An Irredundant Decomposition of Data Flow with Affine Dependences

Corentin Ferry
Univ Rennes, CNRS, Inria, IRISA
& Colorado State University
Fort Collins, CO, USA
cferry@mail.colostate.edu

Steven Derrien
Univ Rennes, CNRS, Inria, IRISA
Rennes, France
steven.derrien@irisa.fr

Sanjay Rajopadhye
Colorado State University
Fort Collins, CO, USA
sanjay.rajopadhye@colostate.edu

ABSTRACT

Optimization pipelines targeting polyhedral programs try to maximize the compute throughput. Traditional approaches favor reuse and temporal locality; while the communicated volume can be low, failure to optimize spatial locality may cause a low I/O performance.

Memory allocation schemes using data partitioning such as data tiling can improve the spatial locality, but they are domain-specific and rarely applied by compilers when an existing allocation is supplied.

In this paper, we propose to derive a partitioned memory allocation for tiled polyhedral programs using their data flow information. We extend the existing MARS partitioning [7] to handle affine dependences, and determine which dependences can lead to a regular, simple control flow for communications.

This paper is a theoretical study that could potentially enhance previous work on data partitioning in inter-node scenarios that has shown improved bandwidth utilization.

1 INTRODUCTION

The performance of programs is determined by multiple metrics, among which execution time and energy consumption. One of the main drivers of these two metrics is data movement: communication latency causes bottlenecks that limit compute throughput, and interchip communication significantly increases power consumption. Optimizing data movement is a tedious task that involves significant modifications to the program and cries for automation. Powerful compiler analyses and abstractions have been developed for this, one of the most powerful of which is the polyhedral model.

In the polyhedral model, it is possible to entirely determine the execution sequence of a program, and its data movement. Optimizations are done in two ways: first, reducing the amount of communication by exhibiting *locality*; second, by optimizing the existing communications to reduce their latency and better utilize the available bandwidth.

Techniques improving bandwidth/access utilization [3, 5, 7] propose to decompose the data flowing between statements within a program and group together intermediate results based on their users; coalescing data accesses allows better bandwidth utilization. However, these data flow optimizations ignore input data, and address only on intermediate results. We need to also optimize input data transfers for locality and memory access performance.

Furthermore, dependences to input variables are rarely *uniform*, because the data arrays usually have fewer dimensions than the domain of computation.

IMPACT '24, January 17, 2024, Munich, Germany
.

This paper seeks to extend the partitioning of Ferry et al. [7] to handle the entire data flow of the tile and maximize access contiguity. Our contributions are as follows:

- We propose a partitioning scheme, called Affine-MARS, of data spaces and iteration spaces with a specified tiling,
- We formalize the construction of this partitioning scheme and determine its limitations.

We will first motivate the issues (Section 2). After giving the notions of MARS and the linear algebra concepts required to understand it (Section 3), we propose the construction methods for Affine-MARS according to the dependences (Section 4). Section 5 compares this approach to existing iteration- and data-space partitioning methods.

2 MOTIVATION

The motivation of this work stems from two driving forces: the necessity to exhibit data access contiguity, and the limitations of existing analyses preventing efficient (coalesced) memory accesses.

2.1 Necessity of spatial locality

To motivate this work, we can consider a matrix multiplication program. At each step of its computations, it needs input values ($a_{i,k}$ and $b_{k,j}$), an intermediate result (partial sum of $c_{i,j}$) and produces a new result. Previous work has shown that using loop tiling increases the performance due to improved locality. When tiling is applied, the matrices are processed in "patches" as illustrated in Figure 1.

In this application, multiplying matrices $A = (a_{i,j})$ and $B = (b_{i,j})$ is done by computing all $a_{i,k} \times b_{k,j}$. Loop tiling, for locality, can be applied and gives a division of the space as in Figure 1.

In this example, an entire patch of each input matrix A, B is transferred for the execution of each tile.

Despite the added locality, the application can still be memory-bound: tiled matrix product lacks data access contiguity. Barring any data layout manipulations, data is contiguous within a row (for row-major storage) or column (for column-major storage). A patch of A, B or C is never contiguous because it contains *multiple parts* of contiguous rows (or columns). The lack of contiguity therefore induces multiple short burst accesses to retrieve the entire patch.

As for intermediate results, it is desirable to increase spatial locality and leverage contiguity to obtain higher performance on the input variables. Not surprisingly, data blocking has been known to increase the performance of matrix multiplication, especially because when the blocks correspond to the "footprint" iteration tiles.

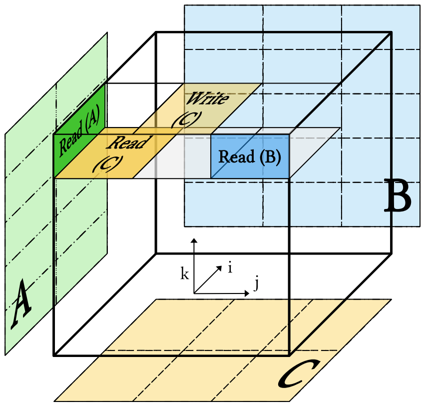

Figure 1: Tiled matrix product

2.2 Limitations of existing transformations

Although data tiling is sufficient for matrix multiplication, more complex computational patterns require a finer data partitioning.

Ferry et al. [7] proposes a breakup of intermediate results of programs with purely uniform dependence patterns, that enables contiguity. However, such dependence patterns exclude commonly found affine dependences, such as the *broadcast*-type dependences of matrix multiplication, despite the existence of natural solutions like data blocking.

Moreover, automatic data blocking is mostly applied by domain-specific compilers that del explicitly with the memory allocation (e.g. Halide [10], AlphaZ [15]). For input codes with pre-specified memory allocation and data layout, compilers follow it unless overridden by specific directives (e.g. the `ARRAY_PARTITION` directive in FPGA high-level synthesis tools based on [4]). Allowing the compiler to change this allocation would open the door to better bandwidth utilization. Work on inter-node communication in distributed systems [5, 17] where memory allocation only exists within nodes (not across nodes) can resort to very specific data grouping to minimize the communication overhead; it makes sense to apply this idea likewise to host-to-accelerator communications, despite a pre-specified global memory allocation.

We generalize the principle of data blocking to automatically partition the data arrays as a function of when (in time) they are consumed. We propose this as a generalization of our previous work [7] to affine dependences.

3 BACKGROUND

We first recap polyhedral analysis and transformation and elements of linear algebra.

3.1 Polyhedral representation, tiling

To be eligible for affine MARS partitioning, a program (or a section thereof) must have a polyhedral representation. It may come either from the analysis of an imperative program (e.g. using PET [13] or Clan [2]) or a domain-specific language. In any case, the following elements are assumed to be available:

- A d-dimensional iteration domain $\mathcal{D} \subset \mathbf{Z}^d$, or a collection of such domains,
- A k-dimensional data domain $\mathcal{A}$,
- A collection $(\varphi_i)_i$ of access functions $\varphi_i : \mathcal{D} \to \mathcal{A}$, defining the reads and writes at each instance,
- A *polyhedral reduced dependence graph* (PRDG), constructed e.g. via array dataflow analysis [6].

The core elements extracted from the polyhedral representation are the dependences, that model which data must be available for a computation (any point in $\mathcal{D}$) to take place. The data flow notably comprises two kinds of dependences we focus about in this paper:

- Flow dependences: correspond to passing of intermediate results within the polyhedral section of the program,
- Input dependences: correspond to input data going into the program.

Definition 3.1. A *dependence function* is any affine function from an iteration domain $\mathcal{D}$ to another domain $\mathcal{D}'$ (iteration or data). In particular, a dependence function is a single-valued relation (each element of $\mathcal{D}$ has a single image).

Each dependence will be noted B, and as an affine function, it is computed as $B(\vec{x}) = A\vec{x} + \vec{b}$ with A a matrix and $\vec{b}$ a vector.

