Extended Lattice-Based Memory Allocation

Alain Darte, Alexandre Isoard, Tomofumi Yuki

Extended Lattice-Based Memory Allocation

Alain Darte, Alexandre Isoard, Tomofumi Yuki

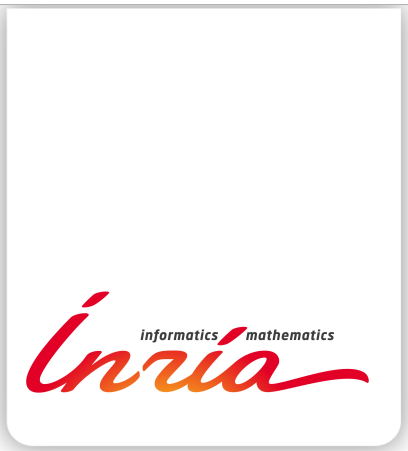
RESEARCH REPORT

N° 8840

Written in November 2015

Project-Team Compsys

ISSN 0249-6399 ISRN INRIA/RR--8840--FR+ENG



Extended Lattice-Based Memory Allocation

Alain Darte, Alexandre Isoard, Tomofumi Yuki

Project-Team Compsys

Research Report n° 8840 — Written in November 2015 — 28 pages

Abstract: This work extends lattice-based memory allocation, an earlier work on memory (array) reuse analysis. The main motivation is to handle in a better way the more general forms of specifications we see today, e.g., with loop tiling, pipelining, and other forms of parallelism available in explicitly parallel languages. Our extension has two complementary aspects. We show how to handle more general specifications where conflicting constraints (those that describe the array indices that cannot share the same location) are specified as a (non-convex) union of polyhedra. Unlike convex specifications, this also requires to be able to choose suitable directions (or basis) of array reuse. For that, we extend two dual approaches, previously proposed for a fixed basis, into optimization schemes to select suitable basis. Our final approach relies on a combination of the two, also revealing their links with, on one hand, the construction of multi-dimensional schedules for parallelism and tiling (but with a fundamental difference that we identify) and, on the other hand, the construction of universal reuse vectors (UOV), which was only used so far in a specific context, for schedule-independent mapping.

Key-words: Static optimization, polyhedral analysis, memory reuse and allocation, array contraction, integer lattices

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

Technique étendue d'allocation mémoire basée sur les réseaux entiers

Résumé : Ce travail étend l'allocation mémoire basée sur les réseaux entiers précédemment proposée en analyse de réutilisation mémoire (de tableaux). La motivation principale est de traiter de meilleure façon les formes plus générales de spécifications rencontrées aujourd'hui, comportant du tuilage de boucles, du pipeline, et d'autres formes de parallélisme exprimées dans les langages à parallélisme explicite. Notre extension a deux aspects complémentaires. Nous montrons comment nous pouvons prendre en compte des spécifications plus générales où les contraintes de conflit (celles qui décrivent les indices de tableaux qui ne peuvent pas partager le même emplacement mémoire) sont spécifiées par une union (non-convexe) de polyèdres. Au contraire des spécifications convexes, ceci requiert d'être capable de choisir des directions (c'est-à-dire une base) adéquates de réutilisation des cases de tableaux. Pour cela, nous étendons deux approches duales, précédemment proposées pour une base fixée, en des schémas d'optimisation permettant de choisir des bases adaptées. Notre approche finale consiste en une combinaison des deux approches, révélant également des liens avec, d'une part, la construction d'ordonnancements multi-dimensionnels pour le parallélisme et le tuilage (avec une différence fondamentale que nous identifions) et, d'autre part, la construction de vecteurs de réutilisation universelle (UOV), qui étaient utilisés jusqu'à présent uniquement dans un contexte spécifique, celui des allocations valides pour tout ordonnancement.

Mots-clés : Optimisations statiques, analyse polyédrique, réutilisation et allocation mémoire, contraction de tableaux, réseaux entiers

Extended Lattice-Based Memory Allocation

Alain Darte, Alexandre Isoard, Tomofumi Yuki

Compsys, LIP
UMR 5668 CNRS, INRIA, ENS-Lyon, UCB-Lyon
email: `Firstname.Lastname@ens-lyon.fr`

Abstract

This work extends lattice-based memory allocation, an earlier work on memory (array) reuse analysis. The main motivation is to handle in a better way the more general forms of specifications we see today, e.g., with loop tiling, pipelining, and other forms of parallelism available in explicitly parallel languages. Our extension has two complementary aspects. We show how to handle more general specifications where conflicting constraints (those that describe the array indices that cannot share the same location) are specified as a (non-convex) union of polyhedra. Unlike convex specifications, this also requires to be able to choose suitable directions (or basis) of array reuse. For that, we extend two dual approaches, previously proposed for a fixed basis, into optimization schemes to select suitable basis. Our final approach relies on a combination of the two, also revealing their links with, on one hand, the construction of multi-dimensional schedules for parallelism and tiling (but with a fundamental difference that we identify) and, on the other hand, the construction of universal reuse vectors (UOV), which was only used so far in a specific context, for schedule-independent mapping.

1 Introduction

As the gap between memory performance and compute power keeps increasing, the importance of efficient memory usage is also increasing. This is even more emphasized when exploiting hierarchical memories and/or when accelerators such as GPUs or FPGAs are used as they are often limited by the on-chip memory capacity and/or the bandwidth between the host and the accelerator. The problem of efficient memory allocation is further complicated by the trade-off between parallelism and memory usage.

