Mono-parametric Tiling is a Polyhedral Transformation

Guillaume Iooss, Sanjay Rajopadhye, Christophe Alias, Yun Zou

RESEARCH REPORT
N° 8802
October 2015
Project-Team Roma

ISSN 0249-6399   ISRN INRIA/RR--8802--FR+ENG

Mono-parametric Tiling is a Polyhedral Transformation

Guillaume Iooss[*], Sanjay Rajopadhye[†], Christophe Alias[‡], Yun Zou[§]

Project-Team Roma

Research Report n° 8802 — October 2015 — 37 pages

Abstract: Tiling is a crucial program transformation with many benefits: it improves locality, exposes parallelism, allows for adjusting the ops-to-bytes balance of codes, and can be applied at multiple levels. Allowing tile sizes to be symbolic parameters at compile time has many benefits, including efficient auto-tuning, and run-time adaptability to system variations. For polyhedral programs, parametric tiling in its full generality is known to be non-linear, breaking the mathematical closure properties of the polyhedral model. Most compilation tools therefore either avoid it by only performing fixed size tiling, or apply it in only the final, code generation step. Both strategies have limitations.
We first introduce mono-parametric partitioning, a restricted parametric, tiling-like transformation which can be used to express a tiling. We show that, despite being parametric, it is a polyhedral transformation. We first prove that applying mono-parametric partitioning (i) to a polyhedron yields a union of polyhedra, and (ii) to an affine function produces a piecewise-affine function. We then use these properties to show how to partition an entire polyhedral program, including one with reductions. Next, we generalize this transformation to tiles with arbitrary tile shapes that can tesselate the iteration space (e.g., hexagonal, trapezoidal, etc). We show how mono-parametric tiling can be applied at multiple levels, and enables a wide range of polyhedral analyses and transformations to be applied.

Key-words: Automatic parallelization, polyhedral compilation, tiling

[*] Inria/ENS-Lyon/UCBL/CNRS/Colorado State University
[†] Colorado State University
[‡] Inria/ENS-Lyon/UCBL/CNRS
[§] Colorado State University

Le tuilage mono-paramétrique est une transformation polyédrique

Résumé : Le tuilage est une transformation de programme importante avec de nombreux avantages: il améliore la localité du programme, expose du parallélisme, permet d'ajuster le ratio calcul-communications et peut être appliqué à de multiples niveaux. Avoir des tailles de tuiles paramétrées pendant la compilation est également intéressant: cela permet d'autotuner efficacement le code et de pouvoir s'adapter pendant l'exécution aux changements du système. Dans le contexte des programmes polyédriques, le tuilage paramétrique est connue pour ne pas être linéaire en général, ce qui brise les propriétés mathématiques de clôture du modèle polyédrique. La plupart des compilateurs polyédriques évitent ce problème en n'appliquant que des tuilages à taille fixée, ou en appliquant cette transformation uniquement à la dernière étape, la génération de code. Chaque stratégie a ses limitations.

Nous introduisons tout d'abord le partitionnement mono-paramétrique, une transformation similaire à un tuilage dont l'aspect paramétrique est restreint, qui peut être utilisé pour définir un tuilage. Nous montrons que, malgré la paramétrisation des tailles de tuiles, cette transformation est polyédrique. Tout d'abord, nous prouvons qu'appliquer le partitionnement mono-paramétrique (i) à un polyèdre produit une union de polyèdres, et (ii) à une fonction affine produit une fonction affine par morceaux. Nous utilisons ensuite ces propriétés pour montrer comment partitionner un programme polyédrique complet, potentiellement contenant des réductions. Ensuite, nous généralisons cette transformation à des formes de tuiles arbitraires qui pavent l'espace d'itération (par exemple, hexagone, trapézoïde, etc). Nous montrons également comment le tuilage mono-paramétrique peut être appliqué à de multiples niveaux et permet l'application d'analyses et de transformations polyédriques.

Mots-clés : Parallélisation automatique, compilation polyédrique, tuilage

1 Introduction

Iteration space tiling [46, 25, 41, 45, 37] is a very important program transformation, and offers a number of benefits. It can be used to improve data locality, to expose parallelism, and to allow for adjusting the granularity or *balance* [9, 48], i.e., the ratio of operations to data (also called the arithmetic or operational intensity [43], that a segment of code performs). It can be applied at multiple levels. Tiling is often applied manually, but it is an even more critical transformation in most compiler tools, especially those that tackle the class of data- and compute-intensive programs that can exploit the mathematical foundations of the *polyhedral model* [36, 35, 15, 17].

Allowing tile sizes to be symbolic parameters at compile time has many benefits, including efficient autotuning, and run-time adaptability to system variations. For polyhedral programs, parametric tiling in its full generality is known to be non-linear, breaking the mathematical closure properties of the model. Most polyhedral compilation tools therefore, either avoid it by only performing fixed size tiling, or apply it in only the final, code generation step. Both strategies have limitations.

Limitations of fixed size tiling The very purpose of tiling is to improve performance, and the tile size(s) have a huge influence on performance. However, the exact nature of how tile sizes affect performance is very poorly understood, so a widely accepted strategy is through autotuning or iterative compilation [27, 28] by searching through or empirically evaluating the space of different compiler parameters (notably tile sizes) and chosing the best one. The search may be guided by cost models, and strategies like machine learning may be used to improve the quality of the results, but the key element is the repeated evaluation of the performance of a number of target programs with different tunable parameters, including tile size. With fixed size tiling, *a separate program must be generated and compiled for each tile size*, and this is very expensive. Furthermore, the best tile sizes may depend on parameters that are not known at compile time, e.g., the input sizes. In such cases, many alternate versions, e.g., the best one for each of a range of input sizes, must be pre-generated and compiled into the final executable, leading to possible code size bloat.

Current limitations of parametric tiling Because parametric tiling takes a program outside the polyhedral model, we cannot subsequently apply any other polyhedral transformation and analysis. Hence, current polyhedral compilation tools apply parametric tiling only at the final, code generation phase. As a result, many decisions about the parallelization and/or optimization strategy are "hardwired" into the code generator, thereby negating one of the main advantages of the polyhedral model—composability of program transformations.

A simple example is that when parallel tiled code is generated by current parametric tiled code generators they all choose the standard "45° wavefront" parallelization[1] among the tiles. While this known to be a legal choice provided tile sizes are large enough [25], it is certainly not the only one. For example, in some situations it may be legal and preferable to use a canonic vector—one of the tile loops may not carry any dependences and could be simply declared to be parallel with a pragma in the generated code. However, in order to incorporate this "simple decision" two very different code patterns must be generated, as in the GPU compiler developed by Konstantinidis et. al [29].

In this paper, we propose a middle ground called mono-parametric tiling: a restricted parametric tiling where all tile sizes are multiples of a single parameter. We first show (Sec. 4) the somewhat unintuitive result that a tiling-like transformation, called *monoparametric partitioning*, where all tile sizes are multiples of a single parameter and the "tiling" is along the canonic axes, is a polyhedral transformation. In particular, applying it (i) to a polyhedron yields a union of polyhedra, and (ii) to an affine function produces a piecewise-affine function. Then, in Sec. 5, we show how to exploit these properties

[1]This corresponds, mathematically, to using a wavefront hyperplane whose normal vector is $\vec{1}$ as the schedule.

to monoparametrically partition an entire polyhedral program, given a partitioning of a subset of data arrays and statements, and also how to handle non-canonic directions. This includes: consistently combining potentially different partitionings of subexpressions, and handling programs with reductions. Next, (Sec. 6) we generalize to partitions of arbitrary polyhedral shapes that tesselate the iteration space, including hexagonal, trapezoidal, etc. We illustrate our results by showing how mono-parametric partitioning can be applied at multiple levels, and enables a wide range of polyhedral analyses and transformations (Sec. 7). We also describe how tiling is a simple extension of partitioning and give a method to check the legality of a proposed mono-parametric tiling.

2 Related Work

Tiling has been an active research research topic for nearly thirty years [46, 25, 41, 45, 37] for both polyhedral and more general programs. Although its importantce for polyhedral programs was well known, it was difficult to incorporate into fully automatic compilation tools, despite early efforts [44, 2, 1]. Early work on scheduling polyhedral programs was mostly focused on optimality under unbounded resource assumptions [17, 18, 10, 11], and an algorithm to optimally choose tile shapes was only developed reletavely recently [8].

Many authors worked on applying tiling optimizations on different programs, especially stencil computations, an important class of programs that occurs in a variety of scientific applications, and for which tiling is often difficult. For this reason, much of the techniques discussed in this paper are related to stencils, although our approach is of course, applicable to arbitrary polyhedral programs.

Dursun et at. [13] and Peng et al.[33] apply tiling on a subset of the computation dimensions (data dimensions) for stencil computations, and benefits are showed on multiprocessors. However, the amount of cache reuse is very limited, and tiling all the dimensions is necessary to maximize data reuse.

For stencil computations, due to the cross dependences in the time dimension, rectangular tiling cannot be applied directly. Time skewing [45, 48, 47, 3] is an important approach to exploit data locality across all dimensions. It looks at the dependences in the whole iteration domain, skews it with respect to the time dimension, then divides it into rectangular tiles. Due to dependences between tiles, it is also necessary to subsequently parallelize the tiled program, say with the classic 45° wavefront parallelization.

Krishnamoorthy et al. [30] pointed out that the classic wavefront parallelization suffers from the pipeline fill-flush overhead, and proposed split tiling and overlapped tiling techniques to enable concurrent start. Concurrent start can also be achieved with different tile shapes, such as diamond and hexagonal. Strzodka presents a technique called cache accurate time skewing (CATS) [42] for stencil computations. CATS tiles a subset of the dimensions to form large diamond tiles, and a sequential wavefront traversal is performed inside the tiles and the parallelism is explored among the tiles. Grosser et al. [20] presented a hybrid hexagonal tiling technique for GPUs, which tiles the face constructed by the time dimension and one data dimension with hexagonal tiles, and all the other dimensions with parallelogram shaped tiles.

Due to the complexity of the tiled code, a code generator is usually implemented when a tiling strategy is presented. Depending on whether tile sizes are left as tunable program parameters in the generated code, the code generators may be either fixed sized or parametric. When tile sizes are fixed (not as tunable program parameters), tiling can be described as a linear transformation, and purely polyhedral code generators as developed by Quilleré et al. [34], Bastoul et al. [6] are adequate, otherwise parametric code generation is necessary.

Since parametric tiling is a non-linear transformation, the resulting program is no longer polyhedral. Existing parametric tiled code generators apply rectangular tiling, possibly after a preprocessing step, and generate codes by purely syntactic manipulation of the AST [39, 40]. Given a d-dimensional loop nest, these tools generate a d-dimensional tiled loop nest that visits all the tile origins/coordinates, and a d-dimensional point loop nest that visits all the points for a visited tile. A straightforward solution is to construct the bounding box of the iteration domain [49], and generate the tile-loops that enumerate the tile origins in the bounding box and perform proper checking for whether a tile contains any valid iteration points. However, the bounding box strategy can end up with enumerating many empty tiles.

In order to avoid the enumeration of empty tiles, PrimeTile [22] tiles the original iteration space one dimension at a time. At each one, it identifies the largest sub-rectangular space formed by the current dimension and the previous dimensions, to which it then then applies rectangular tiling. The sub-rectangular space identification code is generated and executed at run-time, and this may incur run-time overhead. DynTile [23], DTiler [26] and PTile [5] avoid enumeration of empty tiles by constructing an *outset*, which only includes non-empty tile origins. Wavefront parallelization is supported in most of the parametric tiled code generators. DynTile [23] achieves wavefront parallelization through a run-time scheduler, while DTiler [26] and PTile [5] statically constructs the schedule for the classic 45° wavefront parallelization.

Authors have also explored concurrent start for parallelization, such as diamond tiling [42], hexagonal tiling [21, 20], and split and overlapped tiling [30, 24]. However, all the accompanying code generators use fixed size tiling. Generating parametric tiled codes for these shapes is known to be a hard problem. Recently Bertolacci et al. [7], described the loop structure for parametric diamond with a single parameter, but they did not resolve the automatic code generation problem.

In all this work, solutions to many problems that should remain separate—choice of tile shapes, and for each one, choice of parallelization, including communication and data locality, across and within tiles—are all combined in an ad hoc manner. Our goal is to develop a single framework to enable a separation of concerns and allow solutions to be developed modularly. The price of this is a restricted parametric tiling (i.e., mono-parametric tiling)

Finally, a word about our notation. In this paper, we use the term *partitioning* to describe only a *reindexing* transformation that maps a d-dimensional space to a $2d$-dimensional one. It does not, in anay way, alter the program execution order or semantics. This allows us to focus on the mathematical details, and defer the discussion of legality of the transformation. It also enables more sophisticated analyses, as we shall see.

3 Background

The ***polyhedral model*** [36, 35, 15, 17] is a mathematical formalism for analyzing, parallelizing and transforming an important class of compute- and data-intensive programs, or program fragments, called *Affine Control Loops (ACL)*. An ACL satisfies the following properties: (i) it only consists of (sequences of) arbitrarily nested loops, (ii) the statements in the loop are assignment statements, possibly with conditions; (iii) the loop lower (respectively, upper) bounds are the maxima (respectively, minima) of a finite number of affine functions of outer loop indices and program parameters, and (iv) the conditions and array accesses are also affine functions of outer loop indices and program parameters. The following terms are usually used in describing a polyhedral program:

Iteration vector. Each statement in an ACL is surrounded by zero or more loops, and an iteration vector is used to represent the *dynamic instances* of a statement. An iteration vector is described as an n-entry column vector $\vec{i} = (i_0, i_1, i_2, \dots, i_{n-1})^T$, where n is the number of loops surrounding the statement (called the *depth* of the statement), and i_k is the loop index of the kth loop (i_0 being the outermost). In an ACL with s size parameters, the s-entry column vector $\vec{p}$ with these parameters is called the *parameter vector* of the ACL.

Definition 3.1. *The **domain**, $\mathcal{D}$, of a statement describes the set of legal values that its iteration vectors can take. Because of the constrains of ALCs, the iteration space is represented by a set of linear inequalities, and hence is a polyhedron, parametrized by the size parameters:*

$$\mathcal{D} = \left\{ \vec{i} \mid \vec{i} \in \mathbb{Z}^n, Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q} \geq 0 \right\}$$

where $\vec{i}$ and $\vec{p}$ are, respectively, the iteration and parameter vector, Q and $Q^{(p)}$ are constant matrices with size $m \times n$ and $m \times s$ respectively, and $\vec{q}$ is an m-entry constant vector, where the number of constraints defining the polyhedron is m.

Definition 3.2. ***Flow Dependence**: A statement instance $\langle S_1, \vec{i}\rangle$* depends on *the instance $\langle S_2, \vec{j}\rangle$, written as $\langle S_1, \vec{i}\rangle \to \langle S_2, \vec{j}\rangle$ if the value produced by $\langle S_2, \vec{j}\rangle$ is used by $\langle S_1, \vec{i}\rangle$.*

Domains and affine functions are the critical elements of all polyhedral compilation tools.

Definition 3.3. *Given a vector of sizes, $\vec{b} \in \mathcal{Z}^n$, a **canonic partitioning** transformation, $\mathcal{T}_{\vec{b}}$ maps an index point $\vec{i} \in \mathcal{Z}^n$ to a point, $\langle \vec{i_b}, \vec{i_l}\rangle \in \mathcal{Z}^{2n}$, where $\vec{i_b} = \left\lfloor \frac{\vec{i}}{\vec{b}} \right\rfloor$, and $\vec{i_l} = \vec{i} \bmod \vec{b}$, and we have extended the division, modulo and floor operation elementwise to vectors.*

The inverse, $\mathcal{T}_{\vec{b}}^{-1}$ of a canonic partitioning is given by, $\mathcal{T}_{\vec{b}}^{-1}(\langle \vec{i_b}, \vec{i_l}\rangle) = \vec{i_b}.\vec{b} + \vec{i_l}$, where the product and sum are elementwise.

If $\vec{b}$ is a constant vector, the canonic partitioning is said to be fixed size*, if the elements of $\vec{b}$ are all symbolic, it is a* parametric *partitioning, and if all elements are multiples of a single parameter, it is a* monoparametric *partitioning.*

For example, a monoparametric partitioning can have rectangles of size $b \times 2b$, but not $b \times b'$ where b and b' are two *different* parameters. This means that we fix the shape of a partition, and we can only change its size by homothetically scaling it by a single size parameter b. When the context is clear, we may drop the word "canonic," and use "blocking" or "tiling" for partitioning.

(Canonically) partitioning a polyhedron $\mathcal{D}$ means applying a canonic partitioning transformation (Def. 3.3) to it. Note that even though the notation uses floor and modulo operations, it is well known that the image of $\mathcal{D}$ by a fixed size partitioning is a polyhedron. However, the image by a parametric partitioning is not.