Each *domain* is a subset of a *Euclidean vector space* $E \subset \mathbf{Z}^d$. In particular, every point $x \in \mathcal{D}$ is associated to a vector $\vec{x} \in E$. Section 3.2 gives further elements of linear algebra used throughout this paper.

Definition 3.2. A dependence $B(\vec{x}) = A\vec{x} + \vec{b}$ is said to be *uniform* when A is the square identity matrix. A collection of dependences $B_1, \dots, B_n$ are *uniformly intersecting*, as defined by Agarwal et al. [1]; if they all have the same linear part, i.e., the same A matrix.

To create a partitioning of the data spaces, our work relies on an existing partitioning of iteration space specified as a loop tiling [9, 11, 12, 14] via a set of *tiling hyperplanes*. Each hyperplane is defined by a *normal vector* (of unit norm). Tiles are periodically repeated, with a period s called the *tile size*. We notably use *scaled normal vectors* that translate a point from a tile to the same point in another tile by crossing one tiling hyperplane.

In this work, we assume **tiling hyperplanes are mutually linearly independent**. Each tile has (unique) coordinates that are represented by a t-dimensional vector $\vec{t} = (i_1, \dots, i_t)$ where t is the number of tiling hyperplanes. This tile is the set defined by

$$T(\vec{t}) = \left\{ \vec{x} \in E : \bigwedge_{j \in \{1,\dots,t\}} s_j i_j \leqslant \frac{1}{s_j}(\vec{x} \cdot \vec{n}_j) < s_j(1 + i_j) \right\}$$

The *footprint* of a dependence B and a tile $T(\vec{t})$ is the image of the tile by the dependence:

$$B\left\langle T(\vec{t}) \right\rangle = \left\{ B(\vec{x}) : x \in T(\vec{t}) \right\}$$

3.2 Linear algebra

In this paper, we use several fundamental results from linear algebra. Below are reminders of them for the reader's reference.

3.2.1 Spaces and bases.

Definition 3.3. Let E be an Euclidean vector space of d dimensions with its scalar product noted $\vec{x} \cdot \vec{y}$. Let $\mathcal{B} = (\vec{e}_1, \dots, \vec{e}_d)$ be a basis of

E. $\mathcal{B}$ is called an *orthonormal basis* of E when for all $i \neq j$, $\vec{e}_i \cdot \vec{e}_j = 0$ and for all i, $\vec{e}_i \cdot \vec{e}_i = 1$.

Proposition 3.4. *Any Euclidean space E of d dimensions admits an orthonormal basis.*

The proof of Proposition 3.4 is done by applying the Gram-Schmidt basis orthonormalization to an existing basis.

Definition 3.5. The vector space of all linear combinations of a number of vectors $(\vec{e}_1, \ldots, \vec{e}_n)$ is noted $\text{vect}(\vec{e}_1, \ldots, \vec{e}_n)$. Notably, that space has up to n dimensions, and exactly n dimensions if all the n vectors are linearly independent.

Definition 3.6. Two subspaces S_1 and S_2 of a vector space E are *supplementary* into E when their intersection is the null vector $\vec{0}$, and there exists a decomposition of all $\vec{x} \in E$ as $\vec{x}_1 + \vec{x}_2$ with $\vec{x}_1 \in S_1$ and $\vec{x}_2 \in S_2$. That decomposition is notably unique.

3.2.2 Linear applications.

Definition 3.7. Let $A : E \longrightarrow F$ be a linear application. The subspace K of E such that $\forall \vec{x} \in K, A\vec{x} = \vec{0}$ is called the *null space* of A and is noted $\ker(A)$.

Definition 3.8. Let $A : E \longrightarrow F$ be a linear application. The image of E by A is noted $A\langle E \rangle$. Likewise, the image of a subspace $S \subset E$ by A is noted $A\langle G \rangle$. The preimage of a subspace $T \subset F$ is noted $A^{-1}\langle F \rangle$.

Proposition 3.9. *If $A : E \longrightarrow F$ is a linear application, E has d dimensions, and $\ker(A)$ is its null space, then let $k \leqslant d$ be the dimensionality of $\ker(A)$. There exists a $d-k$-dimensional supplementary I of $\ker(A)$ in E, such that:*

$$\forall \vec{x} \in I, (A\vec{x} = \vec{0} \Rightarrow \vec{x} = \vec{0})$$

4 ITERATION/DATA SPACE PARTITIONING

This section constitutes the core of our work: it proposes a breakup of the iteration and data spaces based on the same properties as the existing uniform breakup, detailed in Section 4.1.

The reasoning leading to the MARS starts from a simple, restrictive case (one single dependence, Section 4.3) and progressively relaxes its hypotheses (multiple *uniformly intersecting* dependences, Section 4.4 and non-*uniformly intersecting* dependences, Section 4.5). The last step of the reasoning in Section 4.6 adds the constraint of partitioning an existing tiled space, which allows to partition intermediate results.

4.1 Case of uniform dependences

Ferry et al. [7] introduced Maximal Atomic irRedundant Sets (MARS) as a partition of the *flow-out* iterations of a tile, such that every element of the partition is the largest set of iterations that verifies:

- Atomicity: consumption of a single element from a MARS implies consumption of the entire MARS.
- Maximality: considering all the consumers of a MARS M_0 (C_0) and all the consumers of another MARS M_1 (C_1), if $C_0 = C_1$, then $M_0 = M_1$.
- Irredundancy: each element of the MARS space belongs to no more than a single MARS.

While [7] uses the flow-in and flow-out information in the sense of [3], input data and output data do not belong to this information, and uses the notion of *footprint* [1]; notably, that the *flow-in iterations* of a tile coincide with the footprint of a tile (of iterations) on another tile of iterations.

The properties of MARS constructed with uniform dependences are the same as those sought in this paper. Merely proposing a partition of the iteration or data spaces satisfies the irredundancy property; the properties to actually check from the partitioning are the atomicity and maximality.

4.2 The problem: uniform versus affine dependences

In the uniform case, MARS can be constructed by enumerating all the *consumer tiles* of a given tile, i.e., those other tiles that need data from it. Uniformity guarantees that there are a finite number of consumer tiles, and that all tiles will exhibit the same MARS regardless of their position in the iteration space (i.e., MARS are invariant by translation of a tile).

With affine dependences the consumer tiles may depend on the size parameters and there may be an unbounded number of them. Also, it becomes necessary to assert the property of **shift invariance**, i.e., that the set is invariant under translation.

In the rest of this section, we will prove, for one and multiple dependences:

- The **existence of a finite set of representatives of all consumer tiles**, suitable to determine the MARS partition,
- The **shift invariance** of the partitioning, and conditions to guarantee it.

4.3 Case of a single affine dependence

The simplest case is when there is a single affine dependence between a tiled iteration space and a data space. We start with an example and then explain the general case.

4.3.1 Example. Consider an program with a single dependence, and non-canonical tiling hyperplanes.

- Domain: $\{(i,j) : 0 \leqslant i < N, 0 \leqslant j < M\}$, basis vectors $\vec{e}_i, \vec{e}_j$
- Dependence : $S_0(i,j) \mapsto \mathcal{A}(i)$, represented as $B(i,j) = (i)$ (i.e., $B(\vec{x}) = A\vec{x} + \vec{b}$ with $A : (i,j) \mapsto (i)$ and $\vec{b} = \vec{0}$).
- Tiling hyperplanes : $H_0 : i + j$, $H_1 : j - i$
- Normal vectors: $\vec{n}_1 = (1,1)$, $\vec{n}_2 = (-1,1)$; scaled normal vectors (w.r.t. tile size): $\vec{\mathbf{n}}_1 = (s/2, s/2)$, $\vec{\mathbf{n}}_2 = (-s/2, s/2)$
- Tile size : $s \in \mathbf{N}^*$

We want to construct the MARS on the $\mathcal{A}$ data space. To do so, we are going to compute the *footprint* [1] of a tile onto the data space along the dependence B; then, by noticing that all footprints are a translation of the same footprint, we will determine parametrically which tiles have intersecting footprints, and compute the MARS using the same method as [7].

