives are given in section 4.

2 Periodic-linear interpolation

Periodic interpolation consists in interpolating a sequence of values by a periodic polynomial function. In this work, we only consider periodic-linear functions. A periodic-linear function f is a function of the form $f(x) = ax + b$ where a and b are periodic numbers. A periodic number is a finite list of n numerical values $[a_1, a_2, ..., a_n]_y$ where the rank of the selected value at a given time to evaluate f is given by $y\ mod\ n$, $y \in Z$:

$$f(x) = ax + b = [a_1, a_2, ..., a_n]_y x + b$$

$$= \begin{cases} a_1 x + b, & \text{if } y\ mod\ n = 0 \\ a_2 x + b, & \text{if } y\ mod\ n = 1 \\ ... & ... \\ a_n x + b, & \text{if } y\ mod\ n = n-1 \end{cases}$$

Notice that since b is also a periodic number of m values b_1,b_2,...,b_m, f is also defined depending on $y\ mod\ m$. The number of values of a periodic number is called the *period*.

A periodic-linear interpolation of time-series links non-overlapping successive sub-sequences of values, or patterns, such that any element in pattern i at position j, e_{ij}, is linearly dependent of $e_{i-1,j}$: $e_{ij} = e_{i-1,j} + a_j$, where a_j is constant. The number of elements in each pattern is the lowest common multiple of both periods of the periodic coefficients a and b in the interpolation function f.

With each pattern is associated a date t defining the time space of the model. All patterns are modeled by a periodic linear function $f(t) = at + b$ where a and b are periodic numbers. These periodic numbers can have a large period p and therefore constitute by themselves new time-series. Hence we recursively apply our periodic-linear model to these new traces, i.e., to both periodic numbers, yielding an additional time dimension. Finally the whole application of the model yields a multi-dimensional time space $(t_1, t_2, ..., t_d)$.

This multi-dimensional time model can be fully represented as a loop nest of depth d of the general form shown in figure 1, where the instruction of the innermost loop serves to output the element value associated to a time instant $(t_1, t_2, ..., t_d)$. Function $f(t_1, t_2, ..., t_d)$ is linear relatively to each variable t_i and globally non-linear. In this work, we only consider such functions as being linear functions of the loop indices. If d is maximum, i.e., the model has been applied as far as possible, then the function is no longer periodic, since any period associated to a time dimension t_i is now expressed as a loop index.

Such a periodic-linear interpolation does not generally occur on the whole trace. A program behavior is generally characterized by successive program phases of different behaviors. In our model, we define *phases* as the largest adjacent slices of the trace allowing periodic-linear interpolations of their elements. Hence successive phases can occur at different depth levels yielding a hierarchy of phases. These can be represented as successive loops whose loop indices range from the first to the last element of each phase, and where each loop contains itself successive loops associated to inner level phases and so on.

We implemented our approach as a software taking as input a list of integer values and generating as output a sequence of loop nests modeling the input.

```
for $t_1 = 0$ to $p_1 - 1$
  for $t_2 = l(t_1)$ to $p_2 - 1$
    for $t_3 = l(t_1, t_2)$ to $p_3 - 1$
      ...
        for $t_d = l(t_1, t_2, ..., t_{d-1})$ to $p_d - 1$
          $f(t_1, t_2, ..., t_d)$ ;
```

Figure 1. Loop nest representing the multi-dimensional time model

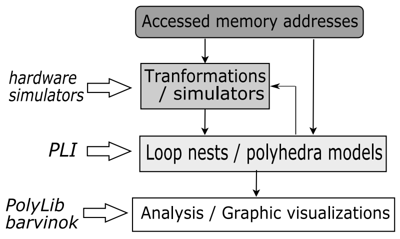
Accessed memory addresses
hardware simulators
Tranformations / simulators
PLI
Loop nests / polyhedra models
PolyLib barvinok
Analysis / Graphic visualizations

Figure 2. the analysis software environment.

3 Visualizing and computing memory behavior characteristics

We use three main software tools that are all freely available : *PLI:* our periodic-linear interpolation program; *Polylib* [3]: a library implementing many functions for polyhedra computations into the space of rational numbers; *barvinok* [1]: a program for computing the number of integer points in a parametrized polyhedron which is the Ehrhart polynomial of the parametrized polyhedron [5].