Program Representation and Reductions In his seminal paper, Feautrier [15] showed that, for any array reference on the right hand side of any statement in an ACL, the corresponding flow dependence can be determined *exactly* and *symbolically*, in the sense that for every flow dependence $\langle S_1, \vec{i}\rangle \to \langle S_2, \vec{j}\rangle$, $\langle S_2, \vec{j}\rangle$, can be expressed as a piece-wise affine function of $\vec{i}$ and $\vec{p}$. Hence we can translate the ACL to the following polyhedral program representation: a program is abstracted as a set of computations of statement instances, each of which is described as follows:

$$\vec{i} \in \mathcal{D} : \langle S, \vec{i}\rangle = g(\langle S_1, f_1(\vec{i})\rangle, \ldots, \langle S_d, f_d(\vec{i})\rangle)$$

where $\vec{i}$ is the iteration vector for statement S, $\vec{D}$ is a subset of the domain for statement S, the function g takes d arguments and each argument is a result of a statement, and for $k = 0 \ldots d$, f_k is a dependence function. This representation is often described as a *Polyhedral Reduced Dependence Graph* (PRDG). Program inputs are represented as special "dummy statements" which are sink nodes in the PRDG.

It is well known that *reductions*, *scans* or *prefix computations* are very powerful programming and computational abstractions. They can be specified as the application of associative (and often commutative) operators to (collections of) values, producing (collections of) values. Redon and Feautrier showed [38] that for ACLs, after obtaining flow dependences as piece-wise affine functions, reductions and scans also also be detected relatively easily. Since monoparametric tiling can be applied to such programs to and provides some significant benefits, we extend our program representation/PRDG to include reductions too.

A reduction is represented with two statements: a "body statement" defined over an iteration domain $\mathcal{D}$, just like any other statement, and a "result statement." Moreover, it has a special *projection function*, π, a many-to one affine function, that maps $\mathcal{D}$ to $\mathcal{D}_r = \pi(\mathcal{D})$. This is the domain of the result statement.

In the PRDG, we have a special *reduction node* for the body, and there is an edge from the result node to this node, labeled with π. A reduction is written as:

$$\vec{i} \in \mathcal{D}_r : \langle S, \vec{i} \rangle = \bigoplus_{\substack{\vec{j} \in \mathcal{D} \\ \vec{i} = \pi(\vec{j})}} g(\langle S_1, f_1(\vec{j}) \rangle, \ldots, \langle S_d, f_d(\vec{j}) \rangle)$$

where $\oplus$ is an associative and communicative binary operator.

Example 3.1. *[15] If* **a** *and* **b** *are vectors, representing two polynomials of degrees, m, and n, respectively, the vector* **c** *representing their product is obtained by computing the outer product of* ***a*** *and* ***b****, and adding up the anti-diagonals of this matrix:*

$$\mathbf{c}[k] = \sum_{k=i+j} \mathbf{a}[i] * \mathbf{b}[j]$$

4 Hyperrectangular Monoparametric partitioning

In this section, we focus on the two main mathematical objects in our program representation: polyhedra and affine functions. We show that applying a monoparametric partitioning transformation to these objects gives us, respectively, a union of polyhedra, and a piecewise affine function. These operations will be applied to partition a complete program in Section 5. In Section 6 we extend these two properties to any general shape and to complete programs.

4.1 Monoparametric partitionning of polyhedra

Consider a polyhedron $\mathcal{D} = \left\{\vec{i} \mid \ldots\right\} \subset \mathbb{Z}^n$. Let us introduce a *block size parameter b* and a diagonal matrix D of size n called *ratio of a tile*, whose coefficients are used to specify the "shape" of the partition.

We want to study the effect of applying monoparametric partitioning on a polyhedron $\mathcal{D}$. Such a transformation partitions each index, $\vec{i}$ into two indices (as in the "strip mining" intuition), a *block* index $\vec{i_b}$, and a *local* index. $\vec{i_l}$ We seek the constraints that these new indices must satisfy, i.e., we want to obtain a $2n$-dimensional set $\Delta = \{\vec{i_b}, \vec{i_l} \mid \text{constraints}\}$. Intuitively, $\vec{i_b}$ corresponds to the block to which $\vec{i}$ is mapped, and $\vec{i_l}$ corresponds to the local coordinate of $\vec{i}$ inside this block (c.f. Figure 1). We also assume that all parameters $\vec{p}$ can be decomposed into $\vec{p} = b.\vec{p_b} + \vec{p_l}$, where $\vec{p_b}$ are the *block parameters* and $\vec{p_l}$ the *local parameters*.

The intuition behind our main results is that starting from the constraints of $\mathcal{D}$, by eliminating the old indices $\vec{i}$ and parameters $\vec{p}$, it is possible to obtain a disjoint disjunction of integral affine constraints on the blocked and local indices and parameters. Therefore, we prove that Δ is a finite union of polyhedra. For example, if we tile a triangle $\mathcal{D} = \{i, j \mid 0 \le i, j \wedge i + j < N\}$ with square tiles (assuming that the block size parameter divides N), we have two tile shapes: the tiles along the diagonal are triangles, all internal tiles are full squares. Therefore, Δ is the union of two polyhedra: a one-dimensional collection of triangles corresponding to the diagonal tiles, and a two dimensional, triangular collection of squares corresponding to the interior tiles.

More interestingly, if the same triangle $\mathcal{D} = \{i, j \mid 0 \le i, j \wedge i + j < N\}$ is tiled with $2b \times b$ rectangles as in Fig. 1, we get three sets of shapes (if b divides N). In the "tall-skinny" triangular region ● $\{i_b, j_b \mid 0 \le i_b, j_b \wedge 2i_b + j_b + 3 \le N_b\}$, where $N_b = N/b$, we have full rectangles, specified by $\{i_l, j_l \mid 0 \le i_l < 2b \wedge 0 \le j_l < b\}$. Along the line segment ● $\{i_b, j_b \mid 2i_b + j_b + 2 = N_b\}$, we have trapezoidal tiles whose shape is $\{i_l, j_l \mid 0 \le i_l, j_l \wedge i_l + j_l < 2b \wedge j_l < b\}$. And finally, along the line segment ● $\{i_b, j_b \mid 2i_b + j_b + 1 = N_b\}$, we have triangular tiles $\{i_l, j_l \mid 0 \le i_l, j_l \wedge i_l + j_l < b\}$.

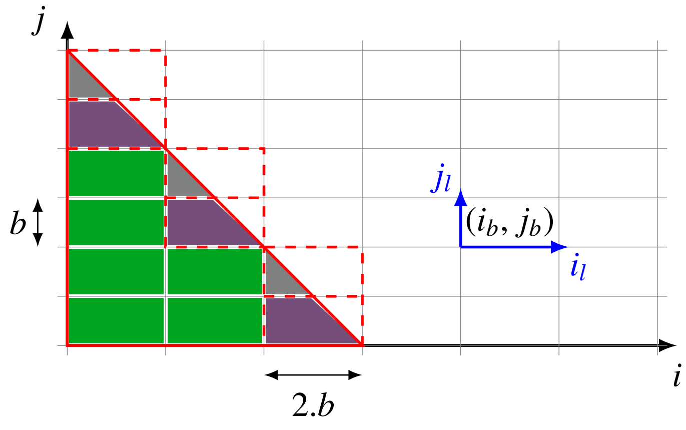


Figure 1: A 2D monoparametric partition with 2×1 rectangles.

Notice how each collection itself is a disjoint polyhedron, and its constraints involve *only the block indices*. Also notice how the constraints defining each shape involve *only the local indices*, and the size parameter, b. This is not a coincidence, and the following theorem shows that Δ is *separable* in this sense.

Theorem 4.1. *The image of a polyhedron $\mathcal{D} = \{\vec{i} \mid Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q} \geq \vec{0}\}$ by a monoparametric blocking transformation is:*

$$\Delta = \bigcap_{c=1}^{m} \Big[\biguplus_{k_c^{\min} < k_c \leq k_c^{\max}} \left\{ \vec{i_b}, \vec{i_l} \;\middle|\; \begin{array}{c} Q_c.D.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c = 0 \\ b.k_c \leq Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c \\ \vec{0} \leq \vec{i_l} < b.D.\vec{1} \end{array} \right\}$$
$$\uplus \left\{ \vec{i_b}, \vec{i_l} \;\middle|\; \begin{array}{c} Q_c.D.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c^{min} \geq 0 \\ \vec{0} \leq \vec{i_l} < b.D.\vec{1} \end{array} \right\} \Big]$$

where $\vec{k}$ enumerates the possible values of $\left\lfloor \frac{Q.\vec{i_l}+Q^{(p)}.\vec{p_l}+\vec{q}}{b} \right\rfloor \in [|\vec{k}^{\min}; \vec{k}^{\max}|]$.

Proof. Let us derive the constraints of Δ from the constraints of $\mathcal{D}$:

$$Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q} \geq \vec{0} \tag{1}$$

Since $\mathcal{D}$ is the intersection of m half planes, each one of them defined by a single constraint $Q_c.\vec{i} + Q_c^{(p)}.\vec{p} + q_c \geq 0$, for $1 \leq c \leq m$, and we consider each constraint independently. Let us use the definitions of $\vec{i_b}$, $\vec{i_l}$, $\vec{p_b}$ and $\vec{p_l}$ to eliminate $\vec{i}$ and $\vec{p}$.

$$b.Q_c.D.\vec{i_b} + Q_c.\vec{i_l} + b.Q_c^{(p)}.\vec{p_b} + Q_c^{(p)}.\vec{p_l} + q_c \geq 0 \tag{2}$$

Notice that these constraints are no longer linear, because of the $b.\vec{i_b}$ and $b.\vec{p_b}$ terms. To eliminate them, we divide each constraint by $b > 0$ to obtain:

$$Q_c.D.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + \frac{Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c}{b} \geq 0$$

In general, this fraction is a rational vector. Thus, to define integer points, we take the floor of each constraint (which is valid because $a \geq 0 \Leftrightarrow \lfloor a \rfloor \geq 0$ and $\lfloor n + a \rfloor = n + \lfloor a \rfloor$ for $n \in \mathbb{Z}$):

$$Q_c.D.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + \left\lfloor \frac{Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c}{b} \right\rfloor \geq 0 \tag{3}$$

Let use define $k_c(\vec{i_l}) = \left\lfloor \frac{Q_c.\vec{i_l}+Q_c^{(p)}.\vec{p_l}+q_c}{b} \right\rfloor$. Now $k_c(\vec{i_l})$ can only take a *constant non parametric* number of values. Indeed, $\vec{i_l}$ belongs to a rectangle: $0 \leq \vec{i_l} < D.b.\vec{1}$. Thus the maximum will be reached on a vertex of the rectangle, i.e., when all the coordinates of $\vec{i_l}$ are either 0 or $d.(b-1)$ (depending on the sign of the coefficient in front of it). Let us define QD_c^+ the vector of non-negative coefficients of $Q_c.D$, $Q_c^{(p)+}$ the vector of non-negative coefficients of $Q^{(p)}$, and note that $||.||_1$ denotes the L1-norm. We have:

$$k_c^{\max} = \max_{\vec{i_l}} \left\lfloor \frac{Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c}{b} \right\rfloor$$

Also: $Q_c.\vec{i_l} = \sum_j Q_{c,j}.\vec{i_l}(j)$. According to the remark above, the sum is maximized for $\vec{i_l}(j) = d_j.(b-1)$ if $Q_{c,j}.d_j(b-1) > 0$ and $\vec{i_l}(j) = 0$ otherwise. Hence: $\max_{\vec{i_l}} Q_c.\vec{i_l} = \sum_j \{Q_{c,j}.d_j(b-1) \mid Q_{c,j}.d_j > 0\}$, which is exactly $(b-1)||QD_c^+||_1$. Therefore:

$$\begin{aligned}
k_c^{\max} &= \left\lfloor \frac{||QD_c^+||_1.(b-1)+Q_c^{(p)}.\vec{p_l}+q_c}{b} \right\rfloor \\
&= ||QD_c^+||_1 + \left\lfloor \frac{Q_c^{(p)}.\vec{p_l}-||QD_c^+||_1+q_c}{b} \right\rfloor \\
&\leq ||QD_c^+||_1 + \left\lfloor \frac{||Q_c^{(p)+}||_1.(b-1)-||QD_c^+||_1+q_c}{b} \right\rfloor \\
&\leq ||QD_c^+||_1 + ||Q_c^{(p)+}||_1 + \left\lfloor \frac{q_c-||Q_c^{(p)+}||_1-||QD_c^+||_1}{b} \right\rfloor
\end{aligned}$$

Thus, we have a constant upper-bound on all $k_c(\vec{i_l})$. Likewise, we can show that we have a constant lower-bound on $k_c(\vec{i_l})$, therefore, $k_c(\vec{i_l})$ can only take a constant number of values. Thus, we can create one polyhedron per value of $k_c(\vec{i_l})$.

Let us build the polyhedron obtained for a value of $k_c(\vec{i_l})$ in $[|k_c^{\min}; k_c^{\max}|]$. Eqn (3) becomes:

$$Q_c.D.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c(\vec{i_l}) \geq 0 \tag{4}$$

$k_c(\vec{i_l})$ is the quotient of the integer division in (3). Then there exists $0 \leq r_c < b$ such that $Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c = b.k_c(\vec{i_l}) + r_c$. Hence:

$$b.k_c(\vec{i_l}) \leq Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c < b.(k_c(\vec{i_l}) + 1) \tag{5}$$

Also, the constraints (4) and (5) are affine, since $k_c(\vec{i_l})$ is a constant, and all we need to do is to ensure that $\vec{i_l}$ belongs to the tile, by adding the constraint $\vec{0} \leq \vec{i_l} < b.D.\vec{1}$, and we get the desired polyhedron.

To summarize, the c^{th} constraint of (1) has the same set of integer solutions as the union of polyhedra obtained for each value of $k_c(\vec{i_l})$:

$$\biguplus_{k_c} \left\{ \vec{i_b}, \vec{i_l} \;\middle|\; \begin{array}{c} Q_c.D.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c \geq 0 \\ b.k_c \leq Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c < b.(k_c+1) \\ \vec{0} \leq \vec{i_l} < b.D.\vec{1} \end{array} \right\}$$

where k_c enumerates all possible values of $\left\lfloor \frac{Q_c.\vec{i_l}+Q_c^{(p)}.\vec{p_l}+q_c}{b} \right\rfloor$ in the interval $[|k_c^{\min}; k_c^{\max}|]$.

Now, all we need to do is to intersect these unions for each constraint $c \in [|1; m|]$ to obtain the blocking. Actually, it is possible to improve the result, as described below.

First, let us study the pattern of the constraints of the polyhedra of the union. Let us call $(Block_{k_c})$ the constraint on the block indices and $(Local_{k_c})$ the constraints on the local indices. We can notice some properties among these constraints (Figure 2):

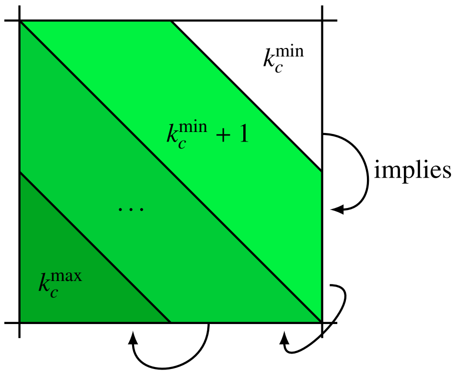


Figure 2: Stripe coverage inside a given tile

- Each k_c covers a different stripe of a tile (whose equations is given by $(Local_{k_c})$). The union of all these stripes, for $k_c^{\min} \leq k_c \leq k_c^{\max}$ forms a partition of the whole tile (by definition of $k_c^{\min}$ and $k_c^{\max}$).

- If a tile $\vec{i_b}$ satisfies the constraint $(Block_{k_c})$ for a given k_c, then the same tile also satisfies $(Block_{k'_c})$ for every $k'_c > k_c$ (because $a \geq 0 \Rightarrow a+1 \geq 0$). In other words, if the k_cth stripe in a tile is non-empty, the tile will have all the k'_c stripes, for every $k'_c > k_c$.

Thus, if a block $\vec{i_b}$ satisfies $(Block_{k_c^{\min}})$, then it satisfy all the $(Block_{k_c})$ for $k_c \geq k_c^{\min}$ and the whole rectangular tile is covered by the union of polyhedra Δ

Also, if a block $\vec{i_b}$ satisfies exactly $(Block_{k_c})$ (i.e., if $Q_c.D.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c = 0$), then it does not satisfy the $(Block_{k'_c})$ for $k'_c < k_c$ and we do not have the stripes below k_c. Therefore, only the local indices i_l which satisfy $(b.k_c \leq Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c)$ are covered by the union of polyhedra Δ.