We first define a tile of iterations with a parametric set : we call $T(i_0, i_1)$ the set :

$$T(i_0,i_1) = \{(i,j) : si_0 \leqslant i + j < s(1+i_0) \wedge si_1 \leqslant j - i < s(1+i_1)\}$$

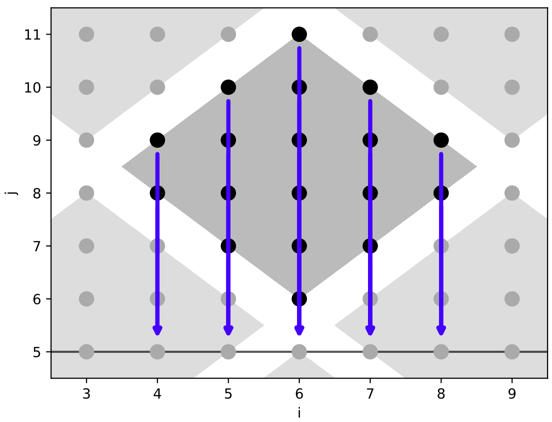

Figure 2: Footprint of one tile with a single affine dependence $B(i,j) = (i)$. The one-dimensional destination space is shown as a continuous line.

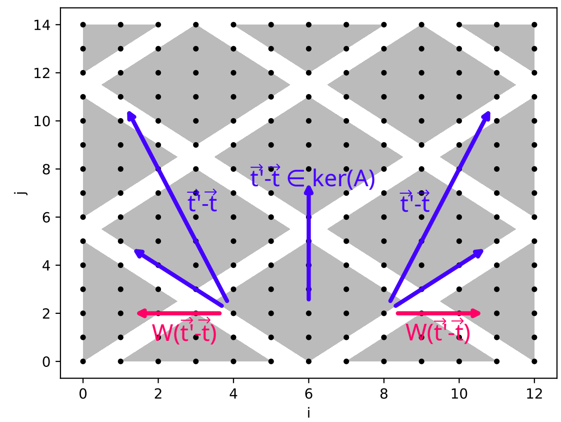

Figure 3: Some consumer tiles of one tile $T(\vec{t})$ with a single dependence $B(i,j) = (i)$, and the projection $W(\vec{t'} - \vec{t})$ on a supplementary of $\ker(A)$. There are a finite number of such projected vectors, and they are constant.

The footprint of $T(i_0, i_1)$ by the dependence B, appearing in Figure 2, is therefore:

$$B\langle T(i_0,i_1)\rangle = \{(i) : \exists j : si_0 \leqslant i+j < s(1+i_0) \wedge si_1 \leqslant j-i < s(1+i_1)\}$$

where the existential quantifier may be removed:

$$B\langle T(i_0,i_1)\rangle = \{(i) : s(i_0-i_1-1) < 2i < s(i_0-i_1+1)\}$$

Given (i_0, i_1), we now seek the other tiles whose footprint's intersection with $B\langle T(i_0,i_1)\rangle$ is not empty: let (i_2, i_3) be another tile.

$$\begin{aligned} &B\langle T(i_0,i_1)\rangle \cap B\langle T(i_2,i_3)\rangle = \\ &\{(i) : s(i_0-i_1-1)+1 \leqslant 2i \leqslant s(i_0-i_1+1)-1 \\ &\wedge s(i_2-i_3-1)+1 \leqslant 2i \leqslant s(i_2-i_3+1)-1\} \end{aligned}$$

The intervals $[\![s(i_0-i_1-1)+1; s(i_0-i_1+1)-1]\!]$ and $[\![s(i_2-i_3-1)+1; s(i_2-i_3+1)-1]\!]$ intersect if $i_0-i_1 = i_2-i_3+1$, $i_0-i_1 = i_2-i_3$ or $i_0-i_1 = i_2-i_3-1$.

The valid (i_2, i_3)s are therefore:

$$\begin{aligned} (i_2,i_3) \in &\{(i_0+p, i_1+p); p \in \mathbf{Z}\} \\ &\cup \{(i_0+p-1, i_1+p); p \in \mathbf{Z}\} \\ &\cup \{(i_0+p+1, i_1+p); p \in \mathbf{Z}\} \end{aligned}$$

as shown in blue in Figure 3.

The space of valid (i_2, i_3) is potentially unbounded: we can visually see this since all tiles along a vertical axis share the same footprint on $\mathcal{A}$. We formalize this intuition by computing the kernel of A as

$$\ker(A) = \mathrm{vect}(\vec{e}_j)$$

and the image of a point on $\mathcal{A}$ is invariant by any vertical translation.

There are however only three *distinct* footprints intersecting with that of $T(i_0, i_1)$; all the others come from tiles which are translations along $\ker(A)$. These footprints come from the tiles to the north-west, north-east and vertically above, plus the vertical translation of all these tiles; these consumer tiles are shown in Figure 3.

We can decompose the space (i, j) using a basis of the kernel and a supplementary, for instance $E = \mathrm{vect}(\vec{e}_i) \oplus \ker(A)$.

In this basis, we can express the coordinates of tile's origins for the case where $(i_2, i_3) = (i_0+p+1, i_1+p)$ with $p \in \mathbf{Z}$ using the scaled normal vectors $\vec{\mathbf{n}}_1, \vec{\mathbf{n}}_2$:

$$\begin{aligned} i_2\vec{\mathbf{n}}_1 + i_3\vec{\mathbf{n}}_2 &= i_2\frac{s}{2}(\vec{e}_i+\vec{e}_j) + i_3\frac{s}{2}(\vec{e}_j-\vec{e}_i) \\ &= \frac{s}{2}(i_0+p+1)(\vec{e}_i+\vec{e}_j) + \frac{s}{2}(i_1+p)(\vec{e}_j-\vec{e}_i) \\ &= \frac{s}{2}(i_0-i_1+1)\vec{e}_i + \frac{s}{2}(i_0+i_1+2p+1)\vec{e}_j \end{aligned}$$

which, when projected onto $\mathrm{vect}(\vec{e}_i)$, gives:

$$P_{\mathrm{vect}(\vec{e}_i)}(i_2\vec{\mathbf{n}}_1 + i_3\vec{\mathbf{n}}_2) = \frac{s}{2}(i_0-i_1+1)\vec{e}_i$$

which is independent of p. This means that all points within tile $T(i_2, i_3)$ have the same image by B. Therefore, **given (i_0, i_1), the entire family of tiles (i_0+p+1, i_1+p) have the same footprint on $\mathcal{A}$.** We can therefore consider a single representative of that family to compute the MARS.

Likewise, if $(i_2, i_3) = (i_0+p-1, i_1+p)$ with $p \in \mathbf{Z}$, then

$$P_{\mathrm{vect}(\vec{e}_i)}(i_2\vec{\mathbf{n}}_1 + i_3\vec{\mathbf{n}}_2) = \frac{s}{2}(i_0-i_1-1)\vec{e}_i$$

which is also independent of p; the same conclusion holds for $(i_2, i_3) = (i_0+p, i_1+p)$ and $P_{\mathrm{vect}(\vec{e}_i)}(i_2\vec{\mathbf{n}}_1 + i_3\vec{\mathbf{n}}_2) = \vec{0}$. Figure 3 shows in pink the projection of the *translation vectors* from the tile $T(\vec{t})$ to its consumers (there are only two non-null projections, so only two such vectors appear).

There are infinitely many tiles whose footprint intersects with that of a given tile; however, to compute the MARS, we have demonstrated that it is sufficient to take three representative tiles. The same procedure as in [7] can be applied once these three *consumer tiles* have been determined.