Memory reuse is a standard technique for allocating scalar variables to registers. Memory reuse for arrays, in particular intra-array reuse, is used to reduce statically the memory footprint of data-intensive applications, after analyzing the liveness of the different elements of an array. The need for such array contraction is of course important for high-level program specifications (for example array languages) where the programmer expresses its applications in an abstract view of the storage locations, possibly even using arrays in single

assignment, thus without paying too much attention to memory usage. But such a memory allocation technique is also required within compilers themselves as a complementary step to many code transformations, for the design of intermediate buffers introduced by the compiler and the management of local memories, and to reduce the effect of some previous array expansion phases.

In this paper, we extend a technique called lattice-based memory allocation [9] that was originally proposed as a generalization of different strategies based on affine mappings with foldings by modulo operations (called *modular mappings*), formalized with integer lattices. The original work was aimed at handling regular kernels executed by sequential and/or limited forms of parallel schedules where simple optimization strategies appeared to be sufficient. We extend this framework on two main aspects:

- *conflict set* (a relation to express array elements that may not be mapped to the same memory location) as union of polyhedra, where the initial work is limited to a convex polyhedron;
- *optimized basis* heuristics to choose the direction of the modular mapping allocation, or the basis of the corresponding lattice.

The key insight is that optimizing the dimensions of a multi-dimensional modular mapping successively, greedily minimizing the resulting array size for each dimension, may lead to worse overall allocations, a phenomenon that is exacerbated when the conflict set is not convex. By designing an optimization procedure to build short reuse vectors (that indicate array elements mapped to the same location), sharing similarities with UOV (universal occupancy vector) based approaches [17, 18, 22], we address this issue and propose a combined heuristic to address this issue. Section 2 presents a simple example, in an informal way, to give the key intuition behind our approach and the reasons why previous approaches would fail to handle it efficiently. Section 3 gives some necessary background and formal definitions, defining more precisely the notions of conflict sets an modular mappings. Section 4 is the heart of our paper. It details the theory and algorithms we use to extend two previously-proposed dual approaches for memory mapping [8] to non-convex conflict sets and with optimized basis. It also shows how parametric solutions can be obtained and how the two heuristics can be combined. Section 5 illustrates on several examples our technique, showing how a library such as isl, for manipulating integer set relations, can be used to implement it. Finally, as our work shows multiple connections with a priori separated earlier work, we review in Section 6 some of them, detailing their links and differences with our technique. We conclude in Section 7.

2 Intuition of the Approach

In this section, we illustrate the key intuition behind our work using an example. There are many existing array mapping techniques that work well when the conflict set is described with a single polyhedron [2, 9, 14, 15]. We are interested in more complex cases which involve unions of polyhedra. A common situation that gives rise to such a non convex domain is when loop tiling [21] is applied and we need to store the live-out values of a tile [2]. Note that we present here

only a simplified view of the problem to give the important intuition. The actual problem is finding an allocation, given a schedule or a set of possible schedules, which may include parallelism and tiling, and whose legality is formulated using a relation, computed thanks to liveness analysis and representing the possible overlaps between the live-ranges of array elements.

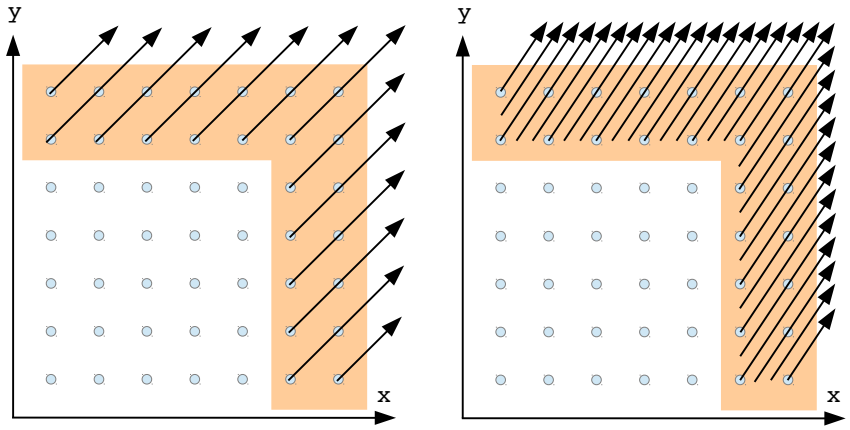
Let us consider a case where we seek an optimized allocation for a reverse-L shaped region depicted in Figure 1 (see also the figure caption for detailed explanations). The number of live-out values is $4N - 4$ if N is the length of the square edge (24 in the figure, with $N = 7$). One "good" affine allocation to maps all these values to different locations is to "project" along $(1, 1)$ with an additional modulo factor of 2 (as depicted in Figure 1a), which corresponds to an array of size $2(2N - 1)$, thus only 2 elements more than the "optimal", with the corresponding mapping $(x, y) \mapsto (x - y \bmod (2N - 1), y \bmod 2)$.