Using these observations, we can separate the tiles into two categories: those which satisfy $(Block_{k_c^{\min}})$ (corresponding to a full tile), and those which satisfy exactly a $(Block_{k_c})$ where $k_c^{\min} < k_c$ (corresponding to a portion of the tile).

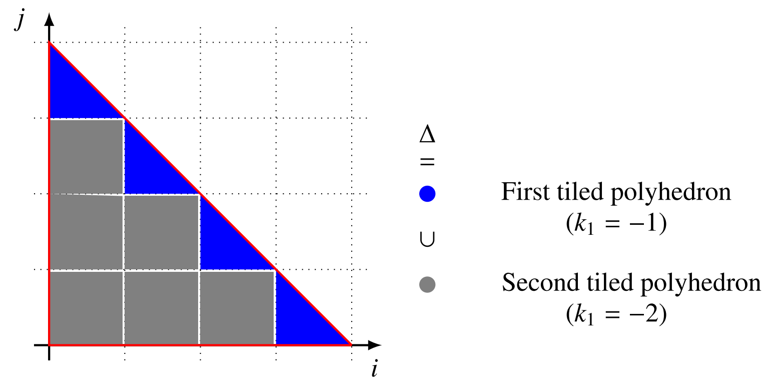
Mathematically, by splitting all of the polyhedra of the union according to the constraints $Q_c.D\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c = 0$, $k_c^{min} < k_c \leq k_c^{max}$, then pasting them together, we obtain the following improved expression:

$$\biguplus_{k_c^{\min} < k_c \leq k_c^{\max}} \left\{ \vec{i_b}, \vec{i_l} \;\middle|\; \begin{array}{c} Q_c.D.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c = 0 \\ b.k_c \leq Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c \\ \vec{0} \leq \vec{i_l} < b.D.\vec{1} \end{array} \right\}$$
$$\uplus \left\{ \vec{i_b}, \vec{i_l} \;\middle|\; \begin{array}{c} Q_c.D.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c^{min} \geq 0 \\ \vec{0} \leq \vec{i_l} < b.D.\vec{1} \end{array} \right\}$$

Thus, by intersecting all of these unions for each constraint, we obtain the expression of Δ. By distributing the intersection of the union of polyhedra, we obtain a union of disjoint polyhedra. After eliminating the empty polyhedra, the number of obtained disjoint polyhedra is the number of different tile shapes of the partitioned version of $\mathcal{D}$. □

Example 4.2. *Let us consider the following parameterized triangle:*

$$\mathcal{D} = \{i, j \mid N - 1 - i - j \geq 0 \wedge i \geq 0 \wedge j \geq 0\}$$



Obtained union of tiled polyhedra Δ

Figure 3: Obtained union of tiled polyhedra Δ for Example 4.2

We consider monoparametric tiles of size $b \times b$. Let us introduce $\begin{pmatrix} i \\ j \end{pmatrix} = b.\begin{pmatrix} i_b \\ j_b \end{pmatrix} + \begin{pmatrix} i_l \\ j_l \end{pmatrix}$ and, to simplify the result, let us assume that the parameter N is a multiple of the size parameter b: $N = N_b.b$. Then, the first inequality becomes:

$$\begin{aligned} N - 1 - i - j \geq 0 \quad &\Leftrightarrow \quad N_b.b - 1 - b.i_b - i_l - b.j_b - j_l \geq 0 \\ &\Leftrightarrow \quad N_b - i_b - j_b + \left\lfloor \frac{-i_l - j_l - 1}{b} \right\rfloor \geq 0 \end{aligned}$$

Let us study the values of $k_1(i_l, j_l) = \left\lfloor \frac{-i_l - j_l - 1}{b} \right\rfloor$. Because of the sign of the numerator coefficients, the maximum is -1 ($i_l = j_l = 0$) and the minimum is -2 ($i_l = j_l = b - 1$). After analyzing the two other inequalities, we obtain:

$$\Delta = \left\{ i_b, j_b, i_l, j_l \mid \begin{array}{c} N_b - i_b - j_b - 1 = 0 \\ i_b, j_b \geq 0 \\ 0 \leq i_l, j_l < b \\ -b \leq -i_l - j_l - 1 \end{array} \right\} \biguplus \left\{ i_b, j_b, i_l, j_l \mid \begin{array}{c} N_b - i_b - j_b - 2 \geq 0 \\ i_b, j_b \geq 0 \\ 0 \leq i_l, j_l < b \end{array} \right\}$$

This union of polyhedra is shown in Figure 3.

Example 4.3. *Let us consider the following polyhedron: $\mathcal{D} = \{i, j \mid i + j \leq N - 1 \ \wedge \ j \leq M \ \wedge \ 0 \leq i, j\}$ with tiles of size $b \times b$. Let us define $N = N_b.b + N_l$ and $M = M_b.b + M_l$ the block and local parameters, where $0 \leq M_l < b$ and $0 \leq N_l < b$. By going through the same steps as in the proof, we obtain:*

$$\left\{ \begin{array}{l} N - 1 - i - j \geq 0 \\ M - j \geq 0 \\ i \geq 0 \\ j \geq 0 \end{array} \right. \rightsquigarrow \left\{ \begin{array}{l} N_b - i_b - j_b + k_1 \geq 0 \\ M_b - j_b + k_2 \geq 0 \\ i_b + k_3 \geq 0 \\ j_b + k_4 \geq 0 \end{array} \right.$$

where

$$\left\{ \begin{array}{l} k_1 = \left\lfloor \frac{N_l - i_l - j_l - 1}{b} \right\rfloor = -2, -1 \ or \ 0 \\ k_2 = \left\lfloor \frac{M_l - j_l}{b} \right\rfloor = -1 \ or \ 0 \\ k_3 = \left\lfloor \frac{i_l}{b} \right\rfloor = 0 \\ k_4 = \left\lfloor \frac{j_l}{b} \right\rfloor = 0 \end{array} \right.$$

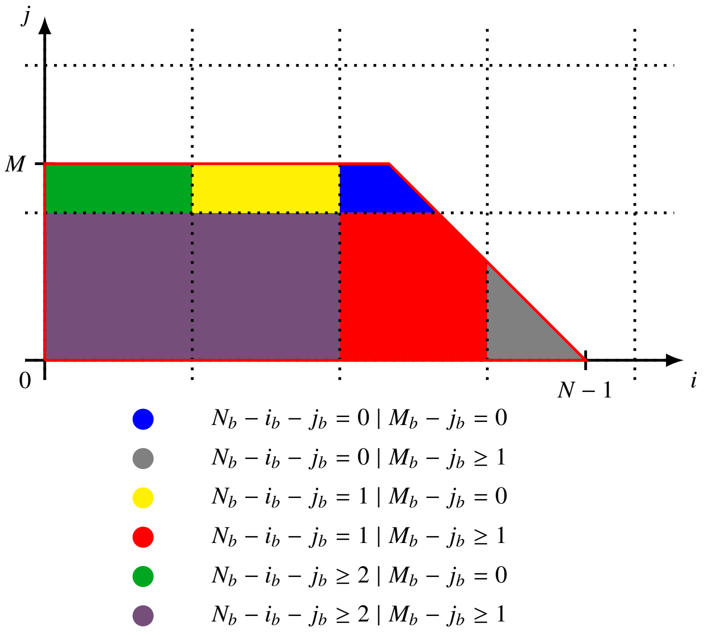

j
M
0
$N-1$
i
🔵 $N_b - i_b - j_b = 0 \mid M_b - j_b = 0$
⚪ $N_b - i_b - j_b = 0 \mid M_b - j_b \geq 1$
🟡 $N_b - i_b - j_b = 1 \mid M_b - j_b = 0$
🔴 $N_b - i_b - j_b = 1 \mid M_b - j_b \geq 1$
🟢 $N_b - i_b - j_b \geq 2 \mid M_b - j_b = 0$
🟣 $N_b - i_b - j_b \geq 2 \mid M_b - j_b \geq 1$

Figure 4: Example 4.3 - obtained union of polyhedra

We obtain a union of 6 polyhedra, one for each possible values of (k_1, k_2, k_3, k_4) *which are shown in Figure 4. We notice that two of these polyhedra (yellow and green) have the same shapes: this is because* $M_l \leq N_l$*, and these two polyhedra do not have the same shape otherwise. Moreover, when* $M_l \geq N_l$*, the blue and grey polyhedra will have the same shape.*

4.2 Monoparametric partitionning of affine functions

Let us consider an affine function $f : (\vec{i} \mapsto Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q})$. We assume that each block size is a multiple of the same *size parameter* b. Thus, we have, for the *input indices*, the following nonlinear (quadratic) equality: $\vec{i} = b.D.\vec{i_b} + \vec{i_l}$, where:

- $\vec{i_b}$ are called the *block indices*, $\vec{i_l}$ are called the *local indices* and we have $\vec{0} \leq \vec{i_l} < b.D.\vec{1}$.
- D is a diagonal matrix of strictly positive integer coefficients.

Likewise, we can introduce the same objects for the *output indices*: $\vec{i'} = b.D'.\vec{i_b'} + \vec{i_l'}$ with similar assumptions. We also assume that all parameters $\vec{p}$ can be decomposed in the same fashion: $\vec{p} = b.\vec{p_b} + \vec{p_l}$ where $\vec{p_b}$ is called the *blocked parameters*, $\vec{p_l}$ the *local parameters* and $\vec{0} \leq \vec{p_l} < b.\vec{1}$.

Blocking the affine function f is the same as replacing the input indices by $\vec{i_b}$ and $\vec{i_l}$, and expressing the result in the tiled space $(\vec{i_b'}, \vec{i_l'})$. Thus, if $f : \mathbb{Z}^n \mapsto \mathbb{Z}^{n'}$, we want to obtain a function $\phi : \mathbb{Z}^{2n} \mapsto \mathbb{Z}^{2n'}$. By starting with the definition of f, we can derive the value of ϕ, using a similar derivation as in the previous subsection:

Theorem 4.4. *Given two monoparametric blocking transformation* ($\mathcal{T}_b$ *and* $\mathcal{T}'_b$) *and any affine function* ($f(\vec{i}) = Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q}$) *the composition* $\mathcal{T}'_b \circ f \circ \mathcal{T}_b^{-1}$ *is a piecewise affine function, whose branches* *are:*

$$\begin{aligned}\phi(\vec{i_b},\vec{i_l}) = &\begin{pmatrix} D'^{-1}.Q.D.\vec{i_b} + D'^{-1}.Q^{(p)}.\vec{p_b} + \vec{k} \\ Q.\vec{i_l} + Q^{(p)}.\vec{p_l} + \vec{q} - b.D'.\vec{k} \end{pmatrix} \\ &\textit{if} \quad b.\vec{k} \le D'^{-1}.Q.\vec{i_l} + D'^{-1}.Q^{(p)}.\vec{p_l} + D'^{-1}.\vec{q} < b.(\vec{k} + \vec{1})\end{aligned}$$

for each $\vec{k} \in [|\vec{k}^{\min}; \vec{k}^{\max}|]$*, and assuming that* $(D'^{-1}.Q.D)$ *and* $(D'^{-1}.Q^{(p)})$ *are integer matrices.*

Proof. Let us start from the definition of f: $\vec{i'} = Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q}$. With similar arguments as at the beginning of the proof of Theorem 4.1, we can get rid of $\vec{i_l}'$ to obtain:

$$\vec{i_b}' = \left\lfloor D'^{-1}.Q.D.\vec{i_b} + D'^{-1}.Q^{(p)}.\vec{p_b} + \frac{D'^{-1}.(Q.\vec{i_l} + Q^{(p)}.\vec{p_l} + \vec{q})}{b} \right\rfloor$$

However, we have no guaranty that, in general, $(D'^{-1}.Q.D.\vec{i_b})$ and $(D'^{-1}.Q^{(p)}.\vec{p_b})$ are integral vectors. To allow us to draw these terms outside the floor operator, we will assume that $(D'^{-1}.Q.D)$ and $(D'^{-1}.Q^{(p)})$ are integer matrices. We will show later that these two hypotheses form a necessary and sufficient condition to have only affine conditions in the piecewise affine function ϕ. We obtain:

$$\vec{i_b}' = D'^{-1}.Q.D.\vec{i_b} + D'^{-1}.Q^{(p)}.\vec{p_b} + \left\lfloor \frac{D'^{-1}.(Q.\vec{i_l} + Q^{(p)}.\vec{p_l} + \vec{q})}{b} \right\rfloor$$

By defining $\vec{k}(\vec{i_l}) = \left\lfloor \frac{D'^{-1}.(Q.\vec{i_l}+Q^{(p)}.\vec{p_l}+\vec{q})}{b} \right\rfloor$ and by conducting the same kind of analysis as previously, we manage to bound $\vec{k}(\vec{ii})$ between $\vec{k}^{\min}$ and $\vec{k}^{\max}$. Finally, we obtain a piecewise expression of $\vec{i_b}'$, in which each branch corresponds to a value of $\vec{k}(i_l)$:

$$\begin{aligned}&\vec{i_b}' = D'^{-1}.Q.D.\vec{i_b} + D'^{-1}.Q^{(p)}.\vec{p_b} + \vec{k} \\ &\quad \text{if} \quad b.\vec{k} \le D'^{-1}.Q.\vec{i_l} + D'^{-1}.Q^{(p)}.\vec{p_l} + D'^{-1}.\vec{q} < b.(\vec{k} + \vec{1})\end{aligned}$$

for each $\vec{k} \in [|\vec{k}^{\min}; \vec{k}^{\max}|]$.

We can easily compute $\vec{i_l}'$ for each obtained branch by using the definition of $\vec{i_b}'$, to obtain the expression of ϕ as a piecewise affine function. Indeed, for a given branch:

$$\begin{aligned}\vec{i_l}' &= \vec{i'} - b.D'.\vec{i_b}' = Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q} - b.D'.(D'^{-1}.Q.D.\vec{i_b} + D'^{-1}.Q^{(p)}.\vec{p_b} + \vec{k}) \\ &= Q.(b.D.\vec{i_b} + \vec{i_l}) + Q^{(p)}.(b.\vec{p_b} + \vec{p_l}) + \vec{q} - b.Q.D.\vec{i_b} - b.Q^{(p)}.\vec{p_b} - b.D'.\vec{k} \\ &= Q.\vec{i_l} + Q^{(p)}.\vec{p_l} + \vec{q} - b.D'.\vec{k}\end{aligned}$$

□

Compared to the decomposition we obtained for polyhedra, we might have several pieces in the same tile. Indeed the value of the piecewise affine function is different for each branch, thus we cannot merge them.

Example 4.5. *Let us consider the affine function* $f : (i, j \mapsto 2i, N - j - 1, i + j)$*.*

Let us introduce $\begin{pmatrix} i \\ j \end{pmatrix} = b.\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.\begin{pmatrix} i_b \\ j_b \end{pmatrix} + \begin{pmatrix} i_l \\ j_l \end{pmatrix}$ *where* $0 \le i_l, j_l < 2b$ *and* $\begin{pmatrix} i' \\ j' \end{pmatrix} = b.\begin{pmatrix} i'_b \\ j'_b \end{pmatrix} + \begin{pmatrix} i'_l \\ j'_l \end{pmatrix}$ *where* $0 \le i'_l, j'_l < b$*. We assume that the parameter* N *is divisible by* b*, and we introduce* $N = N_b.b$*. We can check that* $(D'^{-1}.Q.D)$ *and* $(D'^{-1}.Q^{(p)})$ *are both integral, thus we will have polyhedral constraints.*

After performing the operations described previously, we obtain an expression of $\vec{i_b}'$*:*

$$\begin{bmatrix} i'_b \\ j'_b \\ k'_b \end{bmatrix} = \begin{pmatrix} 4 & 0 \\ 0 & -2 \\ 1 & 1 \end{pmatrix} \begin{bmatrix} i_b \\ j_b \end{bmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} . \begin{bmatrix} M \end{bmatrix} + \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

$$\phi(i_b, j_b, i_l, j_l) = \begin{cases} (4i_b, M - 2j_b - 1, i_b + j_b, & 2i_l, b - j_l - 1, i_l + j_l) \\ & \quad \text{if } \; 0 \le i_l < b \wedge 0 \le j_l < b \wedge 0 \le i_l + j_l < 2b \\ (4i_b + 1, M - 2j_b - 1, i_b + j_b, & 2i_l - b, b - j_l - 1, i_l + j_l) \\ & \quad \text{if } \; b \le i_l < 2b \wedge 0 \le j_l < b \wedge 0 \le i_l + j_l < 2b \\ (4i_b, M - 2j_b - 2, i_b + j_b, & 2i_l, 2b - j_l - 1, i_l + j_l) \\ & \quad \text{if } \; 0 \le i_l < b \wedge b \le j_l < 2b \wedge 0 \le i_l + j_l < 2b \\ (4i_b, M - 2j_b - 2, i_b + j_b + 1, & 2i_l, 2b - j_l - 1, i_l + j_l - 2b) \\ & \quad \text{if } \; 0 \le i_l < b \wedge b \le j_l < 2b \wedge 2b \le i_l + j_l < 4b \\ (4i_b + 1, M - 2j_b - 1, i_b + j_b + 1, & 2i_l - b, b - j_l - 1, i_l + j_l - 2b) \\ & \quad \text{if } \; b \le i_l < 2b \wedge 0 \le j_l < b \wedge 2b \le i_l + j_l < 4b \\ (4i_b + 1, M - 2j_b - 2, i_b + j_b + 1, & 2i_l - b, 2b - j_l - 1, i_l + j_l - 2b) \\ & \quad \text{if } \; b \le i_l < 2b \wedge b \le j_l < 2b \wedge 2b \le i_l + j_l < 4b \end{cases}$$

Figure 5: Example 4.5 - obtained piecewise affine function

where $k_1 = \left\lfloor \frac{2i_l}{b} \right\rfloor$, $k_2 = \left\lfloor \frac{-j_l-1}{b} \right\rfloor$ *and* $k_3 = \left\lfloor \frac{i_l+j_l}{2b} \right\rfloor$. *Thus,* $0 \le k_1 \le 1$, $-2 \le k_2 \le -1$ *and* $0 \le k_3 \le 1$.