Per [7], we compute the respective intersections with $B\langle T(i_0,i_1)\rangle$ of all other consumer tiles: for $(i_2, i_3) = (i_0+p+1, i_1+p)$ with

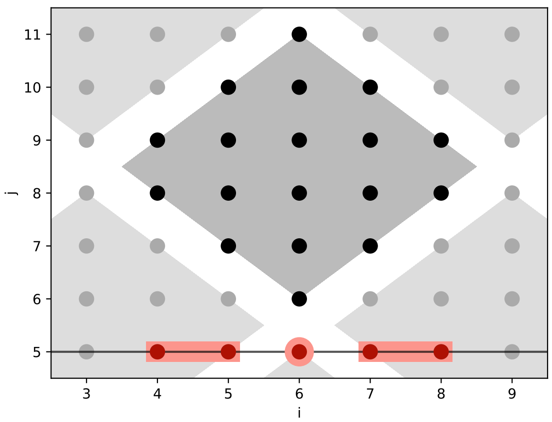

Figure 4: MARS obtained with a single affine dependence $B(i, j) = (i)$.

$p \in \mathbf{Z}$,

$$\begin{aligned} B\langle T(i_0, i_1)\rangle \cap B\langle T(i_0+p+1, i_1+p)\rangle &= \\ \{(i) : s(i_0 - i_1 - 1) < 2i < s(i_0 - i_1 + 1) \\ \wedge s(i_0 - i_1) < 2i < s(i_0 - i_1 + 2)\} \\ &= \{(i) : s(i_0 - i_1) < 2i < s(i_0 - i_1 + 1)\} \end{aligned}$$

When $(i_2, i_3) = (i_0 + p - 1, i_1 + p)$ with $p \in \mathbf{Z}$,

$$\begin{aligned} B\langle T(i_0, i_1)\rangle \cap B\langle T(i_0+p-1, i_1+p)\rangle &= \\ &= \{(i) : s(i_0 - i_1 - 1) < 2i < s(i_0 - i_1)\} \end{aligned}$$

Finally, when $(i_2, i_3) = (i_0 + p, i_1 + p)$,

$$\begin{aligned} B\langle T(i_0, i_1)\rangle \cap B\langle T(i_0+p, i_1+p)\rangle &= \\ &= \{(i) : s(i_0 - i_1) = 2i\} \end{aligned}$$

Also, $B\langle T(i_0+p+1, i_1+p)\rangle \cap B\langle T(i_0+p-1, i_1+p)\rangle = \emptyset$, so we have all the MARS.

The MARS on symbol $\mathcal{A}$ for this program seen from a tile $T(i_0, i_1)$ are therefore the three sets $\{(i) : s(i_0 - i_1) < 2i < s(i_0 - i_1 + 1)\}$, $\{(i) : s(i_0 - i_1 - 1) < 2i < s(i_0 - i_1)\}$ and $\{(i) : s(i_0 - i_1) = 2i\}$. These MARS are shown in Figure 4.

4.3.2 General case. In the general case, computing the MARS for a single dependence leading to a non-tiled space can be done as follows. Let $\mathcal{D}$ be the N-dimensional iteration space of the consumer points of the dependence, and E be the vector space such that $\mathcal{D} \subset E$.; let $\mathcal{A}$ be the destination space. Let B be the dependence with $B(\vec{x}) = A\vec{x} + \vec{b}$. Let $(H_1, \ldots, H_t)$ be the t tiling hyperplanes, $(\vec{n}_1, \ldots, \vec{n}_t)$ normal vectors to the tiling hyperplanes, $(s_1, \ldots, s_t)$ be the tile sizes.

For a tile coordinate be $\vec{t} = (i_1, \ldots, i_t)$, the tile is defined as

$$T(\vec{t}) = \left\{\vec{x} = (x_1, \ldots, x_N) : \forall j \in \{1, \ldots, t\} : s_j i_j \leqslant \vec{x} \cdot \vec{n}_j < s_j(1 + i_j)\right\}$$

We can compute, with $\vec{t}$ and another tile $\vec{t'}$ as a parameter, when the intersection of $B\left\langle T(\vec{t})\right\rangle$ and $B\left\langle T(\vec{t'})\right\rangle$ is non-empty using affine operations. Let $V(\vec{t})$ be:

$$V(\vec{t}) = \left\{\vec{t'} : B\left\langle T(\vec{t})\right\rangle \cap B\left\langle T(\vec{t'})\right\rangle \neq \emptyset\right\}$$

which is obtainable by taking the parameter space of $I(\vec{t}, \vec{t'})$. Here, $V(\vec{t})$ represents the tile coordinates of all tiles whose footprint on $\mathcal{A}$ intersects that of $T(\vec{t})$.

In [7], $V(\vec{t})$ is determined by browsing through neighboring tiles. The main difficulty here is that $V(\vec{t})$ **is potentially unbounded.** We will demonstrate that there are only a finite number of *distinct footprints* overlapping with $B\left\langle T(\vec{t'})\right\rangle$. To determine them, we suggest decomposing E into $\ker(A)$ and a supplementary I of $\ker(A)$, i.e.,

$$E = I \oplus \ker(A)$$

Proposition 3.9 gives us that such a decomposition always exists, and per Proposition 3.4, there is an orthonormal basis of the resulting space.

If $r = \operatorname{rank}(A)$, let $(\vec{e}_1, \ldots, \vec{e}_r)$ be a basis of I and $(\vec{e}_{r+1}, \ldots, \vec{e}_d)$ be a basis of $\ker(A)$ such that $(\vec{e}_1, \ldots, \vec{e}_d)$ is an orthonormal basis of E. Let $(\vec{n}_1^p, \ldots, \vec{n}_t^p)$ be the orthogonal projections of the $\vec{n}_i$s onto $(\vec{e}_1, \ldots, \vec{e}_r)$; in particular, these have zero $r+1$-th through d-th coordinates.

For any $\vec{t'} \in V(\vec{t})$, if $\vec{t'} - \vec{t} = (\delta_1, \ldots, \delta_t)$, we can compute

$$W(\vec{t'} - \vec{t}) = \sum_{i=1}^{t} \delta_i \vec{n}_i^p$$

which represents the part of the translation between tiles that results in translating the images.

The most important result needed to construct the MARS is the ability to enumerate *all* the footprints. The following proposition formalizes this.

Proposition 4.1. *The set $P(\vec{t}) = \left\{W(\vec{t'} - \vec{t}) : \vec{t'} \in V(\vec{t})\right\}$ is finite, and for each consumer tile $\vec{t'} \in V(\vec{t})$, there exists a unique $\vec{p} \in P(\vec{t})$ such that*

$$B\left\langle T(\vec{t'})\right\rangle = \left\{\vec{y} + A\vec{p} : \vec{y} \in B\left\langle T(\vec{t})\right\rangle\right\}$$

and that $\vec{p}$ is a constant vector, independent of $\vec{t}$ (i.e., the consumer tiles are invariant by translation).

Proof. **Completeness of footprints:** Let $\vec{t'} \in V(\vec{t})$, i.e., a tile whose footprint intersects that of tile $\vec{t}$. We know that $W(\vec{t'} - \vec{t}) = \sum_{i=1}^{t}(t'_i - t_i)\vec{n}_i^p = \vec{p} \in P(\vec{t})$. Then:

$$\begin{aligned} B\left\langle T(\vec{t'})\right\rangle &= \left\{A\vec{x} + \vec{b} : \forall i : s_i t'_i \leqslant \vec{x} \cdot \vec{n}_i \leqslant (1 + t'_i)s_i\right\} \\ &= \left\{A\vec{x} + \vec{b} : \forall i : s_i(t_i + (t'_i - t_i)) \leqslant \vec{x} \cdot \vec{n}_i \leqslant (1 + t_i + (t'_i - t_i))s_i\right\} \\ &= \left\{A\left(\vec{x} + \textstyle\sum_{i=1}^{t}(t'_i - t_i)\vec{n}_i\right) + \vec{b} : \forall i : s_i t_i \leqslant \vec{x} \cdot \vec{n}_i \leqslant (1 + t_i)s_i\right\} \\ &= \left\{A\left(\vec{x} + \textstyle\sum_{i=1}^{t}(t'_i - t_i)\vec{n}_i^p\right) + \vec{b} : \vec{x} \in T(\vec{t})\right\} \\ &= \left\{\vec{y} + A\vec{p} : \vec{y} \in B\left\langle T(\vec{t})\right\rangle\right\} \end{aligned}$$

using the fact that $A(\vec{n}_i) = A(\vec{n}_i^p)$.