Existing techniques struggle to find this allocation for different reasons. One of the earliest, yet powerful, method for memory allocation by Lefebvre and Feautrier [14] consists in choosing some successive modulo folding for each dimension, restricting to mappings along the canonical axes. The first modulo should be larger than the maximal distance between two points (here $N - 1$ along the x axis), then the second modulo larger than the maximal distance between two points with the same value of x, which is also $N - 1$. This results here in an allocation of size N^2, with the corresponding mapping $(x, y) \mapsto (x \bmod N, y \bmod N)$.

Several other techniques have been presented that explore allocations using non-canonical projections or mappings [2, 9, 15]. We leave the detailed discussions of these work to Section 6 and only give here a high-level description of the most recent work by Bhaskaracharya, Bondhugula, and Cohen [2], which has the same objectives as ours. They proposed a method that combines ideas from multi-dimensional affine scheduling with memory allocation. The key idea is to interpret each dimension of a multi-dimensional affine mapping as a family of parallel hyperplanes that separate points from each other. If each parallel hyperplane contains at most one point of the domain to be stored, the corresponding affine function represents a legal memory allocation mapped to a one-dimensional array where the "latency" [11] (width or maximal distance), plus 1, is the number of elements in the array. Using this formulation, their approach first tries to find such a family of hyperplanes that contain at most one point each. If such hyperplanes cannot be found, their technique finds a second family of hyperplanes focusing on points that lie in the same hyperplane of the first family, and so on. The dimensionality of the final array is the number of linearly-independent hyperplanes generated.

One possible allocation that can be found by their technique is illustrated in Figure 1b. Since the mapping they compute also depends on how the domain is decomposed into a union of polyhedra, their technique may find other (worse) allocations, as explained in Section 6.2. The key reason of this inefficiency lies in the objective functions used. Their primary objective tries to minimize the number of hyperplane families, i.e., the dimensionality of the mapped array, by maximizing at each stage the number of polyhedra whose points are all separated by hyperplanes. Their secondary objective is to minimize the number of elements in the corresponding dimension, i.e., to find hyperplanes yielding minimal width.

The primary objective can be satisfied by using the hyperplanes shown in Figure 1b, parallel to the vector $(2, 3)$ (or similarly parallel to $(3, 2)$), thus



(a) (b)

Figure 1: An example to illustrate the key intuition. The reverse-L shape of width 2 may be viewed as the live-out values of a square tile of size N, when the data dependences (not depicted here) across tiles have a uniform length of 2 in each dimension. The memory allocation characterized by the reuse vectors in Figure 1a leads to the optimal affine mapping (projection plus modulo 2) that we seek. Figure 1b illustrates another possible affine mapping (a projection) that may be found by other techniques, with N extra storage.

orthogonal to $(3, -2)$, with width $5(N-1)$. The corresponding mapping is $(x, y) \mapsto 3x - 2y \bmod (5N-4)$. For $N = 7$, this gives, as depicted, 31 different memory locations. However, if a one-dimensional allocation cannot be found, the greedy heuristic will minimize the number of elements in the current dimension, which, as explained in Section 6.2, may favor in this example hyperplanes parallel to the canonical axes (i.e., the shortest one-dimensional schedules), leading to the same mapping as Lefebvre and Feautrier, with size N^2. The optimal solution is not obvious in this iterative formulation, since the "good" hyperplanes parallel to $(1, 1)$ (i.e., orthogonal to $(1, -1)$) do not satisfy the primary objective. Indeed, some points are still mapped to a common location, and it is not optimal for this dimension because its corresponding width is $2(N-1)$ while $N-1$ is achievable.

In our work, we use a different formulation to overcome inefficiency in these cases. For our example, it can be intuitively explained as finding the shortest vector that points to somewhere outside of the domain of interest, which is the vector $(2, 2)$. In general, we need multiple linearly-independent vectors, captured through lattices.

3 Background

We now introduce the two key concepts used in our paper, *conflict set*, which gives the constraints for valid mappings, and *modular mappings*, a particular form of functions used for intra-array reuse.

3.1 Conflict Set

Lattice-based memory allocation [8, 9], as well as all prior work on intra-array reuse [10, 15, 14], is based on the concept of **conflicting (array) elements**, i.e., the set of pairs of elements that should not be mapped to the same location. It is expressed as a binary relation $\bowtie$, which is the counterpart, for array elements, of the well-known interference graph defined for register allocation.

In register allocation, vertices correspond to scalar variables, and edges indicate that two variables should not be mapped to the same register, so that graph coloring can be used to derive a valid register assignment. For intra-array reuse, this interference graph (the $\bowtie$ relation) is not expressed in extension, but in a symbolic way, e.g., with polyhedra specifying a set of conflicting indices for a given array A. This specification is then used to derive a **valid mapping** σ, i.e., a mapping such that $\sigma(\vec{i}) \neq \sigma(\vec{j})$ if $\vec{i} \bowtie \vec{j}$ (meaning that $A(\vec{i})$ and $A(\vec{j})$ should not be mapped to the same location).

For intra-array reuse, the construction of the conflicting indices requires some symbolic liveness analysis. Such array mappings can be defined in a post-scheduling phase, i.e., for a particular execution or schedule (schedule-dependent mappings [10, 14, 15]), or before the final schedule is defined (schedule-independent mappings [17, 18, 22]), so that the mapping is valid for any further code transformation (or a subclass, such as loop tiling). The latter situation arises also when compiling programs, expressed in a parallel language such as OpenMP or X10, on top of a runtime system, in which case the exact schedule is not statically known.