The visualizations and computations based on the polytope model and presented in the next subsections take part of a general analysis software environment that is currently under development and schematized on figure 2.

3.1 Memory access phases

In the generated loops, the indices define a multi-dimensional time-space in which the lexicographic order defines the access order of the memory addresses. This representation exhibits explicitly the memory access behavior of the analyzed memory profile while making apparent the following characteristics: the nested repetitions of linearly linked and nested memory access patterns and the number of such repetitions ; the periodicity of the occurring memory accesses that are linearly linked, which is given by the ranges of the loop indices ; the linear relations between memory addresses whose accesses are equally spaced, which is given by the functions of the innermost loops ; the sequence of nested phases characterized by the above properties with is given by the consecutive loop nests. All the generated loop nests can be represented as

```
for (k = 0 ; k < IMAGEDIM ; k++) {
 for (f = 0 ; f < IMAGEDIM ; f++) {
  pcoeff = &coefficients[0];
  parray = &array[k*ARRAYDIM + f];
  parray2 = parray+ARRAYDIM;
  parray3 = parray+ARRAYDIM+ARRAYDIM;
  *poutput = 0 ;
  for (i = 0 ; i < 3 ; i++)
   *poutput += *pcoeff++ * *parray++;
  for (i = 0 ; i < 3 ; i++)
   *poutput += *pcoeff++ * *parray2++;
  for (i = 0 ; i < 3 ; i++) {
   *poutput += *pcoeff++ * *parray3++;
  poutput++; } }
```

Figure 3. Main loop of program `fir2dim`.

```
$offset = 530832;$
for $t_1$ = 0 to 99
 for $t_2$ = 0 to 49 {
  for $t_3$ = 0 to 2 {
   for $t_4$ = 0 to 1
    accessed_block = $51t_1 + t_2 + 51t_3 + offset;$
   for $t_4$ = 2 to 2
    accessed_block = $51t_1 + t_2 + 51t_3 + 1 + offset;$ }
  for $t_3$ = 3 to 5 {
   for $t_4$ = 0 to 1
    accessed_block = $51t_1 + t_2 + 51t_3 - 153 + offset;$
   for $t_4$ = 2 to 2
    accessed_block = $51t_1 + t_2 + 51t_3 - 152 + offset;$ }}
```

Figure 4. Loop nests representing accesses to memory blocks.

polytopes. Each polytope is then associated to a phase of the memory access behavior. In each phase, linearly dependent memory address patterns are accessed.

Example 1 Consider the benchmark program `fir2dim` from the DSPStone suite [10], which performs a finite impulse response filter on a 2D image. The main loop of the program is shown on figure 3. We instrument the code with `IMAGEDIM = ARRAYDIM - 2` = 100 and get all the values taken by the pointers `parray`, `parray2` and `parray3`. Since the accessed objects are 4-bytes integers, we divide each value by 8 to get references to 32-bytes memory blocks. We finally consider as input a trace of 90000 successive accesses to memory blocks.

Our periodic-linear model yields the loop nests shown on figure 4 and representing exactly the whole input trace. This loop nest program exhibits one main phase of 100 linearly linked patterns of 900 memory addresses (loop index t_1). Each of these patterns embeds one phase of 50 linked patterns of 18 values (loop index t_2), embedding themselves two phases each of 3 linked patterns of 3 values (loop index t_3). Finally these latter patterns embeds two phases, one of two linked values and one of one unique value (loop index t_4).

The iterations of dimensions t_1, t_2 and t_3 are represented as the integer points contained in two polytopes shown on figure 5. Each polytope represents a different block-access phase following our periodic-linear classification. Each phase is associated to a different loop indexed by t_3. Following the chosen representation granularity, any integer point represents 3 accesses to memory blocks, corresponding to 3 iterations of the t_4-loops. The access order is given by the lexicographic order of t_1, t_2 and t_3.