Uniqueness of $\vec{p}$: A is bijective between I (supplementary of $\ker(A)$ in E) and $\mathrm{Im}(A)$. Therefore, because $\vec{p} \in P(\vec{t})$ is in I, it is the unique element of I which $A\vec{p}$ is the image. Therefore, $\vec{p}$ is unique in the sense of this proposition.

Finiteness of $P(\vec{t})$: For all $\vec{t'} \in V(\vec{t})$, $B\left\langle T(\vec{t'})\right\rangle$ is a translation of $B\left\langle T(\vec{t})\right\rangle$ by $A\vec{p}$ with some $\vec{p} \in P(\vec{t})$. The coordinates of $\vec{t}$ are integers, therefore $A\vec{p}$ is an integer linear combination of the $A\vec{n}_i^p$s for $i \in \{1,\ldots,t\}$. $T(\vec{t})$ being bounded, the footprint $B\left\langle T(\vec{t})\right\rangle$ is bounded, therefore only a finite number of translations of itself by $A\vec{p}$s intersect with it.

Constantness of $\vec{p}$: Let $\vec{t}_0, \vec{t}_1 \in \mathbf{Z}^t$ and $\vec{t'}_0 \in V(\vec{t}_0)$, and $\vec{p} = W(\vec{t'} - \vec{t})$. Let $\vec{t'}_1 = \vec{t}_1 + (\vec{t'}_0 - \vec{t}_0)$. Then:

$$\begin{aligned}
B\left\langle T(\vec{t'}_1)\right\rangle &= \left\{A\left(\vec{x} + \textstyle\sum_{i=1}^{t}(t'_{1i} - t_{1i})\vec{n}_i\right) + \vec{b} : \forall i : s_i t_{1i} \leqslant \vec{x}\cdot\vec{n}_i \leqslant (1+t_{1i})s_i\right\} \\
&= \left\{A\left(\vec{x} + \textstyle\sum_{i=1}^{t}(t_{1i} - t_{1i} + (t'_{0i} - t_{0i}))\vec{n}_i\right) + \vec{b} : \vec{x} \in T(\vec{t}_1)\right\} \\
&= \left\{A\vec{x} + \textstyle\sum_{i=1}^{t}(t'_{0i} - t_{0i})A\vec{n}_i + \vec{b} : \vec{x} \in T(\vec{t}_1)\right\} \\
&= \left\{B(\vec{x}) + A\vec{p} : \vec{x} \in T(\vec{t}_1)\right\}
\end{aligned}$$

which means that the translation between the images of $T(\vec{t}_1)$ and $T(\vec{t'}_1)$ is the same as that of $T(\vec{t}_0)$ and $T(\vec{t'}_0)$.

□

We can therefore enumerate $P(\vec{t})$, knowing that for each $\vec{w} \in P(\vec{t})$, $P^{-1}(\vec{w})$ represents *consumer tiles* that all have the same footprint by B. That footprint is computed as follows:

$$\Phi(\vec{w}) = B\left\langle T(\vec{t}) + \vec{w}\right\rangle \text{ where } T(\vec{t}) + \vec{w} = \left\{\vec{x} + \vec{w} : \vec{x} \in T(\vec{t})\right\}$$

We can then compute the MARS. For all the combinations of $\vec{w}$s, i.e., for all $C \in \mathcal{P}\left(P(\vec{t})\right)$, we determine the MARS associated with that combination of consumer tiles:

$$\mathcal{M}_C = \bigcap_{\vec{w}\in C}\left(\Phi(\vec{w}) \cap B\left\langle T(\vec{t})\right\rangle\right) \setminus \bigcup_{\vec{w}\notin C}\left(\Phi(\vec{w}) \cap B\left\langle T(\vec{t})\right\rangle\right)$$

4.4 Case of multiple, uniformly intersecting dependences

4.4.1 General case. If the dependences are uniformly intersecting, they all have the same linear part. This means that they all have the same null space, and therefore the space decomposition into $E = I \oplus \ker(A)$ still applies.

Let there be Q dependences $B_1, \ldots, B_Q$ that are uniformly intersecting. This means that there exists an unique matrix A such that:

$$\forall i : B_i(\vec{x}) = A\vec{x} + \vec{b}_i$$

Because all tiles share the same linear part, the space of consumer tiles for each dependence will be the same *up to a translation.* Their linear part will notably be the same, and the same argument as in the case above holds to guarantee that $P(\vec{t}) = \left\{W(\vec{t'} - \vec{t}) : \vec{t'} \in V(\vec{t})\right\}$ is finite.

Let, by abuse of the notation, $B\left\langle T(\vec{t})\right\rangle$ be the combined footprint of all dependences:

$$B\left\langle T(\vec{t})\right\rangle = \bigcup_{i=1}^{Q} B_i\left\langle T(\vec{t})\right\rangle$$

For each dependence B_i with $i \in \{1,\ldots,Q\}$, we therefore compute $V_i(\vec{t})$ by intersecting $B_i\left\langle T(\vec{t'})\right\rangle$ and $B\left\langle T(\vec{t})\right\rangle$ (i.e., we want the intersection of the footprint of *one dependence* and the footprint of *all other dependences*); let

$$V(\vec{t}) = \bigcup_{i=1}^{Q} V_i(\vec{t})$$

The same decomposition $E = \mathrm{vect}(\vec{e}_1, \ldots, \vec{e}_r) \oplus \mathrm{vect}(\vec{e}_r + 1, \ldots, \vec{e}_d)$ is applicable due to all B_is sharing the same linear part A.

We can give a more meaningful expression for $P(\vec{t})$:

$$P(\vec{t}) = \left\{W(\vec{t'} - \vec{t}) : \vec{t'} \in \bigcup_{i=1}^{Q} V_i(\vec{t})\right\}$$

which means that $P(\vec{t})$ is composed of the projections of the vectors leading to any consumer tile of *any dependence* (and therefore takes into account the uniform translations between dependences).

The MARS can be computed by using $P(\vec{t})$. There are two differences with the case when there is only a single dependence:

- The footprints of the consumer tiles $\Phi(\vec{w})$ are specific to each dependence,
- The footprint of the tile $\vec{t}$ is the *union* of the footprint of all dependences.

For $i \in \{1,\ldots,Q\}$, let $\Phi_i(\vec{w})$ be:

$$\Phi_i(\vec{w}) = B_i\left\langle T(\vec{t}) + \vec{w}\right\rangle \text{ where } T(\vec{t}) + \vec{w} = \left\{\vec{x} + \vec{w} : \vec{x} \in T(\vec{t})\right\}$$

The MARS are constructed by taking all subsets of consumer tiles from $P(\vec{t})$, and looking at the points consumed *only* by these tiles.

Formally, let the cardinality of $P(\vec{t})$ be $\#C$. For all $K : 1 \leqslant K \leqslant \#C$ and all permutations σ of $\{1,\ldots,\#C\}$, let

$$C = \left\{\vec{t}_{\sigma(1)}, \ldots, \vec{t}_{\sigma(K)}\right\} \text{ and } \overline{C} = \{\vec{t}_{\sigma(K+1)}, \ldots, \vec{t}_{\sigma(\#C)}\}$$

Then, a MARS is constructed according to the following rules:

- For each consumer tile coordinates $\vec{t'} \in C$, there exists a dependence leading to $T(\vec{t'})$,
- No dependence leads to a consumer tile $\vec{t'} \in \overline{C}$

These two conditions to form a MARS can be written as:

$$\mathcal{M}_C = \bigcap_{\vec{w}\in C}\left(\bigcup_{i=1}^{Q}\Phi_i(\vec{w}) \cap B\left\langle T(\vec{t})\right\rangle\right) \setminus \bigcup_{\vec{w}\in\overline{C}}\left(\bigcup_{i=1}^{Q}\Phi_i(\vec{w}) \cap B\left\langle T(\vec{t})\right\rangle\right)$$

and there are at most $\mathrm{card}(\mathcal{P}(P(\vec{t}))) = 2^{\mathrm{card}(P(\vec{t}))}$ Cs and therefore as many MARS.