Two out of the resulting eight branches have unsatisfiable conditions. Therefore, after pruning them out, we obtain the expression of ϕ *described in Figure 5.*

Example 4.6. *Let us consider the affine function* $f : (i, j \mapsto 2N + 2i + 4j - 1)$, *with a block size of* $b \times b$ *for the input indices, and* $2b$ *for the output indices. The conditions are verified, and we obtain after derivation:*

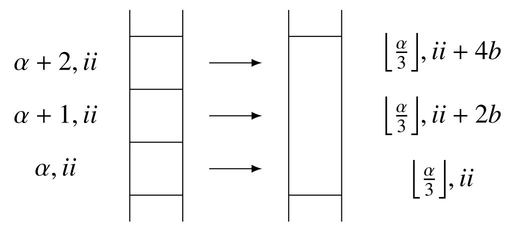
$$i'_b = N_b + i_b + 2j_b + k_1 \;\; \textit{where} \;\; k_1 = \left\lfloor \frac{2.N_l + 2.i_l + 4.j_l - 1}{2b} \right\rfloor$$

The final result is:

$$(i'_b, i'_l) = \begin{cases} (N_b + i_b + 2.j_b - 1, N_l + i_l + 2.j_l + b) & \textit{if} \;\; 2.N_l + 2.i_l + 4.j_l - 1 < 0 \\ (N_b + i_b + 2.j_b, N_l + i_l + 2.j_l) & \textit{if} \;\; 0 \le 2.N_l + 2.i_l + 4.j_l - 1 < 2b \\ (N_b + i_b + 2.j_b + 1, N_l + i_l + 2.j_l - b) & \textit{if} \;\; 2b \le 2.N_l + 2.i_l + 4.j_l - 1 < 4b \\ (N_b + i_b + 2.j_b + 2, N_l + i_l + 2.j_l - 2b) & \textit{if} \;\; 4b \le 2.N_l + 2.i_l + 4.j_l - 1 < 6b \\ (N_b + i_b + 2.j_b + 3, N_l + i_l + 2.j_l - 3b) & \textit{if} \;\; 6b \le 2.N_l + 2.i_l + 4.j_l - 1 \end{cases}$$

In Theorem 4.4, we have introduced a condition on two products of matrices to have only affine conditions in ϕ. Let us see what happens when this condition is not satisfied.

Example 4.7. *Let us consider the identity function* $(i \mapsto i)$ *where* $D = (2)$ *and* $D' = (6)$. *Because* $Q = (1)$ *and* $Q^{(p)} = (0)$, $D'^{-1}.Q.D = \begin{pmatrix} \frac{1}{3} \end{pmatrix}$ *and* $D'^{-1}.Q^{(p)} = (0)$, *the conditions are not satisfied. In particular, given a point* (i_b, i_l) *in the antecedent domain of this function, you have to know the result of the integer division of* i_b *by 3 to know in which block we end up, thus you need to know the value of* i_b *mod* 3 *to compute the new local index.*


$\alpha + 2, ii$
$\alpha + 1, ii$
α, ii
$\left\lfloor \frac{\alpha}{3} \right\rfloor, ii + 4b$
$\left\lfloor \frac{\alpha}{3} \right\rfloor, ii + 2b$
$\left\lfloor \frac{\alpha}{3} \right\rfloor, ii$

Necessary and sufficient condition to avoid modulo constraints Now, let us show that the condition in Theorem 4.4 is a necessary and sufficient condition to have only affine conditions in ϕ, and when it is not respected, conditions containing modulo appear in ϕ.

Theorem 4.8. *Given two monoparametric blocking transformation ($\mathcal{T}_b$ and $\mathcal{T}'_b$) and any affine function ($f(\vec{i}) = Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q}$). Assuming that the ratio of the tile of $\mathcal{T}_b$ is D and the ratio of the tile of $\mathcal{T}'_b$ is D', the composition $\mathcal{T}'_b \circ f \circ \mathcal{T}_b^{-1}$ has only polyhedral constraint iff ($D'^{-1}.Q.D$) and ($D'^{-1}.Q^{(p)}$) are integer matrices.*

Proof. In the proof of Theorem 4.4, we had obtained the following equality:

$$\vec{i_b}' = \left\lfloor D'^{-1}.Q.D.\vec{i_b} + D'^{-1}.Q^{(p)}.\vec{p_b} + \frac{D'^{-1}.(Q.\vec{i_l} + Q^{(p)}.\vec{p_l} + \vec{q})}{b} \right\rfloor$$

Let us consider the c-th dimension, $0 \leq c < |\vec{i_b}'|$:

$$i'_{bl,c} = \left\lfloor \frac{Q_c.D.\vec{i_b}}{D'_{c,c}} + \frac{Q_c^{(p)}.\vec{p_b}}{D'_{c,c}} + \frac{Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c}{D'_{c,c}.b} \right\rfloor$$

If the fraction $\frac{Q_c.D}{D'_{c,c}}$ is non-integral, it can affect the value of k_c (which was previously only a function of the local indices and parameters). This means that, depending the value of $\vec{i_b}$ and its modulo with respect to $D'_{c,c}$, the value of k_c is shifted and the cuts are different. Thus, we need to distinguish the different values of i_b modulo $D'_{c,c}$, which is a non-affine constraints. Likewise, if the fraction $\frac{Q_c^{(p)}.\vec{p_b}}{D'_{c,c}}$ is non-integer, $\vec{p_b}$ affects the value of k_c and we have non-affine constraints on $\vec{p_b}$.

Therefore, we just have shown that if the condition is not satisfied, then we have modulo constraints. Theorem 4.4 has already shown that if the condition is satisfied, we do not have modulo constraints. Therefore, this condition is a necessary and sufficient condition. □

Example 4.7 shows what happens in practice when the condition is not satisfied.

Derivation when the condition is not satisfied If the necessary and sufficient condition is not satisfied, we can still finish the computation of ϕ and obtain a piecewise affine function with modulo conditions, as shown by the following theorem:

Theorem 4.9. *If the condition is not satisfied, ϕ is a piecewise affine function with modulo conditions.*

Proof. Assuming that this condition is not satisfied, we introduce $\vec{i_b} = \vec{i_b}^{(div),l}.D'_{l,l} + \vec{i_b}^{(mod),l}$ and $\vec{p_b} = \vec{p_b}^{(div),l}.D'_{l,l} + \vec{p_b}^{(mod),l}$ where $\vec{0} \leq \vec{i_b}^{(mod),l} < D'_{l,l}.\vec{1}$ and $\vec{0} \leq \vec{p_b}^{(mod),l} < D'_{l,l}.\vec{1}$. We obtain the following equality in our derivation:

$$\vec{i_{bl}}' = Q_l.D.\vec{i_b}^{(div),l} + Q_l^{(p)}.\vec{p_b}^{(div),l} + \left\lfloor \frac{Q_l.D.\vec{i_b}^{(mod),l} + Q_l^{(p)}.\vec{p_b}^{(mod),l}}{D'_{l,l}.b} + \frac{Q_l.\vec{i_l} + Q_l^{(p)}.\vec{p_l} + q_l}{D'_{l,l}.b} \right\rfloor$$

Let us define $k_l(\vec{i_b}^{(mod),l}, \vec{p_b}^{(mod),l}) = \left\lfloor \frac{Q_l.D.\vec{i_b}^{(mod),l} + Q_l^{(p)}.\vec{p_b}^{(mod),l}}{D'_{l,l}.b} + \frac{Q_l.\vec{i_l} + Q_l^{(p)}.\vec{p_l} + q_l}{D'_{l,l}.b} \right\rfloor$. Because $\vec{i_b}^{(mod),l}$ and $\vec{p_b}^{(mod),l}$ can only take a finite number of value, we can do one analysis of k_l for each of their values.

The number of branches resulting from the analysis of the l-th dimension correspond to the number of values the triplet $(\vec{i_b}^{(mod),l}, \vec{p_b}^{(mod),l}, k_l)$ can take. The total number of branches of the piecewise affine function is the product between the number of branches coming from each dimension. Thus, the number of branches is considerable, but an expression of ϕ can be found. □

Even if we manage to get an expression of ϕ when the condition is not satisfied, the number of branches is considerable, and it means going outside of the polyhedral model.

Example 4.10. *Let us consider* $f : (i, j \mapsto i, j)$ *where the input indices are tiled as* $\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} i_b \\ j_b \end{pmatrix}.b + \begin{pmatrix} i_l \\ j_l \end{pmatrix}$ *and the output indices are tiled as* $\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{pmatrix} 2i'_b \\ 3j'_b \end{pmatrix}.b + \begin{pmatrix} i'_l \\ j'_l \end{pmatrix}$*. Let us consider the first output dimension:*

$$\begin{aligned} i' = i \quad &\Leftrightarrow \quad 2.i'_b.b + i'_l = i_b.b + i_l \\ &\Rightarrow \quad i'_b = \left\lfloor \tfrac{i_b}{2} + \tfrac{i_l}{2b} \right\rfloor = i^{(1)}_{bb} + \left\lfloor \tfrac{i^{(1)}_{bl}}{2} + \tfrac{i_l}{2b} \right\rfloor \end{aligned}$$

where $i_b = 2.i^{(1)}_{bb} + i^{(1)}_{bl}$ *and* $0 \leq i^{(1)}_{bl} \leq 1$*. Likewise, we have:*

$$j'_b = j^{(2)}_{bb} + \left\lfloor \frac{j^{(2)}_{bl}}{3} + \frac{j_l}{3b} \right\rfloor$$

where $j_b = 3.j^{(2)}_{bb} + j^{(2)}_{bl}$ *and* $0 \leq j^{(2)}_{bl} \leq 2$*. Finally, we can build the pieces of* ϕ *by enumerating all the possible values of* $i^{(1)}_{bl}$ *and* $j^{(2)}_{bl}$*. For example, for* $i^{(1)}_{bl} = j^{(2)}_{bl} = 0$*:*

$$\vec{k}(i_l, j_l) = \left(\left\lfloor \tfrac{i_l}{2b} \right\rfloor \quad \left\lfloor \tfrac{j_l}{3b} \right\rfloor \right)^T$$

We only have one possible value for $k_1(i_l, j_l)$ *and* $k_2(i_l, j_l)$ *(which is* 0 *in both cases), thus we will only have one branch in* ϕ *corresponding to these values. The full expression of* ϕ *is:*

$$\phi : \begin{bmatrix} i_b \\ j_b \\ i_l \\ j_l \end{bmatrix} \mapsto \begin{cases} (i_b/2, j_b/3, & i_l, j_l)^T & \text{if } i_b \equiv 0 \bmod 2 \wedge j_b \equiv 0 \bmod 3 \\ (i_b/2, (j_b-1)/3, & i_l, j_l + b)^T & \text{if } i_b \equiv 0 \bmod 2 \wedge j_b \equiv 1 \bmod 3 \\ (i_b/2, (j_b-2)/3, & i_l, j_l + 2b)^T & \text{if } i_b \equiv 0 \bmod 2 \wedge j_b \equiv 2 \bmod 3 \\ ((i_b-1)/2, j_b/3, & i_l + b, j_l)^T & \text{if } i_b \equiv 1 \bmod 2 \wedge j_b \equiv 0 \bmod 3 \\ ((i_b-1)/2, (j_b-1)/3, & i_l + b, j_l + b)^T & \text{if } i_b \equiv 1 \bmod 2 \wedge j_b \equiv 1 \bmod 3 \\ ((i_b-1)/2, (j_b-2)/3, & i_l + b, j_l + 2b)^T & \text{if } i_b \equiv 1 \bmod 2 \wedge j_b \equiv 2 \bmod 3 \end{cases}$$

5 Hyperrectangular monoparametric partitioning program transformation

In the previous section, we showed how to monoparametrically partition a polyhedron and an affine function, the two main mathematical objects in any polyhedral program representation. Now, we show how we can apply these tools to block a complete polyhedral program. We will show afterward how to choose a ratio for the local variables, which does not introduce modulo conditions in our transformed program.

5.1 Applying the monoparametric partitionning transformation to a program

Let us consider a polyhedral program:

$$\begin{aligned} (\forall \vec{i} \in \mathcal{D}) : S[\vec{i}] \quad &= \quad Expr(S_1[u_1(\vec{i})], \ldots, S_d[u_d(\vec{i})]) \\ (\forall \vec{i} \in \mathcal{D}_r) : S[\vec{i}] \quad &= \quad \bigoplus_{\substack{\vec{j} \in \mathcal{D} \\ \vec{i} = \pi(\vec{j})}} Expr(S_1[f_1(\vec{j})], \ldots, S_d[f_d(\vec{j})]) \end{aligned}$$

To apply the monoparametric partitionning transformation to this program, we have to replace all the polyhedron and affine function of this program by their monoparametric partitionned version. The number of dimensions of all domain is doubled, and, because the polyhedra and affine functions remain the same (but are expressed with different indices) the computation (and the order of computation) performed by the program is not changed.

However, this substitution introduces piecewise affine functions in the middle of the program, which is not allowed. Thus, a normalization step is required to rise the conditions of the branches of such piecewise affine function.

Example 5.1. *Let us consider the following program, corresponding to a Jacobi1D computation:*

$$\begin{array}{lcl}
(\forall 0 \leq i < N) : Out[i] & = & Temp[T-1, i] \\
(\forall 0 \leq i < N \wedge t = 0) : Temp[t, i] & = & I[i] \\
(\forall i = 0 \wedge 0 < t < T) : Temp[t, i] & = & Temp[t-1, i] \\
(\forall i = N-1 \wedge 0 < t < T) : Temp[t, i] & = & Temp[t-1, i] \\
(\forall 0 < i < N-1 \wedge 0 < t < T) : Temp[t, i] & = & (Temp[t-1, i-1]+ \\
 & & Temp[t-1, i] + Temp[t-1, i+1])/3
\end{array}$$

where Out is an output variable and I and input defined over $\{i | 0 \leq i < N\}$.

For simplicity, we assume that the parameters N and T are multiples of the tile size parameter b ($N = N_b.b$ *and* $T = T_b.b$*). After substituting every polyhedron and affine function and before the normalization step, we obtain the program described in Figure 5.1.*

The normalization step flattens all the branch conditions and pruning the empty branches. Each branch corresponds to a different computation. However, if we do this in two separated steps, this transformation does not scale. In our *Jacobi1D* example, if we consider the last equation, we have a summation between 3 variables. After flattening them and before pruning the empty branches, we have a total of $4 \times 2 \times 4 = 32$ branches (some of them being empty). This number explodes when considering stencils of higher-orders. For example, if we consider a *Jacobi2D* example, we have a summation between 9 variables, corresponding to a total of $4^8 * 2 = 2^{17}$ branches before pruning. Thus, the pruning must occur at every steps of the normalization transformation.

5.2 Derivation of the partitioning

While applying the monoparametric partitioning transformation, we might have different partitionings (a.k.a. ratio, in the case of rectangular tiles) interacting within an expression. For example, if we choose a ratio of 1×2 for a statement T, what ratio should we pick for a statement that uses T, say $S[i, j, k] = g(\dots T[i, k + j] \dots)$, and how to adapt this expression to make the (potentially different) statement partitioning compatible?