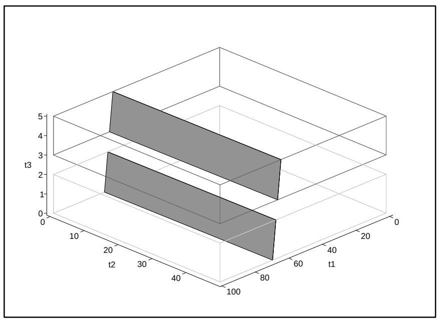

Figure 5. Polytopes representing two access phases to memory blocks and cutting planes of accesses to memory block 534332.

As other examples, it is shown in table 1 some nested loop models that were generated for the three Spec2000 benchmark programs `mcf`, `equake` and `ammp` using the `ref.in` input files. The loop nests represent the memory addresses successively accessed through pointers in the most time-consuming functions. Only the program `fir2dim` will be considered in the following analysis examples. □

3.2 Identifying accesses and access frequency

Integer points associated with accesses to a given address A are defined as points contained in the intersections between each of the previous polytopes and the hyperplanes defined by the equations stating that the innermost loops polynomials are equal to A. These sets are represented as hyperplanes cutting the polytopes.

Example 2 Let us identify all accesses to the memory block 534332. Those are integer points belonging to cutting planes defined by the equations $51t_1+t_2+51t_3+offset = 534332$ and $51t_1+t_2+51t_3+1+offset = 534332$. They are represented on figure 5. □

The access frequency to any address A can be defined as the number of integer points associated with accesses to any address A. It is also the number of integer points in the parametrized union of polytopes defined by intersecting the previous polytopes and the parametrized hyperplanes defined by the equations stating that the innermost loops polynomials are equal to any address A, where A is now considered as an unknown parameter. This number is expressed as several Ehrhart polynomials defined on disjoint intervals of A values. Those Ehrhart polynomials are computed using the program *barvinok* and the polyhedral library *Polylib*. This symbolic expression of the access frequency to any address A can then be used for symbolic analysis or for the generation through instantiation of several visual representations or computed information.

Program	Function	Model

mcf	price_out_impl	for $t_1 = 0$ to M
		for $t_2 = 0$ to N
		for $t_3 = 0$ to t_2
		$120t_2 - 120t_3 + offset$;

equake	smvp	for $t_1 = 0$ to $timesteps - 1$
		for $t_2 = 0$ to $N - 1$
		for $t_3 = 0$ to 2
		$128t_2 + 32t_3 + offset$;

ammp	a_number	for $t_1 = 0$ to 3598
		for $t_2 = 0$ to $t_1 + 1$
		$2224t_2 + offset$;
		for $t_1 = 3599$ to N
		for $t_2 = 0$ to $nb\_of\_atoms - 1$
		$2224t_2 + offset$;

Table 1. Nested loop models for three Spec2000 programs.

memory access indices of cache misses	loop indices values
$3600t_1 + 72t_2 - 943200$ $3600t_1 + 72t_2 - 943174$	$263 \leq t_1 \leq 286$ $0 \leq t_2 \leq 12$
$3600t_1 + 72t_2 - 943200$ $3600t_1 + 72t_2 - 943192$	$263 \leq t_1 \leq 286$ $13 \leq t_2 \leq 24$
$3600t_1 - 941400$ $3600t_1 - 941394$ $3600t_1 - 941338$	$263 \leq t_1 \leq 286$
$3600t_1 + 72t_2 - 948312$ $3600t_1 + 72t_2 - 948250$	$263 \leq t_1 \leq 286$ $97 \leq t_2 \leq 107$
$3600t_1 + 72t_2 - 948312$ $3600t_1 + 72t_2 - 948268$	$263 \leq t_1 \leq 286$ $108 \leq t_2 \leq 120$

Table 2. Memory accesses generating cache misses.