4.4.2 Example: uniform dependences. In this paragraph, we show that the computation of MARS using [7] coincides with that proposed in this paper when the dependences are uniform. Such dependences are a special case of uniformly intersecting dependences, with a linear part being identity. Note that the destination space is considered to be a data space, and therefore dependences within a tile are counted in the footprint (self-consumption of data produced by a tile is dealt with in the next section).

Consider the Jacobi 1D example:

- Domain: $\{(i,j) : 0 \leqslant i < N, 0 \leqslant j < M\}$, basis vectors $\vec{e}_i, \vec{e}_j$
- Dependences : $B_1(i,j) = (i-1, j-1)$, $B_2(i,j) = (i, j-1)$, $B_3(i,j) = (i+1, j-1)$

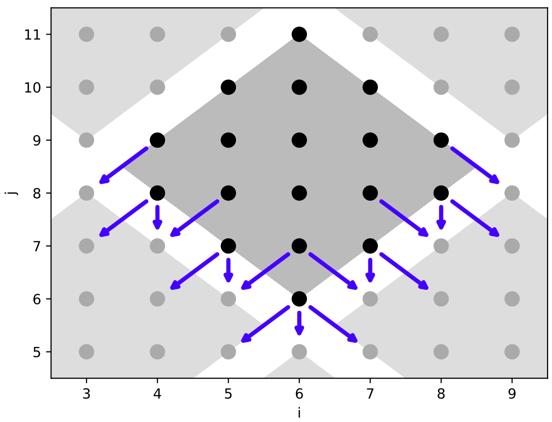

Figure 5: Flow-in dependences of tile $T(\vec{t})$ with uniformly intersecting dependences (Jacobi 1D).

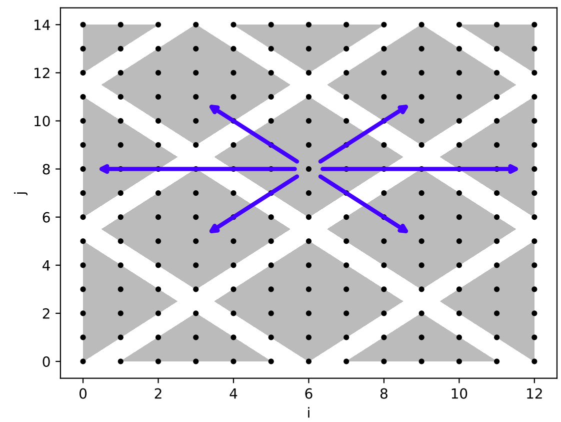

Figure 6: Consumer tiles of array $\mathcal{A}$ with Jacobi 1D dependences, sharing their footprint with tile $T(\vec{t})$.

- Tiling hyperplanes : $H_1 : i + j$ ($\vec{n}_2 = (1, 1)$), $H_2 : j - i$ ($\vec{n}_2 = (-1, 1)$)
- Tile size : $s \in \mathbb{N}^*$

We compute the unified footprint $B\left\langle T(\vec{t})\right\rangle$:

$$B\left\langle T(\vec{t})\right\rangle = \{(i, j) : si_1 \leqslant i + j + (2 - p) < s(1 + i_1) \\ \wedge si_2 \leqslant j - i + p < s(1 + i_2) : p \in \{0, 1, 2\}\}$$

Notably, if we confuse the data space $\mathcal{A}(i, j)$ and the iteration space (i, j) (that is, each cell of $\mathcal{A}$ contains the result of one iteration), and we restrict the footprint to those points outside tile $T(\vec{t})$, we obtain the *flow-in* of that tile as in Figure 5, corresponding to the same definition as in [7].

We determine the individual $V_i(\vec{t})$s:

$$V_1(\vec{t}) = \{(i_1, i_2 - 1), (i_1, i_2), (i_1 + 1, i_2 - 1), (i_1 + 1, i_2)\}$$

$$V_2(\vec{t}) = \{(i_1, i_2), (i_1 + 1, i_2), (i_1, i_2 - 1), (i_1 - 1, i_2), (i_1, i_2 + 1)\}$$

$$V_3(\vec{t}) = \{(i_1, i_2), (i_1 - 1, i_2), (i_1, i_2 + 1), (i_1 - 1, i_2 + 1)\}$$

which gives

$$V(\vec{t}) = \{(i_1, i_2), (i_1 - 1, i_2), (i_1, i_2 + 1), (i_1 - 1, i_2 + 1), (i_1 + 1, i_2), \\ (i_1, i_2 - 1), (i_1 + 1, i_2 - 1)\}$$

As $\ker(A) = \{0\}$, we easily get that $E = \text{vect}(\vec{e}_i, \vec{e}_j)$ and therefore constructing the $W(\vec{t'} - \vec{t})$ is straightforward, yielding the following $P(\vec{t})$:

$$P(\vec{t}) = \{(0, 0), (-1, 0), (0, 1), (-1, 1), (1, 0), (0, -1), (1, -1)\}$$

This $P(\vec{t})$ means there are seven tiles (including $\vec{t}$ itself) whose footprint (i.e., *any dependence*) intersects with $B\left\langle T(\vec{t})\right\rangle$. These consumer tiles are shown in Figure 6.

For the sake of shortness, we will not enumerate all combinations of consumer tiles. The MARS that appear after partitioning the footprints stemming from all consumer tiles are shown in Fig.7.

Again, if we confuse the iteration and data spaces, we can obtain the same MARS as computed in [7] by removing those MARS that are contained within $T(\vec{t})$; the result of this operation, shown in

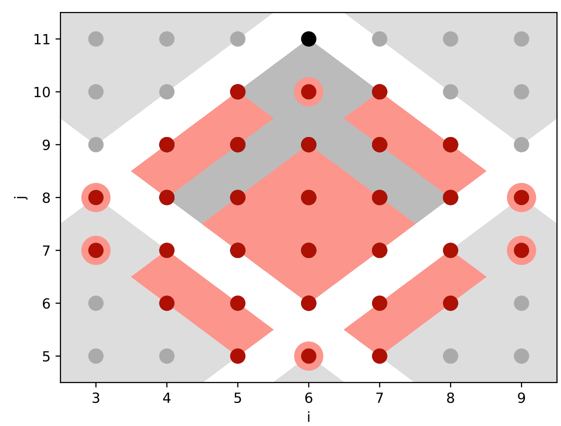

Figure 7: MARS for uniformly intersecting dependences (Jacobi 1D)

Figure 8, illustrates the coincidence between the MARS computed using uniform and affine dependences.

4.5 Case of multiple, non-uniformly intersecting dependences

We now consider the case where the dependences are not uniformly intersecting. In this case, the main difference is that dependences no longer share the same linear part. Therefore, we need to write every dependence separately:

$$\forall i \in \{1, \ldots, Q\} : B_i(\vec{x}) = A_i\vec{x} + \vec{b}_i$$

and each dependence having its own null space, there is one orthonormal basis of the null space and a supplementary per dependence, and therefore one projection $W_i(\vec{t'} - \vec{t})$ per dependence.

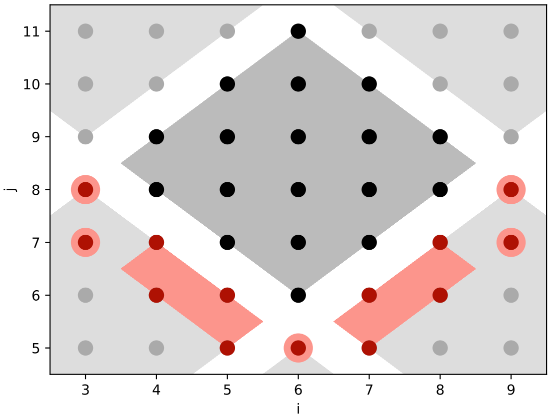

Figure 8: Coincidence between MARS computed with the uniform dependence method, and those computed with the affine method.