Actually, all these cases are particular instances of the more general problem of defining liveness analysis and "simultaneously live" array elements in explicitly-parallel specifications. How to build such a relation $\bowtie$ is not our concern here: we assume it has been computed, possibly over-approximated, and that it expresses, for each array to be contracted, the set of pairs of conflicting indices (or their differences) in a compact symbolic form. For optimization purposes, we will mainly discuss the case where the conflicting differences are the integer points in a union of polyhedra $\mathcal{K} = \mathcal{P}_1 \cup \cdots \cup \mathcal{P}_r$. But the theory can be used in more general situations, e.g., with formulas in Presburger arithmetic as available in the Integer Set Library [19], or even polynomial expressions [13], as long as the corresponding optimizations can be carried out.

3.2 Modular Mappings

In lattice-based memory allocation, the mappings are restricted, both for optimization and code generation purposes, to **modular mappings**. A modular mapping $(M, \vec{b})$, defined by a $p \times n$ integral matrix M and a positive integral vector $\vec{b}$ of dimension p, maps the index $\vec{i}$ of a n-dimensional array to $\sigma(\vec{i}) = M\vec{i} \bmod \vec{b}$ (the modulo is applied component-wise) in a p-dimensional array of shape $\vec{b}$. The **size** of the mapping is the size of the resulting array, i.e., the product of the elements of $\vec{b}$. Its **dimension** is the dimension of the resulting array, i.e., p. As modular mappings are affine, it is enough to work with the symmetric set of **conflicting differences** $\mathcal{K} = \{(\vec{i} - \vec{j}) \mid \vec{i} \bowtie \vec{j}\}$. The initial theory of lattice-based memory allocation [8, 9] focuses on the case where $\mathcal{K}$ is the set of integer points in a polyhedron, in which case lower and upper bounds on memory size can be given and more properties proved.

Although we focus on intra-array reuse in this paper, it is worth to point out that modular mappings can be used in other contexts, e.g., for bank allocation, to allow parallel accesses to memory [5, 9], or more generally whenever renewable resources need to be shared.

4 Greedy Mapping and Lattice Constructions

In this section, we present our main contribution that extends lattice-based memory allocation. Our work is based on two dual approaches proposed by Darte et al. [8] that are equivalent when the set of conflicting differences $\mathcal{K}$ is a polyhedron. We show how we can extend them to handle the case where $\mathcal{K}$ is a union of polyhedra, and then to further optimize the selection of the basis (i.e., the matrix M in the modular mapping) to reduce the memory size. The resulting optimizations lead to two complementary greedy approaches, in which the rows of the matrix M are optimized in the opposite order. We then show in Section 4.3 how to combine them to try to get the best of the two worlds, the first one being well suited to handle parameters while the second one is more suitable to detect directions with constant (i.e., non-parametric) reuse.

Both approaches define mappings given a basis of $\mathbb{Z}^n$, i.e., how to choose the modulo vector $\vec{b}$. The first approach directly works with the matrix M of the mapping while the second one works with its kernel, i.e., the set of vectors $\vec{i}$ such that $\sigma(\vec{i}) = 0$. Such a set is a *lattice* of $\mathbb{Z}^n$, thus the name of the technique. Section 4.1 extends the former over the mapping space, Section 4.2 extends the latter over the lattice space. We combine the two in Section 4.3. Unimodular matrices (invertible in the integers) play an important role in this construction, both for the optimizations in the two approaches, and for their combination. In this case, the first approach builds the rows of M, while the second builds the column of M^{-1}.

4.1 Basis Selection in Mapping Space

The following mechanism [8] is the "successive modulo" principle [14], generalized to any set of n linearly independent integral vectors. $\mathcal{K}$ is the 0-symmetric set of conflicting differences.

Heuristic 1.

- *Choose n linearly independent integral vectors $(\vec{c}_1, \ldots, \vec{c}_n)$.*
- *Compute $F_i^*(\vec{c}_i) = \sup\{\vec{c}_i.\vec{z} \mid \vec{z} \in \mathcal{K}, \vec{c}_j.\vec{z} = 0, \forall j < i\}$, successively for all $1 \leq i \leq n$.*
- *Define M the matrix with row vectors $(\vec{c}_i)_{1 \leq i \leq n}$ and $\vec{b}$ an integer vector such that $b_i > F_i^*(\vec{c}_i)$.*

For each i, $F_i^*(\vec{c}_i)$ is the **width** along $\vec{c}_i$ of the intersection of $\mathcal{K}$ and the orthogonal of the vector space defined by $(\vec{c}_1, \ldots, \vec{c}_{i-1})$. Our goal is to adapt this heuristic when $\mathcal{K}$ is described with a union of polyhedra and to design an optimization to choose a suitable basis.