Example 3 The number of accesses to any memory block B is given by the counting of the number of integer points in a union of polytopes parametrized by B. The answer is given as a list of 16 Ehrhart polynomials defined on 16 adjacent intervals of B values. For example, all blocks B between 530935 and 530983 are accessed 18 times, excepting blocks B such that $B \ mod \ 51 = 23$ which are accessed 6 times and blocks B such that $B \ mod \ 51 = 24$ which are accessed 12 times. These polynomials allow to generate different graphic visualizations as an histogram of the block access frequencies for any value range of B. □

3.3 Analyzing the cache behavior

Accessed memory addresses are just one possible information characterizing memory accesses. Other information are relevant to memory accesses as the cache misses generated while accessing memory. Hence instead of considering a trace of successively accessed memory addresses, a trace of the number of cache misses measured after each memory access can also be interestingly considered. Moreover, the models generated through our periodic-linear approach can be used in a collaborative fashion in order to associate memory addresses to cache misses. Since the total number of values in both input traces are equal, both generated sequences of loop nests represent the same number of iterations, and the i^{th} memory access corresponds to i^{th} number of cache misses. Evaluations from one model to the other can be done through a loop indices conversion between both. Some further analysis can also be achieved by instrumenting the generated loop nests and running them in order to generate new traces. Those new traces are then also modeled to allow the computation of additional analysis information.

Example 4 We simulate a 32KB direct mapped cache of 1024 lines with the LRU policy and ask the cache simulator *DineroIV* [2] to output the total number of cache misses after each memory access. The resulting trace of cache misses is modeled by our tool as the loops shown on figure 6, representing the largest phase which englobes 96% of all memory accesses. The cache miss behavior can be represented graphically as polytopes whose integer points are associated to successive memory accesses, and where surfaces in gray color represent accesses generating cache misses (see figure 7). It can also be precisely computed from this loop nest what accesses are generating cache misses (see table 2). The distances between two consecutive accesses generating cache misses can also be deduced. The block addresses whose accesses are generating cache misses can also be known by transforming the memory access indices of table 2 as indices of the loops in figure 4: let I be any memory access index, the corresponding indices (t_1, t_2, t_3) for the loops representing the addresses of the accessed memory blocks can be computed in the following way:

$$I \longrightarrow \begin{cases} t_1 = \left\lfloor \frac{I}{900} \right\rfloor \\ t_2 = \left\lfloor \frac{I \ mod \ 900}{18} \right\rfloor \\ t_3 = \left\lfloor \frac{(I \ mod \ 900) \ mod \ 18}{3} \right\rfloor \\ t_4 = ((I \ mod \ 900) \ mod \ 18) \ mod \ 3 \end{cases}$$

For example, the 29186^{th} memory access, which is generating a cache miss following table 2, is an access to the memory block whose access indices in the loops of figure 4 are $(32, 21, 2, 2)$. Hence it corresponds to an access to the block whose address is $51 \times 32 + 21 + 51 \times 2 + 1 + 530832 = 532588$.

To know how many cache misses are generated by accessing a given memory block B, a direct analysis of the above loops and the use of conversion formulas yields too complex computations and answers, mainly due to their non-linearity. Another way is to instrument the loop nests of figure 6 to build a simulator and to run it, in order to generate a trace of all block addresses that generate cache misses. Then the modeling of this latter trace as loop nests allows to compute the number of cache misses generated by accessing any memory block B. For the simulator, we implement a conversion function that computes from a current value of the indices (t_1, t_2, t_3) the corresponding block address. This function uses table 2, the formulas giving the corresponding indices for the loops on figure 4 and the address functions of the innermost loops on the same figure. The generated trace is finally modeled as a loop nests.

```
for $t_1 = 263$ to 286 {
 for $t_2 = 0$ to 12 {
  for $t_3 = 0$ to 25
   #cache_misses = $51t_1 + t_2 - 13336$ ;
  for $t_3 = 26$ to 71
   #cache_misses = $51t_1 + t_2 - 13335$ ; }
 for $t_2 = 13$ to 24 {
  for $t_3 = 0$ to 7
   #cache_misses = $51t_1 + t_2 - 13336$ ;
  for $t_3 = 8$ to 71
   #cache_misses = $51t_1 + t_2 - 13335$ ; }
 for $t_2 = 25$ to 30
  #cache_misses = $51t_1 - 13311$ ;
 for $t_2 = 31$ to 86
  #cache_misses = $51t_1 - 13310$ ;
 for $t_2 = 87$ to 96
  #cache_misses = $51t_1 - 13309$ ;
 for $t_2 = 97$ to 107 {
  for $t_3 = 0$ to 61
   #cache_misses = $51t_1 + t_2 - 13406$ ;
  for $t_3 = 62$ to 71
   #cache_misses = $51t_1 + t_2 - 13405$ ; }
 for $t_2 = 108$ to 120 {
  for $t_3 = 0$ to 43
   #cache_misses = $51t_1 + t_2 - 13406$ ;
  for $t_3 = 44$ to 71
   #cache_misses = $51t_1 + t_2 - 13405$ ; } }
```

Figure 6. Loop nests representing the increase of the number of cache misses after each memory access between the 3601^{th} and the last.