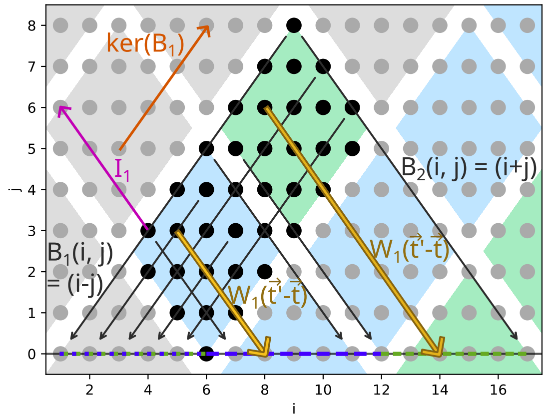

Figure 9: Non-uniformly intersecting dependences do not guarantee that the vectors $W_i(\vec{t'} - \vec{t})$ do not depend on $\vec{t}$, i.e., each tile's footprint is not necessarily a translation of another tile's footprint.

4.5.1 *Single null space requirement.* Because the dependences may no longer have the same linear part, each linear part may have a different null space. When considering any consumer tile $\vec{t'} \in V(\vec{t})$, it is no longer true that the projection of $\vec{t'} - \vec{t}$ onto each null space is independent of the tile coordinates. The shift invariance of a tile from Proposition 4.1 therefore no longer holds.

Figure 9 gives an example of this case with two dependences: $B_1(i, j) = (i - j)$ and $B_2(i, j) = (i + j)$. Here, the $\vec{p}$s depend on $\vec{t}$. Due to the dependence B_1, all the tiles northwest of each tile will intersect with its footprint; but the dependence B_2 generates a footprint southwest, also consumed (because of B_1) by all tiles to its northwest.

In Figure 9, we show the $W_1(\vec{t'} - \vec{t}) : \ker(A_1)$ points to the northeast, and I_1 (supplementary of $\ker(A_1)$) points to the northwest, parallel to the dependence B_2.

A sufficient condition for a position-independent footprint to exist is that all dependences have the same null space:

Proposition 4.2. *If all dependences have the same null space, then all tiles have the same footprint up to a translation. Otherwise said, for any $\vec{\delta} \in \mathbf{Z}^t$, there exists $\vec{u} \in \mathrm{Im}(B)$ such that:*
For each $\vec{t} \in \mathbf{Z}^t$, if $\vec{t'} = \vec{t} + \vec{\delta}$, then

$$\bigcup_{i=1}^{Q} B_i \left\langle T(\vec{t'}) \right\rangle = \left\{ \vec{y} + \vec{u} : \vec{y} \in \bigcup_{i=1}^{Q} B_i \left\langle T(\vec{t}) \right\rangle \right\}$$

Proof. We know that the sought $\vec{u}$ exists for each dependence per Proposition 4.1: for each $i \in \{1, \dots, Q\}$, there is a $\vec{u}_i$ such that

$$B_i \left\langle T(\vec{t'}) \right\rangle = \left\{ \vec{y} + \vec{u}_i : \vec{y} \in B_i \left\langle T(\vec{t}) \right\rangle \right\}$$

This $\vec{u}_i$ is constructed as:

$$\vec{u}_i = \sum_{i=1}^{t} (t'_i - t_i) \vec{n}_i^p$$

where the $\vec{n}_i^p$s are the projections of the normal vectors onto a supplementary of the null space of each dependence. Because all of the dependences have the same null space, it comes that all of the $\vec{u}_i$ have the *same* projection onto the *same* supplementary of that null space. Therefore, they are all equal. ◻

4.5.2 *Constructon of MARS with a single null space.* We must prove the requirements stated in Section 4.2, proved in the previous two cases, still hold to compute the MARS.

The uniqueness of $\vec{p}$ (invariance by translation of a tile) has become a hypothesis, and the dependences must satisfy this requirement to compute MARS. The previous paragraph only gave a sufficient condition for it to be satisfied.

The finiteness (and enumerability) of the set representatives of consumer tiles still holds if the dependences all have the same null space.

We can construct the MARS using the same procedure as in 4.4: the footprints of all dependences are distinct, but the null space is the same, therefore the same definition for $P(\vec{t}) = \left\{ W(\vec{t'} - \vec{t}) \right\}$ as in 4.4 holds.

4.6 Case of dependences between tiled spaces

Dependences that lead to tiled spaces correspond to the passing of intermediate results between tiles. These dependences were supported in [7], and transmission of intermediate results was done through MARS transiting in the main memory. This produced a partitioning of the *flow-out set* and *flow-in set* of each tile. In this section, we extend this principle to affine dependences.

Uniform dependences used in [7] guaranteed that the producer and consumer were in the same space (which is not the case with affine dependences), and the identity linear part of the dependences gave that the image of a tile by a dependence was a translation of the tile itself.

The main problem with having different consumer and producer spaces is the relation between the consumer tiles' "footprint" in the producer tiles' space, and the producer space tiling itself: the footprints of the consumer tiles by the dependences produce a tiling that may not match with the existing tiling of the producer space.

In the previous sections (4.3, 4.4, 4.5), the existence of MARS relied on the footprints of the consumer tiles (in a tiled iteration space) in the data space (hereafter *destination space*) being **independent of the consumer tile** (i.e., the origin of the dependence). In this section, the destination space is a tiled iteration space, and we want the tiling induced by the dependence to "match" the existing tiling or be finer than it. To this aim, we add the requirement is that the same footprints are **independent of the producer tile** (i.e., the destination of the dependence).

Assuming there are t tiling hyperplanes in the source space, and q tiling hyperplanes in the destination space, let their (unit) normal vectors be respectively $\vec{n}_1, \ldots, \vec{n}_t$ and $\vec{d}_1, \ldots, \vec{d}_q$ and their tile sizes $s_1, \ldots, s_t$ and $z_1, \ldots, z_q$. Let the scaled normal vectors (translation of one tile along each hyperplace) be $\vec{\mathbf{n}}_1, \ldots, \vec{\mathbf{n}}_t$ and $\vec{\mathbf{d}}_1, \ldots, \vec{\mathbf{d}}_q$.

Let there be Q dependences $B_1, \ldots, B_Q$. Let $\vec{t} = (t_1, \ldots, t_t) \in \mathbf{CT}$ be a *consumer tile* vector coordinate, and let $\vec{u} = (u_1, \ldots, u_q) \in \mathbf{PT}$ be a *producer tile* vector coordinate (in the destination space of the dependences). Let a tile in the destination space be designated as $U(\vec{t})$ using a definition analogous to that of the source space (see 3). Let the consumer tiles of a destination tile $\vec{u} \in \mathbf{DT}$ be

$$X(\vec{u}) = \left\{\vec{t} \in \mathbf{CT} : \exists j \in \{1, \ldots, Q\} : B_j \left\langle T(\vec{t})\right\rangle\right\}$$

and a tile translation vector in the destination space be expressed as:

$$N(\vec{u}) = \sum_{k=1}^{q} u_k \vec{\mathbf{d}}_k$$

The following proposition is a conjecture. It establishes the equivalence between a translation of a tile in the producer space and the translation of multiple tiles in the consumer space.