Theorem 1. *The modular mapping built by Heuristic 1 is a valid mapping for $\mathcal{K}$, assuming that $\mathcal{K}$ is bounded, symmetric w.r.t. $\vec{0}$.*

Proof. Let $\vec{x} \in \mathcal{K}$ with $M\vec{x} \bmod \vec{b} = 0$. Since $b_1 > F_1^*(\vec{c}_1) = \sup\{\vec{c}_1.\vec{x} \mid \vec{x} \in \mathcal{K}\} = \sup\{|\vec{c}_1.\vec{x}| \mid \vec{x} \in \mathcal{K}\}$ (as $\mathcal{K}$ is 0-symmetric), $\vec{c}_1.\vec{x} = 0 \bmod b_1$ implies $\vec{c}_1.\vec{x} = 0$. Then, $\vec{c}_2.\vec{x} = 0 \bmod b_2$, but $b_2 > F_2^*(\vec{c}_2) = \sup\{|\vec{c}_2.\vec{x}| \mid \vec{x} \in \mathcal{K}, \vec{c}_1.\vec{x} = 0\}$, thus $\vec{c}_2.\vec{x} = 0$. Continuing this process, we get $\vec{c}_i.\vec{x} = 0$ for all i. Since the $\vec{c}_i$ are n linearly independent vectors, this implies $\vec{x} = 0$. This shows that the modular mapping $\sigma = (M, \vec{b})$ is valid. Indeed, if $\vec{i} \bowtie \vec{j}$, then $\vec{i} - \vec{j} \in \mathcal{K}$ and thus, unless $\vec{i} = \vec{j}$, $\sigma(\vec{i}) \neq \sigma(\vec{j})$. □

Theorem 1 shows that, although it was initially designed assuming that $\mathcal{K}$ is a polyhedron [8], Heuristic 1 is actually valid with weaker hypotheses. Also, the following important properties remain true:

- The different values F_i^* depend on the order in which the vectors $(\vec{c}_i)_{1 \leq i \leq n}$ are considered. Considering all $n!$ orders can help reducing the size of the mapping. (In practice, n is small.)
- Increasing the values of $\vec{b}$ keeps the validity of the mapping. This can be used for example to restrict the values of $\vec{b}$ to power of 2. Also, if $\mathcal{K}$ is a union of different 0-symmetric pieces, one can compute a vector $\vec{b}_i$ for each piece, each obtained with a particular order of the basis, and then take the maximum, component-wise, of the $\vec{b}_i$ to get a valid $\vec{b}$ for the whole $\mathcal{K}$.
- The same proof as for Theorem 1 shows that the modular mapping $\vec{c}_1.\vec{x} + b_1(\vec{c}_2.\vec{x} + b_2(\cdots + b_{n-1}\vec{c}_n.\vec{x})) \bmod \prod_i b_i$ is valid, of dimension 1, and with same size. In other words, the dimension of a mapping is not related to its size, and some 1D mappings can even be found with a multi-dimensional approach.

4.1.1 Optimizing the Basis

We now show how the previous heuristic can be extended to define an optimized version, where the basis $(\vec{c}_i)_{1 \leq i \leq n}$ is built in a greedy fashion, so that $F_i^*(\vec{c}_i)$ is minimized at each step:

$$\vec{c}_i = \text{argmin}\{F_i^*(\vec{c}) \mid \vec{c} \in \mathbb{Z}^n \text{ linearly independent with } \vec{c}_j, j < i\}$$

This is a min-max problem, which can be solved, when $\mathcal{K}$ is a union of polyhedra, thanks to either the duality theorem of linear programming or the affine form of Farkas lemma [16], in the same way schedules with minimal latencies ($\sim$ width) are built [12, 7]. Let us detail the technique with Farkas lemma, which we recall here.

Lemma (Farkas, affine form)**.** *Let P be a non-empty polyhedron defined by affine inequalities $P = \{\vec{x} \mid Q\vec{x} \leq \vec{e}\}$. Then $\vec{c}.\vec{x} \leq \delta$ for all $\vec{x} \in P$ iff there exists $\vec{y} \geq \vec{0}$ such that $\vec{c} = \vec{y}.Q$ and $\vec{y}.\vec{e} \leq \delta$.*

Now, the width of P along a vector $\vec{c}$ can be computed as follows:

$$\begin{aligned} \max\{\vec{c}.\vec{x} \mid \vec{x} \in P\} &= \min\{\delta \mid \vec{c}.\vec{x} \leq \delta, \forall \vec{x} \text{ s.t. } Q\vec{x} \leq \vec{e}\} \\ &= \min\{\delta \mid \vec{y} \geq \vec{0}, \vec{c} = \vec{y}.Q, \vec{y}.\vec{e} \leq \delta\} \end{aligned}$$

Similarly, the quantity $F_i^*(\vec{c})$ is equal to:

$$F_i^*(\vec{c}) = \min\{\delta \mid \vec{y} \geq \vec{0}, \vec{c} = \vec{y}.Q + \vec{z}.C_{i-1}, \vec{y}.\vec{e} \leq \delta\} \quad (1)$$

where C_{i-1} is the matrix whose rows are $\vec{c}_1, \ldots, \vec{c}_{i-1}$, and the vector $\vec{z}$ does not need to be nonnegative. Finally, for a union of polyhedra $\mathcal{K} = \cup_{1 \leq j \leq r} \mathcal{P}_j$ where $\mathcal{P}_j = \{\vec{x} \mid Q_j \vec{x} \leq \vec{e}_j\}$, one just need to collect the different constraints expressed in Equation 1, with a $\vec{y}_j$ and a $\vec{z}_j$ for each P_j, plus the additional constraints $\vec{y}_j.\vec{e}_j \leq \delta$ for all $1 \leq j \leq r$. It remains to find $\vec{c}$ such that $F_i^*(\vec{c})$ is minimized, which amounts to solve a linear program, with objective function δ, where all these different variables, $\vec{c}$, δ, $\vec{y}_j$ and $\vec{z}_j$, are unknowns.