If we assume that the ratio of all variables were chosen beforehand, we just have to check for their compatibility, i.e., we have to check that partitioning the dependence functions do not introduce non-polyhedral modulo constraints (cf Theorem 4.8). This means that we have to check that, for any dependence function $(\vec{i} \mapsto Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q})$ and ratio D and D', $(D'^{-1}.Q.D)$ and $(D'^{-1}.Q^{(p)})$ are integral.

In a more general situation, we can assume that the ratio of some variables was chosen beforehand (either by the user or by the compiler), but not all ratios were decided. It is possible to adapt the different ratio by partitioning the dependences functions accordingly (with different partitionings on each side). We assume that for any loop in our PRDG representation, at least one statement was given a ratio. For example, if a variable S depends on itself then its ratio must be specified. Or if a variable S depends on T, which depends on S, at least one of these variable must have their ratio specified. This condition avoid recursive divisibility equation when we derive the missing ratio of the program. About the order of derivation of the missing ratio of the program, under this condition, it is always possible to find a statement for which all the statements it uses have already been given a ratio. Therefore, by considering successively such statements, we can derive a ratio for all the statements of the program.

$$\forall \left\{ \begin{array}{l} 0 \le i_b < N_b \\ 0 \le i_l < b \end{array} \right. : Out[i_b, i_l] = Temp[T_b, i_b, b-1, i_l]$$

$$\forall \left\{ \begin{array}{l} 0 \le i_b < N_b \\ 0 \le i_l < b \\ t_b = t_l = 0 \end{array} \right. : Temp[t_b, i_b, t_l, i_l] = I[i_b, i_l]$$

$$\forall \left\{ \begin{array}{l} i_b = i_l = 0 \\ \left(\begin{array}{c} 0 < t_b \wedge 0 \le t_l < b \\ \vee \\ t_b = 0 \wedge 0 < t_l < b \end{array} \right) \end{array} \right. : Temp[t_b, i_b, t_l, i_l] = Temp \left[\begin{array}{l} t_l = 0 : (t_b - 1, i_b, b-1, i_l) \\ 0 < t_l : (t_b, i_b, t_l - 1, i_l) \end{array} \right]$$

$$\forall \left\{ \begin{array}{l} i_b = N_b - 1 \wedge i_l = b - 1 \\ \left(\begin{array}{c} 0 < t_b \wedge 0 \le t_l < b \\ \vee \\ t_b = 0 \wedge 0 < t_l < b \end{array} \right) \end{array} \right. : Temp[t_b, i_b, t_l, i_l] = Temp \left[\begin{array}{l} t_l = 0 : (t_b - 1, i_b, b-1, i_l) \\ 0 < t_l : (t_b, i_b, t_l - 1, i_l) \end{array} \right]$$

$$\begin{array}{l}
\forall \left\{ \begin{array}{l} \left(\begin{array}{c} i_b = 0 \wedge 0 < i_l < b \\ \vee \\ i_b = N_b - 1 \wedge 0 \le i_l < b - 1 \\ \vee \\ 0 < i_b < N_b - 1 \wedge 0 < i_l < b \end{array} \right) \\ \left(\begin{array}{c} 0 < t_b \wedge 0 \le t_l < b \\ \vee \\ t_b = 0 \wedge 0 < t_l < b \end{array} \right) \end{array} \right. : Temp[t_b, i_b, t_l, i_l] = 1/3 \times (\\
Temp \left[\begin{array}{l} t_l = 0 \wedge i_l = 0 : (t_b - 1, i_b - 1, b - 1, b - 1) \\ t_l = 0 \wedge i_l > 0 : (t_b - 1, i_b, b - 1, i_l - 1) \\ 0 < t_l \wedge i_l = 0 : (t_b, i_b - 1, t_l - 1, b - 1) \\ 0 < t_l \wedge i_l > 0 : (t_b, i_b, t_l - 1, i_l - 1) \end{array} \right] + Temp \left[\begin{array}{l} t_l = 0 : (t_b - 1, i_b, b - 1, i_l) \\ 0 < t_l : (t_b, i_b, t_l - 1, i_l) \end{array} \right] \\
\qquad + Temp \left[\begin{array}{l} t_l = 0 \wedge i_l = b - 1 : (t_b - 1, i_b + 1, b - 1, 0) \\ t_l = 0 \wedge i_l < b - 1 : (t_b - 1, i_b, b - 1, i_l + 1) \\ 0 < t_l \wedge i_l = b - 1 : (t_b, i_b + 1, t_l - 1, 0) \\ 0 < t_l \wedge i_l < b - 1 : (t_b, i_b, t_l - 1, i_l + 1) \end{array} \right])
\end{array}$$

Figure 6: Jacobi1D computation, after substituting every polyhedron and affine function by its monoparametric partitioned equivalent) and before the normalization step. We summarize union of polyhedra by allowing disjunctions of polyhedra

We always pick the smallest ratios possible for an expression: indeed, let us assume that we have derived D_{T_k} for a statement $T_k[\vec{i}] = \dots$ and that we reach a statement $S[\vec{i}] = g(\dots T_k[f_k(\vec{i})]\dots)$ in which the ratio of S is determined. We have to make sure that $(D_{T_k}^{-1}.Q.D_S)$ is integer. We systematically take the lowest ratio possible, to minimize the risk the algorithm fails to avoid modulo constraints.

Let us consider a statements for which all the statements it uses have already been given a ratio. Two situations might arise:

- If the statement is normal: $S[\vec{i}] = g(T_1[f_1(\vec{i})], \dots, T_d[f_d(\vec{i})])$. Assuming that each dependence function f_k are of the form $f_k : (\vec{i} \mapsto Q.k.\vec{i} + Q.k^{(p)}.\vec{p} + \vec{q})$, the constraints that must be satisfied by the ratio of S are:

$$\begin{cases} (\forall 1 \leq k \leq d)\ D_{T_k}^{-1}.Qk.D_S \text{ is integer} \\ (\forall 1 \leq k \leq d)\ D_{T_k}^{-1}.Qk^{(p)} \text{ is integer} \end{cases}$$

which means:

$$\begin{cases} (\forall 1 \leq k \leq d)\ (\forall i,j)\ (D_{T_k})_i \text{ divides } Qk_{i,j}.(D_S)_j \\ (\forall 1 \leq k \leq d)\ (\forall i,j)\ (D_{T_k})_i \text{ divides } Qk_{i,j}^{(p)} \end{cases}$$

The last condition (concerning the parameters) does not impact the ratios of S. Moreover, if this condition is not satisfied, then we must have modulo constraints on the parameters when partitioning this dependence expression. Let us now study the first condition to find the smallest ratio of S possible. We can factorize $(D_{T_k})_i$ as a product of prime numbers. Because of the first condition, these prime numbers must be present either inside $Qk_{i,j}$ or $(D_S)_j$ (which is the unknown). If some of them are already inside $Qk_{i,j}$, they do not need to be in $(D_S)_j$. Thus, let us introduce $\delta k_{i,j}$, the product of prime factors of $(D_{T_k})_i$ which are not inside $Qk_{i,j}$:

$$\delta k_{i,j} = (D_{T_k})_i / gcd((D_{T_k})_i, Qk_{i,j})$$

The conditions become $(\forall k)(\forall i\ j), \delta k_{i,j}$ divides $(D_S)_j$. Thus, the smallest ratio we can take for S are:

$$(D_S)_j = lcm_{k,i}(\delta k_{i,j})$$

- If the statement is a reduction:

$$\langle S, \vec{i} \rangle = \bigoplus_{\substack{\vec{i} = \pi(\vec{j}) \\ \vec{j} \in \mathcal{D}}} g(\langle T_0, f_0(\vec{j}) \rangle, \dots, \langle T_d, f_d(\vec{j}) \rangle).$$

In the PRDG this statement has two nodes: one corresponding to the subexpression of the reduction body, and one corresponding to the reduction itself. We can determine the minimal ratios for the body node D_{SExpr} by using the method described in the previous case. However, we still need to partition the projection function $\pi : (\vec{j} \mapsto Q.\vec{j} + Q^{(p)}.\vec{p} + \vec{q})$. The conditions to avoid non-parametric modulo constraints are:

$$\begin{cases} (\forall i,j)\ (D_S)_i \text{ divides } Q_{i,j}.(D_{SExpr})_j \\ (\forall i,j)\ (D_S)_i \text{ divides } Q_{i,j}^{(p)} \end{cases}$$

We notice that the divisibility constraint is in the opposite direction than what we had in the previous case: instead of having to find a value of $(D_S)_i$ which is divisible by another value, we have to find a value of $(D_S)_i$ which divides another value. Thus, we could just take $(D_S)_i = 1$, which is the smallest ratio possible. However, after simplification, we might obtain a projection function which does not admit an integer right inverse [31]. For example, if $D_{SExpr} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$ and the projection function is $\pi : (i, j \mapsto i)$, then we obtain a piecewise affine function with two branches:

$$i_b, j_b, i_l, j_l \mapsto \begin{cases} 2.i_b, i_l & \text{when} \quad 0 \leq i_l < b \\ 2.i_b + 1, i_l - b & \text{when} \quad b \leq i_l < 2b \end{cases}$$

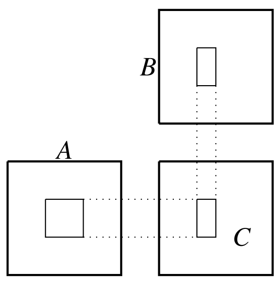


Figure 7: Example 5.2 - Chosen ratios

Because a projection function must be affine, we need to split the reduction into two reductions, whose projection function correspond to a single branch of the piecewise affine function. Because each branch independently does not admit an integer right inverse, each obtained reduction is defined over non-polyhedral domains (respectively "i'_b even" and "i'_b odd").

To avoid this situation, we apply a preprocessing step to the program to make the projection canonic, i.e., of the form $(\vec{x}, \vec{y} \mapsto \vec{x})$. Then, we just keep the ratio of SExpr for the dimensions which are not projected. In that case, the partitioned projection function will have only one branch, of the form $(\vec{x_b}, \vec{y_b}, \vec{x_l}, \vec{y_l} \mapsto \vec{x_b}, \vec{x_l})$.

We can notice that taking a square shape $(1 \times 1 \times \cdots \times 1)$ for every variable automatically satisfies all the constraints. Thus, there always exists an assignment of ratios which can be provided by the user, and which avoid modulo conditions.

Example 5.2. *Let us consider a matrix multiplication computation, where the ratios of A are $2b \times 2b$, the ratios of B are $2b \times b$ and the ratios of C are $2b \times b$:*

$$(\forall 0 \leq i, j < N) C[i, j] = \sum_{0 \leq k < N} A[i, k] * B[k, j]$$

After applying our algorithm, the minimal ratio for the subexpression of the reduction is $2b \times b \times 2b$. After the reduction, because we project out the k dimension, the smallest ratio becomes $2b \times b$. These ratios are the same as C, thus the algorithm succeeds. The result of the derivation is shown in Figure 7.

Example 5.3. *Let us consider the following program, in which A got a ratio of 2, B got a ratio of 3 and Out a ratio of b:*

$$(\forall 0 \leq i < N) Temp[i] = A[i + 1] + B[3.i] (\forall 0 \leq i < N) Out[i] = Temp[i]$$

We consider the first equation to derive the minimum ratio of Temp which does not introduce modulo constraints. The contribution of A in this equation ($A[i+1]$) forces it to be divisible by $2b$. The contribution of B in this equation $B[3.i]$ forces it to be divisible by $3b/3 = b$. Therefore, the minimal ratio of Temp is $2b$.

Then, we consider the second equation. The ratio of Temp is $2b$ and the ratio of Out is b, thus we are forced to have a modulo constraint here and the algorithm fails. If the ratio picked for Out is a multiple of $2b$, the ratios match.

Set of possible ratios Let us show that if our algorithm fails, then there is no possible combination of ratio which does not introduce any modulo conditions.

Theorem 5.4. *The set of valid ratio for a variable are the multiples of a single minimal ratio, which is the one picked by our algorithm. Therefore, if our algorithm fails, then the specified ratios do not lead to any possible combination of ratio for the rest of the program.*

Proof. The ratios we compute for each variable are everytimes locally the smallest one which avoid modulo constraints. Also, all the constraints involving ratios are divisibility constraints. Therefore, the ratio we pick for a variable is the product of the divisors which cannot be eliminated (through the array access function).

Thus, at the end of the algorithm (when we check the coherency of the ratio by examining the equations of the variables which already had their ratio decided), the ratio of the right hand side of every equations must divide the ratio of the equation variable. If this condition is not satisfied, it exists at least a divisor of the ratio of the right hand side of the equation which cannot be eliminated and we cannot avoid the modulo condition. □

We remark that, for an arbitrary subexpression, the set of valid ratio must be a multiple of the minimal ratio we find. In the case where the compiler framework allows modulo constraints, we can technically take whatever aspect ratio we want. However, if we want to minimize the number of generated case, we should still avoid modulo conditions as much as possible.

6 General monoparametric partitioning

In Sections 4 and 5, we only considered *canonic* monoparametric partitioning, i.e., hyperrectangular shapes for the partitions. We now show that this theory can be extended to any polyhedral tile shape (hexagonal [20], diamond [4], etc).

First of all, let us describe what a general monoparametric partitioning is. Let us start from a general fixed size partitioning. We need 3 objects to describe it:

- A non-parametric bounded convex polyhedron $\mathcal{P}$
- A non-parametric integer lattice $\mathcal{L}$ of the tile origins (which admits a basis L) and,
- A function $\mathcal{T}$ which decomposes any point $\vec{i}$ in the following way:

$$\mathcal{T}(\vec{i}) = (\vec{i_b}, \vec{i_l}) \Leftrightarrow \vec{i} = L.\vec{i_b} + \vec{i_l} \quad \text{where } (L.\vec{i_b}) \in \mathcal{L} \text{ and } \vec{i_l} \in \mathcal{P}$$

Notice that if the decomposition is not unique, then we have overlapping tiles. If the decomposition is unique, this partitioning defines a partition of the space. Some partitioning do not have an integral lattice of tile origins (such as diamond partitioning with non-unimodular hyperplanes). We do not consider partitioning with overlapped tiles or with non-integral tile origins in this thesis.

A *homothetic transformation* $a\times\mathcal{D}$, where a is a constant and $\mathcal{D}$ is a set, is the set $a\times\mathcal{D} = \{\vec{z} \mid \vec{z}/a \in \mathcal{D}\}$.

Definition 6.1. *A general* monoparametric partitioning *is a partitioning whose tile shape is the homothetic scaling of a fixed size partitioning, by a factor of b: $\mathcal{P}_b = b \times \mathcal{P}$. The new lattice of tile origins is $\mathcal{L}_b = b \times \mathcal{L}$ and we can obtain the new partitioning function $\mathcal{T}_b$ from $\mathcal{T}$.*

6.1 General monoparametric partitionning of polyhedra

Let us consider a n-dimensional polyhedron $\mathcal{D} = \{\vec{i} \mid Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q} \geq \vec{0}\}$ where $\vec{p}$ are the program parameters. As in Section 4, we want to replace $\vec{i}$ by the *block indices* $\vec{i_b}$ and the *local indices* $\vec{i_l}$, such that $\vec{i} = \mathcal{T}_b(\vec{i_b}, \vec{i_l})$ (cf Figure 8). We still assume that all parameters $\vec{p}$ can be decomposed into block and local parameters.

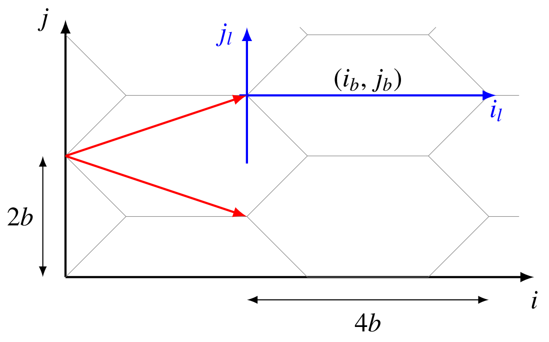


Figure 8: Example of hexagonal monoparametric blocking in the 2D case. The red arrows correspond to a basis of the lattice of tile origins

Let us start from a constraint of $\mathcal{D}$, say $Q_c.\vec{i} + Q_c^{(p)}.\vec{p} + q_c \geq 0$. We can replace $\vec{i}$ by $b.L.\vec{i_b} + \vec{i_l}$ where $\vec{i_l} \in \mathcal{P}_b$. By doing exactly the same operations as in the proof of Theorem 4.1, we obtain the following expression:

$$Q_c.L.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + \left\lfloor \frac{Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c}{b} \right\rfloor \geq \vec{0}$$

We define $k_c(\vec{i_l}) = \left\lfloor \frac{Q_c.\vec{i_l}+Q_c^{(p)}.\vec{p_l}+q_c}{b} \right\rfloor$. Because $\vec{i_l} \in \mathcal{P}_b$ and $\mathcal{P}_b = b{\times}\mathcal{P}$ where $\mathcal{P}$ is bounded, $k_c(\vec{i_l})$ can only take a finite number of values. Because the shape of the tile is more complex as a rectangle, we cannot simply look at the sign of the coefficient to find the extremal values of $k_c(\vec{i_l})$. Because k_c is an affine function and because $\vec{i_l}$ belongs to $\mathcal{P}_b$, we can use linear programming solvers (such as PIP [14]) to find the extremal values of $k_c(\vec{i_l})$. The rest of the proof caries on exactly in the same way as for Theorem 4.1.