The number of cache misses generated by accessing a given memory block B is given by the Ehrhart polynomials of the union of polytopes defined by the loop nests reduced to the case where the accessed block is the parameter B. The histogram of cache miss frequencies per block addresses between 531036 and 532000 is generated from these polynomials and is shown on figure 8. □

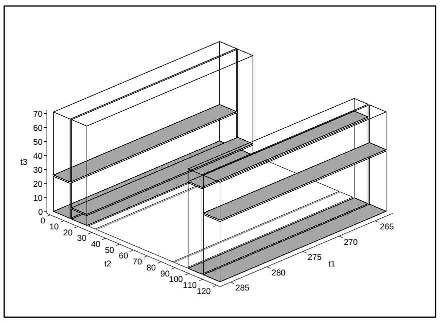

Figure 7. Polytopes representing the cache miss behavior.

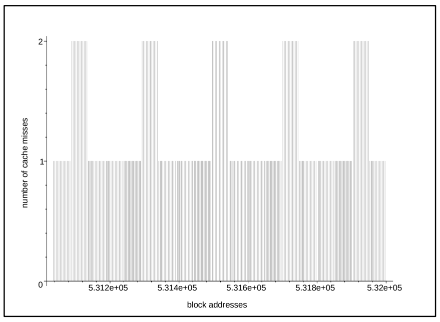

Figure 8. Cache miss frequencies for blocks between 531036 and 532000.

4 Conclusion

In this paper, it was shown that static analysis methods can be used for the analysis of execution-time information whether those are modeled themselves as programs. Generally, control and data structures in programming languages offer a wide range of expressive objects that can be used to model behavior information. Moreover the complexity of the analyzed behavior can be correlated with the complexity of the modeling program, as it is stated in the Kolmogorov complexity theory [9]. Those modeling programs can also be instrumented and executed to simulate certain execution environment or to generate deductible execution-time information. Our future objectives are to handle non-linear functions in the analysis framework and also to extend our modeling approach to non-linear periodic functions and a wider range of control structures that can be statically analyzed.

References

[1] *Barvinok*, a library for counting the number of integer points in parametrized and non-parametrized polytopes. http://freshmeat.net/projects/barvinok.

[2] Dinero iv trace-driven uniprocessor cache simulator. http://www.cs.wisc.edu/ markhill/DineroIV.

[3] The polyhedral library *polylib*. http://icps.u-strasbg.fr/PolyLib.

[4] P. Clauss, B. Kenmei, and J. C. Beyler. The periodic-linear model of program behavior capture. In *ACM/IEEE Euro-Par 2005*, volume 3648 of *LNCS*, pages 325–335. Springer, 2005.

[5] P. Clauss and V. Loechner. Parametric analysis of polyhedral iteration spaces. *Journal of VLSI Signal Processing*, 19(2):Kluwer Academic, 1998.

[6] P. Feautrier. *The Data Parallel Programming Model*, volume 1132 of *LNCS*, chapter Automatic Parallelization in the Polytope Model, pages 79–100. Springer-Verlag, 1996.

[7] I. Issenin and N. Dutt. Foray-gen: Automatic generation of affine functions for memory optimizations. In *DATE '05: Proc. of the conf. on Design, Automation and Test in Europe*, pages 808–813, Washington, DC, USA, 2005. IEEE Computer Society.

[8] J. Lau, S. Schoenmackers, and B. Calder. Transition phase classification and prediction. In *11th Int. Symp. on High Performance Computer Architecture*, February 2005.

[9] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, New York, 1993.

[10] V. Zivojnovic, J. M. Velarde, and C. Schlager. Dspstone: A dsp-oriented benchmarking methodology. In *Int. Conf. Signal Processing Applications and Technology*, october 1994.