4.1.2 Linear Independence and Unimodularity

With no additional constraint, the optimal solution is of course $\vec{c} = \vec{0}$. But $\vec{c}$ should be restricted so that it is linearly independent with all $\vec{c}_j$ with $j < i$, and nonzero if $i = 1$. One way to do it is to complete (e.g., by computing the Hermite normal form), at each step, the vectors $(\vec{c}_j)_{j<i}$, into a n-dimensional basis, with additional vectors $(\vec{d}_j)_{j \geq i}$, and to write $\vec{c} = \sum_{j<i} \lambda_j \vec{c}_j + \sum_{j \geq i} \lambda_j \vec{d}_j$. Then, it is sufficient to solve one linear program for each $j \in [i \ldots n]$ with the additional constraint $\lambda_j \geq 1$, and to define $\vec{c}_i$ as the best of these $(n - i + 1)$ solutions. There is no need to check for $\lambda_j \leq -1$ since $\vec{c}$ and $-\vec{c}$ lead to similar mappings. Another trick is to select a random integer vector $\vec{r}$ and to add a single constraint $\vec{c}.\vec{r} \geq 1$ (or a similar reasoning with the λ_j). Unless bad luck, this would be enough to not miss the optimal with a single linear program. Alternate solutions are possible, e.g., by constructing a basis of the orthogonal of the vector space defined by $(\vec{c}_j)_{j<i}$ or a pseudo-inverse (see also the discussion on Pluto scheduler [4, 1] in Section 6.1).

Also note that, since $F_i^*(\vec{c}) = F_i^*(\vec{c} - \sum_{j<i} \lambda_j \vec{c}_j)$, one can restrict the search to vectors $\vec{c} = \sum_{j \geq i} \lambda_j \vec{d}_j$, i.e., such that $\lambda_j = 0$ if $j < i$. The resulting basis $(\vec{c}_i)_{1 \leq i \leq n}$ (i.e., the matrix M) will then be automatically unimodular (i.e., with determinant ± 1). This property is not formally needed to define a mapping but it makes the construction easier. In particular, completing at each step the vectors into a basis is just one iteration of the Hermite form computation. Also, imposing $\vec{c}$ to have integer components is then equivalent to looking for integer values for the λ_j. To see this, suppose that, before Step i of the heuristic, we have built a unimodular matrix U whose first $i - 1$ rows (matrix C_{i-1}) are the vectors $(\vec{c}_j)_{j<i}$ built so far, and the remaining rows (matrix D^i) are the vectors $(\vec{d}_j)_{j \geq i}$. At Step i, the chosen solution $\vec{c}_i$ is such that $\lambda_j = 0$ for $j < i$ and the λ_j, for $j \geq i$, have no common divisor d (otherwise a better solution can be defined by dividing by d). Thus, there is a unimodular matrix V_i of size $(n - i + 1)$ such that $(\lambda_i, \ldots, \lambda_n)$ is the first row of V_i. Then:

$$\begin{pmatrix} I_{i-1} & 0 \\ 0 & V_i \end{pmatrix} \begin{pmatrix} C_{i-1} \\ D_i \end{pmatrix} = \begin{pmatrix} C_{i-1} \\ V_i D_i \end{pmatrix}$$

is a unimodular matrix whose first i rows are the $(\vec{c}_j)_{j \leq i}$. This technique can thus be used to enforce a final unimodular matrix but also to complete the basis at each step, on the fly.

4.1.3 Handling parameters

In practice, the set of conflicting pairs and the set of conflicting differences are parameterized by structure parameters from the program, such as loop or array

bounds. Heuristic 1 has the nice property that it can easily handle parameters as long as they constrain $\mathcal{K}$ in an affine way. If $P_j = \{\vec{x} \mid Q_j\vec{x} \leq E_j\vec{n} + \vec{e}_j\}$. The standard technique [12] is to bound the width as an affine function of the parameters $\Delta\vec{n} + \delta$ and to apply Farkas lemma considering that $\vec{n}$ as a variable too, possibly taking into account additional affine constraints on the parameters. We then get a set of constraints as before, this time with the variables δ and Δ, with the objective of minimizing $\Delta\vec{n}+\delta$. Classically, parameters are then ordered with some priority so as to optimize the width in a lexicographic manner w.r.t. this order. The width is then parametric (thus the corresponding modulo), but the vector $\vec{c}$ built at each step has constant components thus all mechanisms presented before for basis completion, linear independence, and unimodularity remain true.

4.1.4 Wrap Up

The optimized version of Heuristic 1 presented here as some similarities with multi-dimensional scheduling and tiling, as explained in Section 6.1. It can help finding better mappings than without basis optimization, especially when $\mathcal{K}$ is not a polyhedron. However, unlike multi-dimensional scheduling for maximal parallelism detection for which a greedy approach is asymptotically optimal [6], selecting the smallest width at each step can be sub-optimal, even in order of magnitude, because it may be better to select a direction with larger width if the next width is then smaller in the orthogonal.

This situation arises for the example of Section 2, where the heuristic will simply select the canonical basis with both widths equal to N, i.e., with no array reuse. See more detailed explanations in Figure 2 which depicts the set $\mathcal{K}$ of differences of the live-out points in Figure 1. A more complex optimized variant of Heuristic 1 has been proposed [2] to try to avoid this caveat, but as explained in Section 6.2, it can also fail to find directions of reuse with a small (constant) modulo. The lattice-based heuristic proposed in the next section is aimed to avoid this pitfall.