Therefore, we obtain a union of polyhedron having the same properties as the rectangular case, for a general form of tiles:

Theorem 6.1. *The image of a polyhedron $\mathcal{D} = \{\vec{i} \mid Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q} \geq \vec{0}\}$ by a general monoparametric partitioning transformation is:*

$$\Delta = \bigcap_{c=1}^{m} \Bigg[\biguplus_{k_c^{\min}<k_c\leq k_c^{\max}} \left\{ \vec{i_b}, \vec{i_l} \;\middle|\; \begin{array}{c} Q_c.L.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c = 0 \\ b.k_c \leq Q_c.\vec{i_l} + Q_c^{(p)}.\vec{p_l} + q_c \\ \vec{i_l} \in \mathcal{P}_b \end{array} \right\}$$
$$\uplus \left\{ \vec{i_b}, \vec{i_l} \;\middle|\; \begin{array}{c} Q_c.L.\vec{i_b} + Q_c^{(p)}.\vec{p_b} + k_c^{min} \geq 0 \\ \vec{i_l} \in \mathcal{P}_b \end{array} \right\} \Bigg]$$

where $\vec{k}$ enumerates the possible values of $\left\lfloor \frac{Q.\vec{i_l}+Q^{(p)}.\vec{p_l}+\vec{q}}{b} \right\rfloor$.

After distributing the intersection across the unions and eliminating the empty polyhedral, we obtain as many polyhedra than the number of different tile shapes of the partitioned version of $\mathcal{D}$ (which is, at most the number of different values of $\vec{k}$).

Example 6.2. *Let us consider the following polyhedron: $\{i, j \mid j - i \leq N \wedge i + j \leq N \wedge 0 < j\}$ and the following partitioning:*

- $\mathcal{P}_b = \{i, j \mid -b < j \leq b \;\wedge\; -2b < i + j \leq 2b \;\wedge\; -2b < j - i \leq 2b\}$

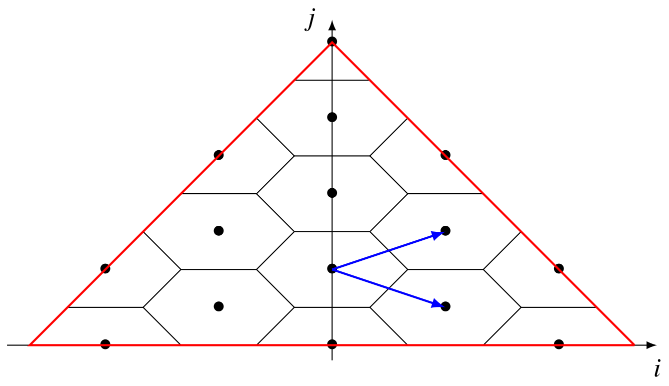


Figure 9: Polyhedron and tiling of Example 6.2. The dots correspond to the tile origins of the tiles contributing to the polyhedron

- $L_b = L.b.\mathbb{Z}^2$ *where* $L = \begin{bmatrix} 3 & 3 \\ 1 & -1 \end{bmatrix}$

For simplicity, we assume that $N = 6.b.N_b + 2b$*, where* N_b *is a positive integer. A graphical representation of the polyhedron and of the tiling is shown in Figure 9.*

Let us start with the first constraint of the polyhedron.

$$\begin{aligned} j - i \leq N \quad &\Leftrightarrow \quad 0 \leq 6.b.N_b + 2.b + b.(3.i_b + 3.j_b) + i_l - b.(i_b - j_b) - j_l \\ &\Leftrightarrow \quad 0 \leq 6.N_b + 2 + 2.i_b + 4.j_b + \left\lfloor \frac{i_l - j_l}{b} \right\rfloor \end{aligned}$$

where $-2b \leq i_l - j_l < 2b$*. Therefore* $k_1 = \left\lfloor \frac{i_l - j_l}{b} \right\rfloor \in [| -2, 1 |]$*. For* $k_1 = -1$ *and* 1*, the equality constraint* $6.N_b + 2.i_b + 4.j_b + 2 + k_1$ *is not satisfied (because of the parity of its terms), thus the corresponding polyhedron are empty.*

Let us examine the second constraint of the polyhedron.

$$\begin{aligned} i + j \leq N \quad &\Leftrightarrow \quad 0 \leq 6.b.N_b + 2.b - b.(3.i_b + 3.j_b) - i_l - L.b.(i_b - j_b) - j_l \\ &\Leftrightarrow \quad 0 \leq 6.N_b + 2 - 4.i_b - 4.j_b + \left\lfloor \frac{-i_l - j_l}{b} \right\rfloor \end{aligned}$$

where $-2b \leq -i_l - j_l < 2b$*. Therefore* $k_2 = \left\lfloor \frac{-i_l - j_l}{b} \right\rfloor \in [| -2, 1 |]$*. For the same reason as the previous constraints,* $k_2 = -1$ *and* 1 *leads to empty polyhedra.*

Let us examine the third constraint of the polyhedron.

$$\begin{aligned} 0 \leq j - 1 \quad &\Leftrightarrow \quad 0 \leq b.(i_b - j_b) + j_l - 1 \\ &\Leftrightarrow \quad 0 \leq i_b - j_b + \left\lfloor \frac{j_l - 1}{b} \right\rfloor \end{aligned}$$

where $-b \leq j_l - 1 < b$*. Therefore* $k_3 = \left\lfloor \frac{j_l - 1}{b} \right\rfloor \in [| -1, 0 |]$

Therefore, we obtain a union of $2 \times 2 \times 2 = 8$ *polyhedra, which are the result of the following intersection:*

$$\begin{aligned} &\begin{bmatrix} \{i_b, j_b, i_l, j_l | 0 \leq 6.N_b + 2.i_b + 4.j_b \wedge (i_l, j_l) \in \mathcal{T}_b\} \\ \uplus \{i_b, j_b, i_l, j_l | 0 = 6.N_b + 2.i_b + 4.j_b + 2 \wedge (i_l, j_l) \in \mathcal{T}_b \wedge 0 \leq i_l - j_l\} \end{bmatrix} \\ \cap &\begin{bmatrix} \{i_b, j_b, i_l, j_l | 0 \leq 6.N_b - 4.i_b - 4.j_b \wedge (i_l, j_l) \in \mathcal{T}_b\} \\ \uplus \{i_b, j_b, i_l, j_l | 0 = 6.N_b - 4.i_b - 4.j_b + 2 \wedge (i_l, j_l) \in \mathcal{T}_b \wedge 0 \leq -i_l - j_l\} \end{bmatrix} \\ \cap &\begin{bmatrix} \{i_b, j_b, i_l, j_l | 0 \leq i_b - j_b - 1 \wedge (i_l, j_l) \in \mathcal{T}_b\} \\ \uplus \{i_b, j_b, i_l, j_l | 0 = i_b - j_b \wedge (i_l, j_l) \in \mathcal{T}_b \wedge 0 \leq j_l - 1\} \end{bmatrix} \end{aligned}$$

6.2 General monoparametric partitionning of affine functions

Let us consider an affine function $f : (\vec{i} \mapsto Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q})$ and two partitionings: one for the input indices and one for the output indices (denoted with primes). Note that the "tile shapes" in the source and destination dimensions, $\mathcal{P}_b$ and $\mathcal{P}'_b$ might be different. Let us show how to adapt the derivation of Theorem 4.4 to these general partitioning.

Theorem 6.3. *Given two general monoparametric partitioning transformations ($\mathcal{T}_b$ and $\mathcal{T}'_b$) and any affine function ($f(\vec{i}) = Q.\vec{i} + Q^{(p)}.\vec{p} + \vec{q}$) the composition $\mathcal{T}'_b \circ f \circ \mathcal{T}_b^{-1}$ is a piecewise affine function, whose branches are of the form:*

$$\phi(\vec{i_b}, \vec{i_l}) = \begin{pmatrix} L'^{-1}.Q.D.\vec{i_b} + L'^{-1}.Q^{(p)}.\vec{p_b} + \vec{k} - \vec{k'} \\ Q.\vec{i_l} + Q^{(p)}.\vec{p_l} + \vec{q} + b.L'(\vec{k'} - \vec{k}) \end{pmatrix}$$
$$\textit{if} \begin{cases} b.\vec{k} \leq L'^{-1}.Q.\vec{i_l} + L'^{-1}.Q^{(p)}.\vec{p_l} + L'^{-1}.\vec{q} < b.(\vec{k} + \vec{1}) \\ Q.\vec{i_l} + Q^{(p)}.\vec{p_l} + \vec{q} + b.L'(\vec{k'} - \vec{k}) \in \mathcal{P}'_b \\ \vec{i_l} \in \mathcal{P}_b \end{cases}$$

for each $\vec{k} \in [|\vec{k}^{\min}; \vec{k}^{\max}|]$, for each $\vec{k'} \in [|\vec{k'}^{\min}; \vec{k'}^{\max}|]$, where L, L' are basis of the lattice of tile origins of respectively $\mathcal{T}$ and $\mathcal{T}'$, and assuming that $(L'^{-1}.Q.L)$ and $(L'^{-1}.Q^{(p)})$ are integer matrices.

Proof. Starting from the definition of f, we can perform the same manipulation as in the proof of Theorem 4.4 to obtain:

$$\vec{i_b}' + \left\lfloor \frac{L'^{-1}.\vec{i_l}'}{b} \right\rfloor = \left\lfloor L'^{-1}.Q.L.\vec{i_b} + L'^{-1}.Q^{(p)}.\vec{p_b} + \frac{L'^{-1}.(Q.\vec{i_l} + Q^{(p)}.\vec{p_l} + \vec{q})}{b} \right\rfloor$$

However, in the case of a non-rectangular partitioning, $\vec{k'}(\vec{i_l}') = \left\lfloor \frac{L'^{-1}.\vec{i_l}'}{b} \right\rfloor$ is not always equal to 0, so we cannot eliminate it, as in the rectangular case [2]. However, because $\vec{i_l}' \in \mathcal{P}'_b$, $\vec{k'}(\vec{i_l}')$ only takes a finite number of values, whose extremal values can be determined through a linear programming solver.

For each value $\vec{k'}$ of $\vec{k'}(\vec{i_l}')$, we can perform the same analysis as in Theorem 4.4. The new necessary and sufficient condition to avoid modulo functions is that the matrices $L'^{-1}.Q.L$ and $L'^{-1}.Q^{(p)}$ are integral. After defining $\vec{k}(\vec{i_l}) = \left\lfloor \frac{L'^{-1}.(Q.\vec{i_l}+Q^{(p)}.\vec{p_l}+\vec{q})}{b} \right\rfloor$, we obtain a piecewise affine function in which each branch corresponds to a different value of $\vec{k}(\vec{i_l})$.

Now, we gather the constraints for each branches. From the definition of $\vec{k}$, we obtain the following constraint:

$$b.\vec{k} \leq L'^{-1}.Q.\vec{i_l} + L'^{-1}.Q^{(p)}.\vec{p_l} + L'^{-1}.\vec{q} < b.(\vec{k} + \vec{1})$$

From the definition of $\vec{k'}$ and after substituting $\vec{i_l}'$ by its value, we obtain the following constraint

$$b.\vec{k'} \leq L'^{-1}.(Q.\vec{i_l} + Q^{(p)}.\vec{p_l} + \vec{q} - b.L'.\vec{k}) < b.(\vec{k'} + \vec{1})$$

However, after simplification, we obtain exactly (and surprisingly) the same constraint than we got from the definition of $\vec{k}$. The two remaining constraints are $\vec{i_l} \in \mathcal{P}_b$ and $\vec{i_l}' \in \mathcal{P}'_b$ (in which $\vec{i_l}'$ can be substituted by its value).

Finally, we regroup all the branches derived for every $\vec{k'}$ to form the partitioned piecewise affine function corresponding to f. □

[2]Intuitively, $\vec{k'} = \vec{0}$ means that $\vec{i_l}'$ belongs to the parallelepiped $\{L'.\vec{z} | \vec{0} \leq \vec{z} < \vec{1}\}$. For a rectangle tile, this is always the case, but for the hexagonal tile shown in Figure 9, it only corresponds to the portion of the hexagon between the two red arrows

Note that the new condition to avoid modulo is now that ($L'^{-1}.Q.L$ and $L'^{-1}.Q^{(p)}$ are integral. The main reason behind our generalization is that this condition depends only on the lattices of tile origins, and the coefficient matrix of the polyhedron, but does not involve any symbolic values.

Example 6.4. *Let us consider the identity affine function* $(i, j \mapsto i, j)$*, and let us consider the two following partitionings:*

- *For the antecedent space, we choose an hexagonal tiling:*
  - $\mathcal{T}_b = \{i, j \mid -b < j \leq b \ \wedge\ -2b < i + j \leq 2b \ \wedge\ -2b < j - i \leq 2b\}$
  - $L_b = L.b.\mathbb{Z}^2$ *where* $L = \begin{bmatrix} 3 & 3 \\ 1 & -1 \end{bmatrix}$
- *For the image space, we choose a rectangular tiling, with the same lattice:*
  - $\mathcal{T}'_b = \{i, j \mid 0 \leq i < 3b \ \wedge\ 0 \leq j < 2b\}$
  - $L'_b = L'.b.\mathbb{Z}^2$ *where* $L' = \begin{bmatrix} 3 & 3 \\ 1 & -1 \end{bmatrix}$

The derivation goes as follow:

$$\begin{aligned} & \begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} i \\ j \end{bmatrix} \\ \Leftrightarrow\ & L'.b.\begin{bmatrix} i'_b \\ j'_b \end{bmatrix} + \begin{bmatrix} i'_l \\ j'_l \end{bmatrix} = L.b.\begin{bmatrix} i_b \\ j_b \end{bmatrix} + \begin{bmatrix} i_l \\ j_l \end{bmatrix} \\ \Leftrightarrow\ & \begin{bmatrix} i'_b \\ j'_b \end{bmatrix} + L'^{-1}.\tfrac{1}{b}.\begin{bmatrix} i'_l \\ j'_l \end{bmatrix} = \begin{bmatrix} i_b \\ j_b \end{bmatrix} + L'^{-1}.\tfrac{1}{b}.\begin{bmatrix} i_l \\ j_l \end{bmatrix} \end{aligned}$$

Because $L'^{-1} = \frac{1}{6}.\begin{bmatrix} 1 & 3 \\ 1 & -3 \end{bmatrix}$*, then the constraints become:*

$$\begin{cases} i'_b + \frac{i'_l + 3.j'_l}{6b} = i_b + \frac{i_l + 3.j_l}{6b} \\ j'_b + \frac{i'_l - 3.j'_l}{6b} = j_b + \frac{i_l - 3.j_l}{6b} \end{cases}$$

After taking the floor of these constraints:

$$\begin{cases} i'_b + \left\lfloor \frac{i'_l + 3.j'_l}{6b} \right\rfloor = i_b + \left\lfloor \frac{i_l + 3.j_l}{6b} \right\rfloor \\ j'_b + \left\lfloor \frac{i'_l - 3.j'_l}{6b} \right\rfloor = j_b + \left\lfloor \frac{i_l - 3.j_l}{6b} \right\rfloor \end{cases}$$

We define $k'_1 = \left\lfloor \frac{i'_l + 3.j'_l}{6b} \right\rfloor$, $k_1 = \left\lfloor \frac{i_l + 3.j_l}{6b} \right\rfloor$, $k'_2 = \left\lfloor \frac{i'_l - 3.j'_l}{6b} \right\rfloor$ *and* $k_2 = \left\lfloor \frac{i_l - 3.j_l}{6b} \right\rfloor$*. After analysis of the extremal values of these quantities, we obtain:*

- $k_1 \in [|-1; 0|]$ *and* $k_2 \in [|-1; 0|]$
- $k'_1 \in [|0; 1|]$ *and* $k'_2 \in [|-1; 0|]$

Therefore, we obtain a piecewise affine function with 16 branches (one for each value of (k_1, k'_1, k_2, k'_2)*). Each branch have the following form:*

$$\begin{gathered} \left(i_b + k_1 - k'_1, j_b + k_2 - k'_2,\ \ i_l + 3b(k'_1 + k'_2 - k_1 - k_2), j_l + b(k'_1 + k_2 - k_1 - k'_2)\right) \\ \textit{when } 0 \leq i_l + 3b(k'_1 + k'_2 - k_1 - k_2) < 3b \ \wedge\ 0 \leq j_l + b(k'_1 + k_2 - k_1 - k'_2) < 2b \\ k_1.b \leq i_l + 3j_l < (k_1 + 1).b \ \wedge\ k_2.b \leq i_l - 3j_l < (k_2 + 1).b \\ -b < j_l \leq b \ \wedge\ -2b < i_l + j_l \leq 2b \ \wedge\ -2b < j_l - i_l \leq 2b \end{gathered}$$

6.3 General monoparametric partitionning program transformation

Now that we have extended the closure properties for the polyhedron and affine function, we can apply in a similar way the general monoparametric partitioning transformation to a complete polyhedral program. We now show how to extend the compatibility algorithm to such case of general tiles.

