Semantic Tiling

Guillaume Iooss, Sanjay Rajopadhye, Christophe Alias

Semantic tiling - Extended Abstract

Guillaume Iooss*, Sanjay Rajopadhye† and Christophe Alias‡

1 Introduction

Tiling is a well known program transformation, typically applied to loop programs, used to reorder the computation with many different goals: exploiting locality in the memory hierarchy, adapting the granularity of computation to data transfers, and/or exposing coarse-grain parallelism, to name just a few. The tiling transformation, $\mathcal{T} : \ddagger \mapsto \langle \ddagger_{\sqcup}, \ddagger_{\updownarrow} \rangle$ maps a point z in the iteration space to a tile z_t and to a local index z_l within a tile. The computation performed at the new, reordered iteration, $\langle z_t, z_l \rangle$ is *identical* to that performed at the original point z. We view such a tiling as a purely "syntactic" tiling. All previous research on tiling has adopted this view. In particular, for every dependence edge $z \to z'$ in the original computation, there is a corresponding edge $\mathcal{T}(z) \to \mathcal{T}(z')$ in the transformed computation.

In this paper, we explore an alternative view, called *semantic tiling*. It is first and foremost a tiling transformation, in the sense that we use a function $\mathcal{T}$ to map the original iterations to the transformed ones. However, it may not be true that the intermediate results computed by the transformed program are identical to those in the original. Alternatively, the transformation may make use of semantic properties of the computations in order to derive the tiled program. We illustrate our ideas with some examples from dense linear algebra, where "blocked" versions of the standard algorithms are well known in the mathematical literature.

Consider the *forward substitution* problem. Let A be a $n \times n$ be a lower triangular matrix and b a vector of size n. The conventional program that solves the equation $A.x = b$ uses the following steps:

(1) Compute $x_0 = b_0 / A_{0,0}$.

(2) Compute $x_i = \left(b_i - \sum_{j=0}^{i-1} A_{i,j} \times x_j \right) / A_{i,i}$, for $i = 1 \ldots n-1$,

Let us break the matrix A into $m \times m$ sub-matrices of size $B \times B$, and let us index it such that $\hat{A}[\alpha, \beta]$ is the $\langle \alpha, \beta \rangle$th submatrix of A. After transforming b into $\hat{b}$, the following program also solves the equation $A.x = b$ by manipulating matrices and vectors of size B:

(1) Compute $\hat{x}[0] = \text{ForwSubst}\Big(\hat{A}[0,0]\ ,\ \hat{b}[0]\Big)$.

(2) Compute, for $i = 1 \ldots m-1$,

$$\hat{x}[i] = \text{ForwSubst}\Big(\hat{A}[i,i], \Big(\hat{b}[i] \ominus \bigoplus_{j=0}^{i-1} \hat{A}[i,j] \otimes \hat{x}[j]\Big)\Big)$$

where $\ominus$ and $\otimes$ are respectively the vector subtraction and the matrix-vector multiplication, and $\bigoplus$ is vector addition "summation."

Notice that the structure of both program is the same, with the exception that the first one manipulates scalars and the second one matrices and vectors. Also, scalar operations are replaced by the corresponding blocked operations, and a recursive call to a scalar instance of "ForwSubst" is used instead of scalar division.

One class of *semantic tiling* is a transformation that takes the scalar version of a program as an input, and which returns such a blocked version. This blocked version has the same structure than the scalar version, however it manipulates matrix and vectors instead of scalars. The algebraic operators used are also transformed into their corresponding higher-order operators.

This program transformation exploits associativity and commutativity. In our example, we changed the order of summation in the accumulation, by regrouping the terms by groups of size B. Thus, we used the associativity of addition and the intermediate data computed by the blocked program may not be the same as in the scalar program.

Moreover, solving the problem at the block level, we need a recursive call to the scalar program on a smaller problem instance. Thus, we have a divide-and-conquer scheme in the semantic tiled version of a program.