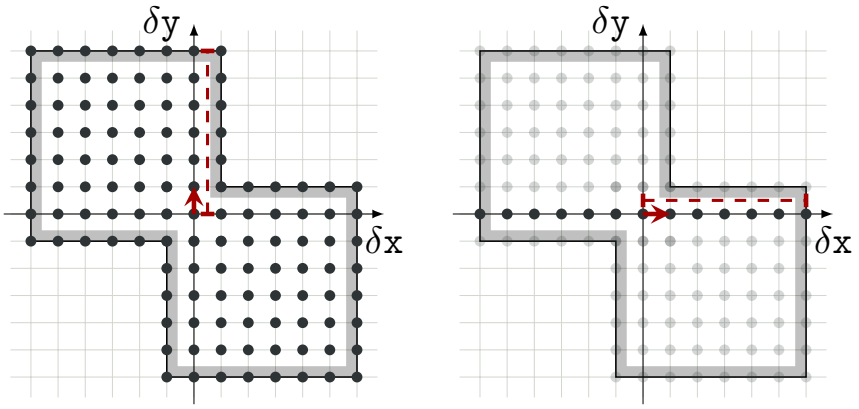
4.2 Basis Selection in Lattice Space

A dual approach to derive a modular mapping $(M, \vec{b})$ is to build an **integer strictly admissible lattice** for $\mathcal{K}$. Given n linearly independent vector $(\vec{a}_i)_{1\leq i\leq n}$, the lattice generated by $(\vec{a}_i)_{1\leq i\leq n}$ is the set $\Lambda = \{\vec{x} \mid \vec{x} = A\vec{u},\, \vec{u} \in \mathbb{Z}^n\}$ where A is the matrix with column vectors $(\vec{a}_i)_{1\leq i\leq n}$. It is strictly admissible for $\mathcal{K}$ if $\lambda \cap \mathcal{K} = \{\vec{0}\}$. A modular mapping is valid if and only if its kernel is an integer strictly admissible lattice for $\mathcal{K}$. Also, from such a lattice Λ, one can build a valid modular mapping whose kernel is Λ and whose size is equal to the determinant of Λ, i.e., $\det(A)$. This is the underlying idea of lattice-based memory allocation [8, 9].

The following scaling mechanism [8] gives a way to build a strictly admissible lattice from a set of n linearly independent integer vector $(\vec{a}_i)_{1\leq i\leq n}$. Again, we want to check if it still works for any $\mathcal{K}$, not just for a convex polyhedron $\mathcal{K}$ as initially formulated, then derive mechanisms to optimize the basis it works with.

Heuristic 2.

- *Choose n linearly independent integral vectors $(\vec{a}_1, \ldots, \vec{a}_n)$.*



(a) (b)

Figure 2: Optimized version of Heuristic 1 for the set $\mathcal{K}$ of differences of the live-out points in Figure 1. The selected basis is $\vec{c}_1 = (0,1)$, then $\vec{c}_2 = (1,0)$ (or the converse). The left figure shows the width $F_1^*(\vec{c}_1) = N - 1$ computed for the first vector. In the next step of the heuristic, $\mathcal{K}$ is restricted to the orthogonal space of the first vector, and $F_2^*(\vec{c}_2) = N - 1$ is computed as illustrated on the right. The resulting mapping is $(x, y) \mapsto (y \bmod N, x \bmod N)$, which is not a good allocation for this example. This is because the basis vectors are greedily selected to minimize the width at each dimension, missing the "hole" along $(1, 1)$.

- *Compute* $F_i(\vec{a}_i) = \inf\{\lambda \geq 0 \mid \vec{a}_i \in \lambda\mathcal{K}_i\}$*, successively for all* $1 \leq i \leq n$*, where* $\mathcal{K}_i = \mathcal{K} + \mathit{Vect}(\vec{a}_1, \ldots, \vec{a}_{i-1})$*.*
- *Define the lattice* Λ *generated by the vectors* $(\rho_i \vec{a}_i)_{1 \leq i \leq n}$ *where the* ρ_i *are integers such that* $\rho_i > 1/F_i(\vec{a}_i)$*.*

For each i, $\mu\vec{a}_i$ with $\mu = 1/F_i(\vec{a}_i)$ is the largest "multiple" of $\vec{a}_i$ that belongs (if $\mathcal{K}$ is closed) to the **extrusion** $\mathcal{K}_i$ of $\mathcal{K}$ along the vectors $(\vec{a}_j)_{j<i}$, i.e., $\mathcal{K}_i = \mathcal{K} + \mathrm{Vect}(\vec{a}_1, \ldots, \vec{a}_{i-1}) = \{\vec{x} \mid \vec{x} = \vec{y} + \sum_{j<i} \alpha_j \vec{a}_j,\ \vec{y} \in \mathcal{K}\}$. In other words, $\mu\vec{a}_i - \sum_{j<i} \alpha_j \vec{a}_j \in \mathcal{K}$ for some real numbers $(\alpha_j)_{j<i}$ and, for any $\rho > \mu$, $\rho\vec{a}_i \notin \mathcal{K}_i$.