In Section 5, we have manipulated rectangular tile sizes $D.b$, which correspond to a special case of the lattice of tile origins, were the vector of the basis are canonic. In the general case, we manipulate lattice bases, whose vectors are the columns of an invertible matrix L. The constraints to avoid modulo conditions are of the form "the matrix $L'^{-1}.Q.L$ is integral," which is the same as saying that the antecedent lattice $Q.L.\mathbb{Z}^n$ is a subset of the image lattice $L'.\mathbb{Z}^m$. For similar reasons as in the canonic case, we want to select the lattice of minimal basis, i.e., to minimize the size of the considered tiles.

The same derivation algorithm can be adapted to affine lattices:

- For a normal edge in the PRDG, corresponding to $\langle S, \vec{i}\rangle = g(\langle T_1, f_1(\dots, \langle T_k, f_k(\vec{i})\dots\rangle)$, and assuming f_k is of the form $f_k : (\vec{i} \mapsto Q_k.\vec{i} + Q_k^{(p)}.\vec{p} + \vec{q_k})$, the constraints that must be satisfied by the lattice of tile origins of S are:
$$\begin{cases} (\forall 1 \leq k \leq d)\ L_{T_k}^{-1}.Q_k.L_S \text{ is integer} \\ (\forall 1 \leq k \leq d)\ L_{T_k}^{-1}.Q_k^{(p)} \text{ is integer} \end{cases}$$
One again, we can drop the second constraint. The first constraint means that all the lattices $L_{T_k}^{-1}Q_k.L_S.\mathbb{Z}^n$ are subsets of the lattice $\mathbb{Z}^m$. Therefore, the lattices $Q_k.L_S.\mathbb{Z}^n$ are a subset of the lattices $L_{T_k}.\mathbb{Z}^m$, $1 \leq k \leq d$. Let us define the affine functions $u_k : (\vec{z} \mapsto Q_k.\vec{z})$. The lattices $u_k(L_S.\mathbb{Z}^n)$ are a subset of the lattices $L_{T_k}.\mathbb{Z}^m$, $1 \leq k \leq d$. Because $[u(A) \subset B \Rightarrow A \subset u^{-1}(B)]$, this constraints means that the lattice $L_S.\mathbb{Z}^n$ is a subset of all the preimage of the lattice $L_{T_k}.\mathbb{Z}^m$ by the affine function u_k. Therefore, we have:
$$L_S.\mathbb{Z}^n \subset \bigcap_{1 \leq k \leq d} u_k^{-1}(L_{T_k}.\mathbb{Z}^m)$$
We can compute the right affine lattice, then take any of its basis as the value of L_S. There is no constraints on the tile shape for S, thus we can select any one we want.

- If the statement is a reduction:
$$\langle S, \vec{i}\rangle = \bigoplus_{\substack{\vec{i} = \pi(\vec{j}) \\ \vec{j} \in \mathcal{D}}} g(\langle T_0, f_0(\vec{j})\rangle, \dots, \langle T_d, f_d(\vec{j})\rangle).$$
For similar reasons as developed in Section 5, we assume that the projection function is canonic. When we tile the projection function, if the result is a piecewise affine function, the affine function inside a branch might not admit an integer right inverse. One way, to be sure that the resulting projection function will be correct, is to force the tile shape of the subexpression of the reduction to be hyperrectangular. Indeed, for such a form, projecting along a canonic dimension is trivial, but this forces the tile shape of statement S to be a rectangle. A more general way is to force the shape of the subexpression to be an orthogonal prism, whose base is the tile shape of S and which spans across the projected dimensions.

7 From Blocking to Tiling: Applying the Theory

In this section we illustrate how the theory we developed can be used to cleanly separate the concerns that in the past have all been addressed in an ad hoc, combined manner. Note that with nearly thirty years of research on tiling, our goal is not to reinvent the wheel and propose new tiling strategies, nor to develop

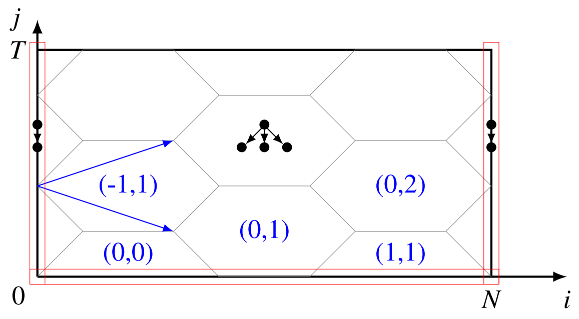


Figure 10: Iteration space of a Jacobi 1D computation with a hexagonal tiling. The border cases of the computation are underlined in red. The tile coordinates (i_{bl}, j_{bl}) are in blue.

sophisticated parallelization algorithms. Rather, we illustrate the flexibility of our approach by applying monoparametric partitioning on a simple example.

Note that partitioning is not a *tiling*, it is a simple re-indexing transformation that does not change the semantics of the original program. This allowed us to ignore the issue of legality. Indeed, partitioning transformations could also be applied to a program description that does not include any schedule or memory information such as that of a program in the polyhedral equational language, Alpha [32, 31] used in our system, **AlphaZ**. This provides a legal program representation, in the sense that its equaltional semantics are unchanged. Evidently, its*interpretation as a tiled imperative program* would be illegal.

The original Jacobi 1D computation can be described as follows.

$$\begin{aligned} Out[i] &= temp[i,T]; \\ temp[i,t] &= \begin{cases} In[i] & \text{if } t = 0 \\ temp[i,t-1] & \text{if } i = 0 \text{ or } N \\ (temp[i-1,t-1] + temp[i,t-1] & \text{otherwise} \\ \quad +temp[i+1,t-1])/3 & \end{cases} \end{aligned}$$

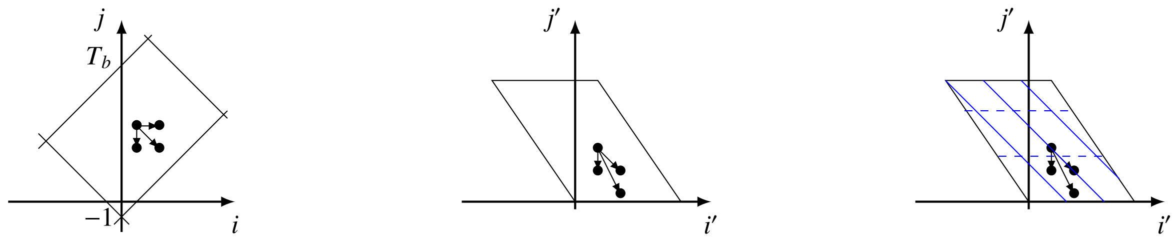
Let us tile this computation using hexagonal tiling, say, as suggested by Grosser et al. [21] and as shown in Figure 10. The tile shape is the following:

$$\mathcal{P}_b = \left\{ i_l, t_l \middle| \begin{array}{c} -b < t_l \leq b \\ t_l \leq i_l < t_l + 4b \\ -t_l < i_l \leq 4b - t_l \end{array} \right\}$$

7.1 Wavefront Parallelization of the "Outer" Program

Once the tile shape is chosen, applying our monoparametric partitioning will provide us with two separate polyhedral program descriptions, one for the computations within each tile (there are finitely many of them, corresponding to the interior tiles and the different kinds of boundary tiles). We also have another polyhedral program describing the set of tiles, and the dependences between the tiles. We call this the *outer* polyhedral program, and now analyze this, without concerning ourselves, for now, with how individual tiles are compiled, and parallelized. Since it is now clear from the context, we also drop the subscripts on the indices, but use N_b and T_b to denote the size parameters, $\frac{N}{b}$ and $\frac{T}{b}$. The iteration space of the outer polyhedral program is

$$\{i, j \mid -3 \leq 3i + 3j \leq N_b \wedge -1 \leq j - i \leq T_b\}$$



(a) Original tile space (b) After wavefront selection and skewing (c) After monoparametric partitioning for the multipass strategy

Figure 11: Tile space of the hexagonal J1D computation, skewing transformation and monoparametric partitioning for the multipass strategy

and the inter-tile dependences are as follows: tile (i, j), depends on the three tiles $(i, j-1)$, $(i+1, j-1)$ and $(i+1, j)$. This is illustrated in Figure 11a.

A classical analysis performed after tiling is the wavefront selection, which aims at finding a parallel schedule between tiles. Indeed, one of the touted advantages of hexagonal tiling of J1D is that it enables concurrent start. For example, in Figure 11a, we can use $(-1, 1)$ as the (normal vector of the) scheduling wavefront, i.e., we use $t(i, j) = j - i$ as the time stamp for tile (i, j), executing the tiles in bottom-right to top-left sweeps. To implement this schedule, we can apply a skewing transformation $(i, j \mapsto i, j - i))$ so that the new vertical axis j' can be interpreted as the time. The transformed domain is showed in Figure 11b and is described by the following set.

$$\{i', j' \mid -3 \leq 6.i' + 3.j' \leq N_b \wedge -1 \leq j' \leq T_b\}$$

Notice that this wavefront selection and the corresponding skewing transformation works exactly in the same way for fixed size tiling.

7.2 Multipass Parallelization: Improving Energy Efficiency

Zou and Rajopadhye proposed a strategy for energy optimization of stencil computations [50]. They suggest that by parallelizing the tiles with multiple passes, a large fraction of the memory footprint of the computation can be retained in the last level of cache. We seek to apply this strategy, but with our hexagonal tiling, since we do not want to lose the advantages of concurrent start. So we start with the computations described in Figure 11b.

A *pass* is a tiling where all but one of the dimensions is tiled, thereby dividing the domain in to bands or passes, that are typically executed in sequence. So we seek to apply a second level of monoparametric partitioning. The pass boundary must be legal tiling hyperplanes, and we choose $i' + j' =$ const, the blue solid lines in Figure 11c. However, we cannot monoparametrically block this single dimension, because the tile space has a constraint involving both i' and j'.

However, *tiling is not the same as partitioning*. We first apply a second level of monoparametric partitioning on the whole space (see the blue solid lines and the horizontal, dashed lines in Figure 11c), which is possible because we block all the dimensions. Then, we use just one of these dimensions to describe the tiling into passes: if the newly introduced indexes are i'_b, i'_l, j'_b and j'_l, we use i'_b as the pass dimension. Then, a pass will simply be a column of blocks and will be living in the 3 dimensional space corresponding to (i'_l, j'_b, j'_l).

Finally, note that whatever strategy we choose, we use a single polyhedral code generator, e.g., CLooG [6] to produce the final codes.

7.3 Revisiting Legality Conditions

As we have seen, monoparametric blocking is (merely) a blocking, replacing the original indices by corresponding block and local indices. Unlike tiling, it *does not*, in and of itself, modify the schedule of a program. For example if $(i, j \mapsto i, j)$ was a valid schedule of a statement before transformation, and we apply monoparametric blocking to it, obtaining the new indices, i_b, j_b, i_l, j_l, then the schedule $(i_b, j_b, i_l, j_l \mapsto i_b, i_l, j_b, j_l)$ to is exactly the same schedule, but expressed with the new indices. This would be like doing strip mining of each of the original loops. We have already seen how keeping this view, for a subset of the indices allowed us to obtain the multipass parallelization.

In general, if we want to go beyond blocking and actually tile the program with atomic tiles, we need to select a multidimensional schedule whose first (outer) dimensions use only block indices, and whose inner dimensions use the local indexes (e.g., $(i_b, j_b, i_l, j_l \mapsto i_b, j_b, i_l, j_l)$). In that case, we need to check for the tiling legality conditions, similar to conventional tiling transformations.

We emphasize the fact that, even though monoparametric blocking requires that we block all dimensions, but we **do not** need to *tile* all of them. For example, if we want to tile only across the i dimension, we can choose the following schedule: $(i_b, j_b, i_l, j_l \mapsto i_b,\ i_l, j_b, j_l)$ and consider that the i_bth tile is in the three dimensions (i_l, j_b, j_l).

Legality condition of tiling Because tiles are executed atomically, we need to check that there is no cycle between them. In our program representation, dependence functions are explicit. Thus, when we apply monoparametric blocking to a dependence function, because we introduce the block and local indexes for the image space of this function, the tiles which might be accessed by this dependence function appear explicitly. Thus, after blocking a statement, we know precisely, the inter-tile dependences.

Consider the so called "tile graph", i.e., the dependence graph of the "outer program". The legality condition of tiling is satisfied iff the tile graph has no cycles. Thus, the legality condition of tiling is equivalent to the existence of a valid schedule for the outer program. Therefore, we can check for legality by using a scheduler [16, 8]: if a schedule is found, then the tiling is legal. Moreover, we obtain information about the parallelism across tiles, which might be needed to generate efficient code. Of course there are often more efficient methods.

Tile groups, reduction and mostly-tileable loops Note that the monoparametric blocking as we have defined it, is applied on a per-statement basis. For many applications, tiling multiple statements together is useful. Indeed, if they are interdependent and create cycles between tiles if they are tiled separately, they *must* be tiled together. Hence, we define the notion of a *tile group*, as a collection of statements that should be tiled together. All statements of a tile group must be of the same dimension (so that the tile space is common), which might require to "massage" the domain of some statements beforehand.

We also note that reductions introduce new dimensions in their body, which do not exist at the level of the statement. These dimensions are also blocked by our transformation, along with the projection function. Thus, a reduction is summing across multiple tiles, along its projected direction. For example, if we consider a matrix multiplication statement: `C[i,j] =` $\sum_k$ `A[i,k] * B[k,j]`, after applying the monoparametric blocking transformation, we obtain the following statement:

$$C[i_b, j_b, i_l, j_l] = \sum_{k_b, k_l} A[i_b, k_b, i_l, k_l] * B[k_b, j_b, k_l, j_l]$$

To differentiate the different blocks summed by this reduction, we introduce a new variable called

TempRed, defined as the intermediate result of the reduction on each block. In our example:

$$\begin{array}{l} C[i_b, j_b, i_l, j_l] = \sum_{k_b} TempRed[i_b, j_b, k_b, i_l, j_l] \\ TempRed[i_b, j_b, k_b, i_l, j_l] = \\ \qquad \sum_{k_l} A[i_b, k_b, i_l, k_l] * B[k_b, j_b, k_l, j_l] \end{array}$$

Because we just have introduce a new variable, we should decide if we should include them in an already existing tile group, or to form a new tile group. A possible strategy consists to create as many tiles as possible, while satisfying the tiling legality conditions. Thus, we form a new tile group composed of all the tiles of TempRed which do not have a cyclic dependence with existing tiles (and, in particular, its originating tile group).

A similar problem was studied for fixed size tiling by Wonnacott [12] on Nussinov's algorithm. This algorithm has the following pattern:

$$N[i, j] = max_{i \leq k < j}(N[i, k] + N[k + 1, j])$$

where "max" is the reduction operator. After applying a monoparametric transformation on this program using square blocks, we can remark that, among the tiles which are reduced together, the two extremal tiles depend on the tile of N currently computed, and all the middle tiles can be executed independently. It is possible to retrieve this information automatically by analyzing the part of the dependences corresponding to the blocked dimensions, and deduce that the only tiles of the reduction which have a cyclic dependence are the two extremal tiles.

8 Discussions

Blocking all dimensions Notice that we also have to block everything in all the dimensions. Indeed, as soon as an index interacts with another one which is blocked (inside an index expression of a polyhedron or an affine function), then we have to block both of them. In some situations, the indexes are separated and we can block only a subset of the indexes (for example, for matrix multiplication, we can only tile one dimension on the three). Another possibility is to block each non-interacting group of dimensions with a different block size parameter (for example, we can have blocks of size $b_1 \times b_2 \times b_3$ for matrix multiplication, because there is no constraints or affine function in the program which involves more than one index).