Proposition 4.3. *We have the following equivalence:*

$$\forall \vec{u}, \vec{u'} \in \mathbf{DT}, \forall \vec{t} \in X(\vec{u}), \forall i \in \{1, \ldots, Q\} :$$
$$\exists \vec{t'} \in X(\vec{u'}) : B_i \left\langle T(\vec{t'})\right\rangle = \left\{\vec{y} + N(\vec{u'} - \vec{u}) : \vec{y} \in B_i \left\langle T(\vec{t})\right\rangle\right\}$$
$$\Updownarrow$$
$$\forall i \in \{1, \ldots, Q\}, \forall j \in \{1, \ldots, t\}, \forall k \in \{1, \ldots, q\} :$$
$$\exists m \in \mathbf{Z} : m((A_i \vec{\mathbf{n}}_j) \cdot \vec{d}_k)\vec{d}_k = \vec{\mathbf{d}}_k$$

If proved, this proposition then establishes a condition on the dependences for there to be a unique control flow, independent of the tile coordinates, for both the MARS to produce (by each producer tile) and the MARS to retrieve (by each consumer tile).

5 RELATED WORK

This work introduces a partitioning of data arrays and iteration spaces based on the consumption pattern of each data. Existing work on partitioning aims first and foremost at locality, before memory access optimization. Our work relies on a locality optimization (tiling) and seeks to further improve memory accesses. Specifically, we combine objectives (partitioning data for spatial locality) and methods (fine-grain partitioning where iteration spaces are already partitioned).

5.1 Goal of partitioning

Existing work on partitioning mainly targets locality, such as Agarwal et al. [1]. Our work uses the same definitions and follows the same reasoning, but with a different objective: while [1] seeks to adjust the tile size and shapes for locality (i.e., the footprint size of each tile), we seek to exhibit spatial locality (data contiguity) opportunities. In that sense, our work is not the first to propose a partitioning of iteration and data spaces using affine dependences; however, the desired result (with a spatial locality objective) differs, and hence the construction procedure and hypotheses.

Parallelism is also an objective: [16] perform iteration and data space partitioning, then fuse partitions to maximize computation parallelism while preserving locality. The resulting code is suitable for CPU and GPU implementation with a cache hierarchy; our partitioning scheme does not follow the temporal utilization of the retrieved data within a tile. It therefore is more adapted to scratchpad memories, and scenarios where memory accesses can be decoupled from computations, because grouping data for spatial locality requires significant on-chip data movement. This makes our approach more suitable for task-level pipelined (*read, execute, write*) FPGA or ASIC accelerators, or for small CPU tiles (register tiling) where the register space can be viewed as a scratchpad.

5.2 Partitioning methods

Instead of partitioning the inter-tile communicated data with a tiling already known, one can consider partitioning the inputs and outputs, and deriving tiled iteration space tiling from the inputs or outputs themselves. This approach is taken in [16] where the tile shapes are iteratively constructed from the (tiled) consumers of the iterations or data.

Monoparametric tiling [8] is performed using an inverse approach as ours: the data spaces (variables) are partitioned into tiles, and then the iteration space is partitioned. It requires the program to be represented as a *system of affine recurrence equations*, where loops do not exist; instead, iteration spaces start to exist at code generation time, when a variable needs to be computed. The main difference is that our partitioning scheme must be applied after loop tiling, and therefore after most locality optimizations.

Dathathri et al. [3, 5] partitions the iteration spaces for inter-node communications in distributed systems, in a manner similar to MARS: the *flow-out* iterations of each tile are partitioned by dependences (*dependence polyhedra*) and consumer tiles (*receiving tiles*). While both approaches are similar with respect to how data is grouped and transmitted, ours is extended to data space partitioning. Our approach however adds a restriction on the dependences: we require that the flow-out partitions are invariant across all tiles, so that a simple, unique control flow can be derived. Our approach can then be used to create *position-independent* accelerators that can process any tile in the iteration space.

It is noteworthy that both our approach and [5], along with other domain-specific inter-node data partitioning schemes (e.g., [17]) acknowledge that, to achieve a high bandwidth utilization of the

RAM or network, inter-tile (inter-node) communications need a specific data layout inferred using static analysis.

6 CONCLUSION

Optimizing programs with respect to memory accesses is a key to improving their performance. This paper proposes an analysis method to automatically partition data and iteration spaces from the polyhedral representation of a program when loop tiling is applied.

Partitioning data arrays is already known to improve spatial locality and, in turn, access performance. In this paper, we propose a fine-grain partitioning scheme that can be used to optimize spatial locality.

REFERENCES

[1] A. Agarwal, D.A. Kranz, and V. Natarajan. 1995. Automatic Partitioning of Parallel Loops and Data Arrays for Distributed Shared-Memory Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 6, 9 (1995), 943–962. https://doi.org/10.1109/71.466632

[2] Cédric Bastoul, Albert Cohen, Sylvain Girbal, Saurabh Sharma, and Olivier Temam. 2003. Putting Polyhedral Loop Transformations to Work. In *LCPC'16 International Workshop on Languages and Compilers for Parallel Computers, LNCS 2958*. College Station, Texas, 209–225.

[3] Uday Bondhugula. 2013. Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM. https://doi.org/10.1145/2503210.2503289

[4] Jason Cong, Wei Jiang, Bin Liu, and Yi Zou. 2011. Automatic memory partitioning and scheduling for throughput and power optimization. *ACM Transactions on Design Automation of Electronic Systems, Vol. 16, No. 2, Article 1* 16 (2011), 1–25. https://doi.org/10.1145/1929943.1929947

[5] Roshan Dathathri, Chandan Reddy, Thejas Ramashekar, and Uday Bondhugula. 2013. Generating Efficient Data Movement Code for Heterogeneous Architectures with Distributed-Memory. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*. IEEE. https://doi.org/10.1109/PACT.2013.6618833

[6] Paul Feautrier. 1991. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming* 20, 1 (02 1991), 23–53. https://doi.org/10.1007/BF01407931

[7] Corentin Ferry, Steven Derrien, and Sanjay Rajopadhye. 2023. Maximal Atomic irRedundant Sets: a Usage-based Dataflow Partitioning Algorithm. In *13th International Workshop on Polyhedral Compilation Techniques (IMPACT'23)*. https://impact-workshop.org/impact2023/papers/paper1.pdf

[8] Guillaume Iooss, Christophe Alias, and Sanjay Rajopadhye. 2021. Monoparametric Tiling of Polyhedral Programs. *International Journal of Parallel Programming* 49, 3 (mar 2021), 376–409. https://doi.org/10.1007/s10766-021-00694-2

[9] F. Irigoin and R. Triolet. 1988. Supernode partitioning. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '88*. ACM, ACM Press, 319–328. https://doi.org/10.1145/73560.73588

[10] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation - PLDI '13*. ACM Press. https://doi.org/10.1145/2491956.2462176

[11] J. Ramanujam and P. Sadayappan. 1992. Tiling multidimensional iteration spaces for multicomputers. *J. Parallel and Distrib. Comput.* 16, 2 (oct 1992), 108–120. https://doi.org/10.1016/0743-7315(92)90027-k

[12] Robert Schreiber and Jack J. Dongarra. 1990. *Automatic blocking of nested loops*. Technical Report.

[13] Sven Verdoolaege and Tobias Grosser. 2012. Polyhedral Extraction Tool. In *Second Int. Workshop on Polyhedral Compilation Techniques (IMPACT'12)*. Paris, France.

[14] Michael E. Wolf and Monica S. Lam. 1991. A data locality optimizing algorithm. *ACM SIGPLAN Notices* 26, 6 (06 1991), 30–44. https://doi.org/10.1145/113446.113449

[15] Tomofumi Yuki, Gautam Gupta, DaeGon Kim, Tanveer Pathan, and Sanjay Rajopadhye. 2012. Alphaz: A system for design space exploration in the polyhedral model. In *International Workshop on Languages and Compilers for Parallel Computing*. Springer, 17–31.

[16] Jie Zhao and Peng Di. 2020. Optimizing the Memory Hierarchy by Compositing Automatic Transformations on Computations and Data. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. https://doi.org/10.1109/micro50266.2020.00044

[17] Tuowen Zhao, Mary Hall, Hans Johansen, and Samuel Williams. 2021. Improving communication by optimizing on-node data movement with data layout. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM. https://doi.org/10.1145/3437801.3441598