*ENS Lyon, and Colorado State University, email: guillaume.iooss@ens-lyon.fr

†Colorado State University: Sanjay.Rajopadhye@colostate.edu

‡ENS Lyon, INRIA, email: christophe.alias@ens-lyon.fr

Semantic tiling can be viewed as a kind of *data tiling* [4] in which the tiles are manipulated as matrices, yielding the same locality benefits as a conventional tiling. As opposed to conventional tiling, semantic tiling does not consider dependences (and might modify them through semantic properties). Thus, semantic tiling may be effective on some example where the conventional tiling does not work [2].

However, its range of application is limited: indeed, we need the notions of matrices and vectors, so this forces an algebraic structure such as a s ring or a semiring on the underlying data (which is the case for linear algebra and graph theory).

We now present our early results about the formalization of semantic tiling, and the different issues raised.

2 Formalizing semantic tiling

We are exploring several approaches to formalize *semantic tiling*.

- The first approach is to find a set of rewriting rules to transform a scalar program into its semantic tiled version. These rules are based on algebraic properties and ensure the correctness of the derived semantic tiled program.
- The alternative is to hypothesize a form of the semantic tiled program from the structure of the scalar program, then to prove its validity by checking the equivalence of both programs.

We have investigated the second approach. We decompose the problem into 3 subproblems:

- Deriving a tiling of the input and output data, then of the temporary data.
- Deriving the semantic tiled program from the scalar version.
- Deriving the equivalence between the guessed program and the original scalar program.

The main issue in the data tiling step is to set the tiling to make matrix operations coherent (e.g., ensuring that we do not multiply two matrices of incompatible size). Moreover, some properties on the inputs might force us to constraint this tiling even more (e.g, if a matrix is lower triangular, we need to have square tiles).

On the second problem, one issue is the *higher-order operator selection*: we might have several candidates for the same scalar operator. For example, when the inputs to a scalar program are transformed to vectors, a scalar multiplication may transform into either a scalar product, or a convolution in the blocked program. A similar issue arises when determining which operators in the scalar program to replace by a recursion.

The problem of *equivalence of two programs* was studied by Barthou and al. [1]. In their work, they managed to reduce the program equivalence problem into a problem of reachability on a *Memory State Automaton*. This problem is undecidable in general, but several heuristics exist for it. However, they do not consider semantic properties of operators, which are needed on our case. We integrate the commutativity/associativity properties over a parametric number of terms [3]. These properties allow us to cover many equivalences in examples drawn from dense linear algebra.

3 Conclusion

Semantic tiling is a generalization of conventional tiling where we exploit semantic properties such as associativity, commutativity, and other algebraic in order to systematically tile programs in ways that is not possible with purely "syntactic tiling." We have applied it successfully by hand on several examples, including LU decomposition, Cholesky, forward substitution and sub-problems of APP (such as shortest path on a graph). Semantic tiling might also be useful for high-level hardware synthesis. Indeed, it can be used to get operations of bigger-grain, which allows a better optimization of the data path.

References

[1] Denis Barthou, Paul Feautrier, and Xavier Redon. On the equivalence of two systems of affine recurrence equations. In *Proceedings of the 8th International Euro-Par Conference on Parallel Processing*, Euro-Par '02, pages 309–313. Springer-Verlag, 2002.

[2] T. Risset D. Cachera, S. Rajopadhye and C. Tadonki. Parallelization of the Algebraic Path Problem on Linear SIMD/SPMD Arrays. Technical Report 1346, IRISA, 2000.

[3] Guillaume Iooss, Christophe Alias, and Sanjay Rajopadhye. On program equivalence with reductions. submitted.

[4] Induprakas Kodukula, Nawaaz Ahmed, and Keshav Pingali. Data-centric multi-level blocking. *SIGPLAN Not.*, 32(5):346–357, May 1997.