Data layout When applying the monoparametric transformation to a program, the number of dimension of all spaces are doubled. In particular, if we have inputs and outputs to our program, their number of dimensions is changed. For example, for matrix multiplication, `C[i,j]` is originally a 2D array which becomes a 4D array `Cbl[ibl, jbl, iloc, jloc]` after blocking. So that we keep the same number of dimensions than the original program, we can use a non-affine macro `Cbl[ibl, jbl, iloc, jloc] = C[ibl * d`$_1$`*b + iloc, jbl * d`$_2$`*b + jloc]` where $d_1.b \times d_2.b$ are the block sizes of the variable C.

Fixed-size partitionning It is possible to obtain a fixed-size partitionned code by using the same process than monoparametric tiling. For example, if we want to apply a rectangular partitionning with constant tile sizes $t_1 \times t_2 \times \cdots \times t_n$, we can take $b = gcd(t_1, t_2, \dots, t_n)$ and the ratio $D = Diag(\frac{t_1}{b}, \frac{t_2}{b}, \dots, \frac{t_n}{b})$.

Adaptation between different tiling Let us consider an identity function ($\vec{i} \mapsto \vec{i}$), with different tiling in both side of the function. By computing the monoparametric partionned version of this function, we obtain a piecewise affine function which can be used to adapt the indexes of two statements with two different tiling. For example, we can theoreticaly mix hexagonal and rectangular tiling in a program, if each one of them is best suited for the part of the program they manage.

Separation of block and local indices In the union of polyhedra and piecewise affine functions we compute through our transformation, the constraints on the block and local indexes are separated, i.e., no constraints involve both kind of indexes. Using this property, we can normalize the program such that the conditions on the block indexes are gathered at the top of the program in a single switch. This is useful to produce efficient code later, because it reduces the control flow inside a program [19]. Moreover, it allows us to "type" the blocks of the program, i.e., to distinguish which set of blocks are performing the same computations. There is, by construction, only a finite number of computation which might be performed by a block.

Benefit of monoparametric partitionning Monoparametric tiling is a polyhedral transformation which offers a restricted form of parametrization of the tile sizes. Compared to fixed-size tiling, we claim that there is *no drawback* of using monoparametric tiling instead of a fixed-size tiling. Indeed, because we remain also inside the polyhedral model, we can obtain the same informations than for the fixed-size tiling transformation, and perform the same kind of optimizations. Thus, theoretically, any code obtainable through fixed-size tiling should also be obtainable through monoparametric tiling, with an additional parameter for the block sizes.

Compared to parametric tiling, monoparametric tiling is more restricted in the kind of tile size a single code can explore. Indeed, the shape of the tiles is fixed at compile time, and we are forced to regenerate the code if we want to change it. Moreover, the machinary required to apply the monoparametric tiling is heavier than a classical parametric tiling. However, because this transformation is polyhedral, we gain in modularity of the code generation strategies, and we do not need to embed this transformation inside a code generator.

Extension to parametric tiling We notice that if a group of indices is isolated from the indices we need to partition, we do not need to partition them. For example, in the classic matrix multiplication computation (which is a 3D computation), we can partition only one computation without having to partition the other ones, because the different dimensions do not interact with each other.

Using a similar reasoning, it is possible to use different tile size parameters $b_1, b_2, \dots$ for groups of indices which do not interact with each other. Thus, we are able to obtain a fully parametrized tiled code through CART, while staying polyhedral. However, as soon as two indices with different tile size parameter interact with each other, when we try to apply CART, we obtain terms of the form $\left\lfloor \frac{b_1}{b_2} \right\rfloor$, which is impossible to manage.

To have an intuition of why it cannot work, let us consider a polyhedron containing only the constraint $i + j \leq 0$, and let us see what happens when we try to tile it with a parametric tile size $b_1 \times b_2$. As shown by Figure 12, there are two reasons why the result cannot be expressed in the polyhedral model:

- Let us estimate the number of different tile shapes on the diagonal. The constraints $i \leq j$ goes through the integer point $(0, 0)$, and we can show that the next integer point it is going through is $(lcm(b_1, b_2), lcm(b_1, b_2))$ where $lcm(x, y)$ is the least common multiple of x and y. Thus, we have $O(lcm(b_1, b_2)/b_1) \approx O(b_1 + b_2)$ different type of tiles.
- If we consider the shape of the diagonal tile $(i_b, j_b) = (0, -1)$, this shape happens for every tile (i_b, j_b) such that $b_1.i_b + b_2(j_b + 1) = 0$, which is not an affine constraint.

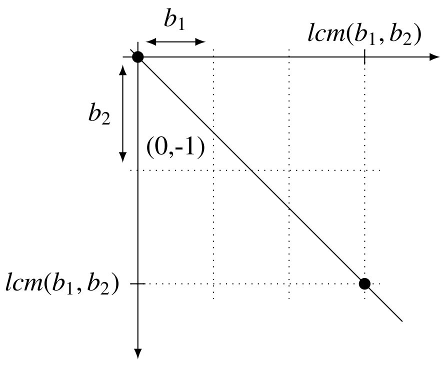


Figure 12: Parametric tiling with $b_1 \times b_2$ rectangular tile sizes on the polyhedron $i + j \leq 0$

Also note that the constraints of the shapes themselves are polyhedral (ex: $ii + jj \leq b_2$ for the diagonal tile $(i_b, j_b) = (0, -1)$). Therefore, it is not possible to express this union as a polyhedral union, even if it might be possible to exploit the fact that each shape is polyhedral.

Because the general parametric tiling transformation is not linear, the current polyhedral compilers hard-code this transformation in their code generator. Thus, they lose in flexibility and cannot apply polyhedral analysis and transformations afterward. Because CART remains in the polyhedral model, we are still able to perform these optimizations afterward. For example, after applying CART once, we can tile the local indices again (which are separated from the blocked indices) to introduce a new level of parametric tiling for free. Or we can devise a new code generation strategy which fits a new architecture without having to redesign and reimplement a full code generator.

9 Conclusion

In this paper, we introduce mono-parametric tiling, a parametric tiling transformation in which tile sizes are multiple of the same block parameter. We first consider the hyperrectangular blocks, and show that the underlying blocking is a polyhedral transformation, before showing how to apply it to complete programs. Then, we extend this blocking transformation to any block shape that is a homothethic scalling of a fixed size tile shape. Finally, we demonstrate the flexibility of code generation strategy after the monoparametric tiling transformation, by showing different polyhedral transformation and analysis which can be easily applied afterward.

The mono-parametric tiling transformation is more general than fixed size tiling, because of the additional parameter for the tile sizes. Compared to parametric tiling, this transformation is less general (the tile shape cannot be changed) and requires a more complicated mathematical machinery, but we are much more flexible in the kind of analysis and transformations we can apply after tiling.

References

[1] V. Adve, G. Jin, J. Mellor-Crummey, and Q. Yi. High Performance Fortran compilation techniques for parallelizing scientific codes. *SC Conference*, 0:11, 1998.

[2] J. Anderson and M. Lam. Global optimizations for parallelism and locality on scalable parallel machines. In *The ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, pages 112–125, Albuquerque, N.M., Jun 1993. ACM Press.

[3] R. Andonov, S. Balev, S. Rajopadhye, and N. Yanev. Optimal semi-oblique tiling. *IEEE Transactions on Parallel and Distributed Systems*, 14(9):944–960, Sept 2003.

[4] Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula. TilingStencil Computations to Maximize Parallelism. In *the International conference on high performance computing, networking, storage and analysis*, SC '12, pages 40:1–40:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[5] M. M. Baskaran, A. Hartono, S. Tavarageri, T. Henretty, J. Ramanujam, and P. Sadayappan. Parameterized tiling revisited. In *CGO*, pages 200–209, 2010.

[6] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *PACT'13: IEEE International Conference on Parallel Architectures and Compilation and Techniques*, pages 7–16, Juan-les-Pins, September 2004.

[7] Ian J. Bertolacci, Catherine Olschanowsky, Ben Harshbarger, Bradford L. Chamberlain, David G. Wonnacott, and Michelle Mills Strout. Parameterized diamond tiling for stencil computations with chapel parallel iterators. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, ICS '15, pages 197–206, New York, NY, USA, 2015. ACM.

[8] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. Pluto: A practical and fully automatic polyhedral program optimization system. In *ACM Conference on Programming Language Design and Implementation*, pages 101–113, Tuscon, AZ, June 2008. ACM SIGPLAN.

[9] David Callahan, John Cocke, and Ken Kennedy. Estimating interlock and improving balance for pipelined architectures. *Journal of Parallel and Distributed Computing*, 5(4):334 – 358, 1988.

[10] A. Darte and Y. Robert. Constructive methods for scheduling uniform loop nests. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):814–822, Aug 1994.

[11] A. Darte and Y. Robert. Affine-by statement scheduling of uniform and affine loop nests over parametric domains. *Journal of Parallel and Distributed Computing*, 29(1):43–59, February 1995.

[12] Allison Lake David Wonnacott, Tian Jin. Automatic tiling of "mostly-tileable" loop nests. In *IMPACT 2015: 4th International Workshop on Polyhedral Compilation Techniques*, Amsterdam, the Netherlands, Jan 2015.

[13] Hikmet Dursun, Ken-ichi Nomura, Weiqiang Wang, Manaschai Kunaseth, Liu Peng, Richard Seymour, Rajiv K. Kalia, Aiichiro Nakano, and Priya Vashishta. In-Core Optimization of High-Order Stencil Computations. In Hamid R. Arabnia, editor, *PDPTA*. CSREA Press, 2009.

[14] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationelle*, 22(3):243–268, Sep 1988.

[15] P. Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53, Feb 1991.

[16] P. Feautrier. Some efficient solutions to the affine scheduling problem, Part II, multidimensional time. Technical Report 78, Labaratoire MASI, Institut Blaise Pascal, October 1992.

[17] Paul Feautrier. Some efficient solutions to the affine scheduling problem. Part I. one-dimensional time. *International Journal of Parallel Programming*, 21(5):313–347, 1992.

[18] Paul Feautrier. Some efficient solutions to the affine scheduling problem. Part II. multidimensional time. *International Journal of Parallel Programming*, 21(6):389–420, 1992.

[19] G. Goumas, M. Athanasaki, and N. Koziris. An efficient code generation technique for tiled iteration spaces. *Parallel and Distributed Systems, IEEE Transactions on*, 14(10):1021–1034, Oct 2003.

[20] Tobias Grosser, Albert Cohen, Justin Holewinski, P. Sadayappan, and Sven Verdoolaege. Hybrid hexagonal/classical tiling for GPUs. In *CGO*, page 66, Orlando, FL, Feb 2014.

[21] Tobias Grosser, Sven Verdoolaege, Albert Cohen, and P. Sadayappan. The relation between diamond tiling and hexagonal tiling. *Parallel Processing Letters*, 24(3), 2014.

[22] A. Hartono, M. M. Baskaran, C. Bastoul, A. Cohen, S. Krishnamoorthy, B. Norris, J. Ramanujam, and P. Sadayappan. Parametric Multi-level Tiling of Imperfectly Nested Loops. In *in Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 147–157, New York, NY, USA, 2009. ACM.

[23] A. Hartono, M.M. Baskaran, J. Ramanujam, and P. Sadayappan. DynTile: Parametric Tiled Loop Generation for Parallel Execution on Multicore Processors. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–12, April 2010.

[24] Tom Henretty, Richard Veras, Franz Franchetti, Louis-Noël Pouchet, J. Ramanujam, and P. Sadayappan. A Stencil Compiler for Short-vector SIMD Architectures. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 13–24, New York, NY, USA, 2013. ACM.

[25] F. Irigoin and R. Triolet. Supernode partitioning. In *15th ACM Symposium on Principles of Programming Languages*, pages 319–328. ACM, Jan 1988.

[26] Daegon Kim. *Parameterized and Multi-Level Tiled Loop Generation*. PhD thesis, Colorado State University, Fort Collins, CO, USA, 2010.

[27] Toru Kisuki, Peter M. W. Knijnenburg, Michael F. P. O'Boyle, François Bodin, and Harry A. G. Wijshoff. A feasibility study in iterative compilation. In *Proceedings of the Second International Symposium on High Performance Computing*, ISHPC '99, pages 121–132, London, UK, UK, 1999. Springer-Verlag.

[28] Toru Kisuki, Peter M. W. Knijnenburg, Michael F. P. O'Boyle, and François Bodin. Combined selection of tile sizes and unroll factors using iterative compilation. In *PACT 2000: Parallel Architectures and Compilation Techniques*, pages 237–246, 2000.

[29] Athanasios Konstantinidis, Paul H. J. Kelly, J. Ramanujam, and P. Sadayappan. Parametric GPU code generation for affine loop programs. In *Languages and Compilers for Parallel Computing - 26th International Workshop, LCPC 2013, San Jose, CA, USA, September 25-27, 2013. Revised Selected Papers*, pages 136–151, 2013.

[30] Sriram Krishnamoorthy, Muthu Baskaran, Uday Bondhugula, J. Ramanujam, Atanas Rountev, and P Sadayappan. Effective Automatic Parallelization of Stencil Computations. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '07, pages 235–244, New York, NY, USA, 2007. ACM.

[31] H. Le Verge. *Un environnement de transformations de programmmes pour la synthèse d'architectures régulières*. PhD thesis, L'Université de Rennes I, IRISA, Campus de Beaulieu, Rennes, France, Oct 1992.

[32] C. Mauras. *ALPHA: un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones*. PhD thesis, L'Université de Rennes I, IRISA, Campus de Beaulieu, Rennes, France, December 1989.

[33] Liu Peng, R. Seymour, K. Nomura, R.K. Kalia, A. Nakano, P. Vashishta, A. Loddoch, M. Netzband, W.R. Volz, and C.C. Wong. High-Order Stencil Computations on Multicore Clusters. In *IEEE International Symposium on Parallel Distributed Processing (IPDPS), 2009.*, may 2009.

[34] F. Quilleré and S. Rajopadhye. Optimizing memory usage in the polyhedral model. *ACM Transactions on Programming Languages and Systems*, 22(5):773–815, September 2000.

[35] P. Quinton and V. Van Dongen. The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing*, 1(2):95–113, 1989.

[36] S. V. Rajopadhye, S. Purushothaman, and R. M. Fujimoto. On synthesizing systolic arrays from recurrence equations with linear dependencies. In *Proceedings, Sixth Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 488–503, New Delhi, India, December 1986. Springer Verlag, LNCS 241.

[37] J. Ramanujam and P. Sadayappan. Tiling multidimensional iteration spaces for multicomputers. *Journal of Parallel and Distributed Computing*, 16(2):108–120, 1992.

[38] X. Redon and P. Feautrier. Detection of scans in the polytope model. *Parallel Algorithms and Applications*, 15(3-4):229–263, 2000.

[39] Lakshminarayanan Renganarayanan, DaeGon Kim, Sanjay V. Rajopadhye, and Michelle Mills Strout. Parameterized tiled loops for free. In *PLDI*, pages 405–414, 2007.

[40] Lakshminarayanan Renganarayanan, Daegon Kim, Michelle Mills Strout, and Sanjay Rajopadhye. Parameterized loop tiling. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 34(1):3, 2012.

[41] R. Schreiber and J. Dongarra. Automatic blocking of nested loops. Technical Report 90.38, RIACS, NASA Ames Research Center, Aug 1990.

[42] Robert Strzodka, Mohammed Shaheen, Dawid Pajak, and Hans-Peter Seidel. Cache Accurate Time Skewing in Iterative Stencil Computations. In *Proceedings of the International Conference on Parallel Processing (ICPP)*. IEEE Computer Society, September 2011.

[43] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009.

[44] M. Wolf and M. Lam. Loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):452–471, Oct 1991.

[45] M. E. Wolf and M. Lam. A data locality optimizing algorithm. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Totonto, CA, june 1991.

[46] M. J. Wolfe. Iteration space tiling for memory hierarchies. *Parallel Processing for Scientific Computing (SIAM)*, pages 357–361, 1987.

[47] D. Wonnacott. Time skewing for parallel computers. In *LCPC 1999: 12th International Workshop on Languages and Compilers for Parallel Computing*, pages 477–480. Springer Verlag, 1999. See `http://ipdps.eece.unm.edu/2000/papers/wonnacott.pdf` for a more detailed version.

[48] D. Wonnacott. Achieving scalable locality with time skewing. *IJPP: International Journal of Parallel Programming*, 30(3):181–221, Jun 2002.

[49] Jingling Xue. *Loop Tiling for Parallelism*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[50] Y. Zou and S. Rajopadhye. Automatic energy efficient parallelization of uniform dependence computations. In *ICS 2015: International Conference on Supercomputing*, page to appear, Newport Beach, CA, June 2015. ACM.

Contents

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l’Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399