Theorem 2. *The lattice* Λ *built by Heuristic 2 is a strictly admissible integral lattice for* $\mathcal{K}$*, if* $\mathcal{K}$ *is bounded, symmetric w.r.t.* $\vec{0}$*.*

Proof. Let $\vec{x} \in \Lambda$. If $\vec{x} \neq \vec{0}$, one can write $\vec{x} = \sum_{j=1}^{i} u_j \rho_j \vec{a}_j$ for some integers $(u_j)_{j \leq i}$ with $u_i \neq 0$. Suppose $\vec{x} \in \mathcal{K}$. If $\mathcal{K}$ is 0-symmetric, one can assume $u_i \geq 1$ without loss of generality. Then $\vec{a}_i = (\vec{x} - \sum_{j<i} \vec{a}_j)/(\rho_i u_i)$, thus $F_i(\vec{a}_i) \leq 1/(\rho_i u_i)$ by definition of $F_i(\vec{a}_i)$, then $F_i(\vec{a}_i) \leq 1/\rho_i$, which is impossible by definition of ρ_i. Thus, Λ is a strictly admissible integral lattice for $\mathcal{K}$. □

4.2.1 Optimizing the Basis

We now assume that $\mathcal{K}$ is **star-shaped**, i.e., if $\vec{x} \in \mathcal{K}$, then $\lambda\vec{x} \in \mathcal{K}$ for all $0 \leq \lambda \leq 1$. In this case, ρ_i is the smallest integer such that $\rho_i \vec{a}_i \notin \mathcal{K}_i$. Theorem 2 shows that Heuristic 2, as for Heuristic 1, is still valid even if $\mathcal{K}$ is not a polyhedron. But how can we optimize the basis $(\vec{a}_i)_{1 \leq i \leq n}$? One could think that minimizing $\lceil 1/F_i(\vec{a}_i) \rceil$ is the right thing to do, i.e., that it is sufficient to pick

any vector outside the ith extrusion $\mathcal{K}_i$ of $\mathcal{K}$ along the previously-built vectors (in which case $\rho_i = 1$ can be chosen). This is of course not true: unlike for Heuristic 1, the size of the resulting mapping cannot be read directly from the ρ_i, it depends also on the determinant of the $\vec{a}_i$. It seems thus more profitable to directly optimize the basis vectors $\vec{l}_i = \rho_i \vec{a}_i$ of Λ, e.g., by minimizing their norm, for some adequate norm to be defined. This is because the determinant, in absolute value, is bounded by the product of the Euclidian norms (and all norms have the same order of magnitude). Also, if the direction $\vec{a}_i$ is chosen, the best solution is to choose $\vec{l}_i$ to be the smallest integral vector out of the extrusion, in the direction of $\vec{a}_i$.

Another indication of why looking for small $\vec{l}_i$ is more likely to be good is that each $\vec{l}_i$ is a **reuse vector** (or occupancy vector in the UOV terminology [17]), i.e., $A(\vec{x})$, $A(\vec{x}+\vec{l}_i)$, ..., $A(\vec{x}+k\vec{l}_i)$ will reuse the same memory location. The mapping will exploit more reuse if a larger number of copies of $\vec{l}_i$ can traverse the original array space thus, intuitively, a smaller vector will lead to more reuse. However, as for the UOV optimization [17], the best direction remains difficult to anticipate: it depends on the extent of the next extrusion $\mathcal{K}_{i+1}$, i.e., of the other $\vec{a}_j$ not yet defined. Nevertheless, despite this inaccuracy, minimizing the norm is interesting because it will select, in priority, non-parametric reuse vectors, thus favor the reduction of the mapping size in order of magnitude (if there is one large parameter N and p constant reuse vectors are found, the mapping size will be of order N^{n-p}). For all these reasons, at each step of the heuristic, we will look for an integral vector with minimal norm that is out of the extrusion. For computation reasons, we will choose a norm that can be minimized with linear programming, for example $||.||_\infty$ (max of the absolute value of components) or $||.||_1$ (sum of the absolute value of components, i.e., Manhattan distance). To break ties, we can also use the norm proposed for the computation of AUOV [18], which is a two-dimensional lexicographic optimization, first the Manhattan distance, then the sum of the absolute value of all differences of two components, which tends to lead to more "diagonal" vectors.

4.2.2 Linear Independence and Unimodularity

Unlike Heuristic 1 for which we perform an optimization based on an expression of $\mathcal{K}$, here we want to find a short vector **not in** $\mathcal{K}_i$, or at least on the border of $\mathcal{K}_i$. We tried many different optimization schemes, using Farkas lemma or the duality theorem, but we found no better solution than working directly with the complement of $\mathcal{K}_i$, expressed as a union of polyhedra. Then, we just need to minimize the norm in each piece, with integer linear programming, and to pick the best solution $\vec{l}_i$. As any linear combination of $(\vec{a}_j)_{j<i}$ belongs to $\mathcal{K}_i$, linear independence is automatically satisfied.

Now, let us see if we can enforce some unimodularity property and how it can be used to simplify the computations. Suppose the vectors $(\vec{a}_j)_{j<i}$ have been computed so that they can be completed into a unimodular matrix, thanks to additional vectors $(\vec{d}_j)_{j\geq i}$. We can look for the vector $\vec{l}_i$ expressed in this basis: $\vec{l}_i = \sum_{j<i} \lambda_j \vec{a}_j + \sum_{j\geq i} \lambda_j \vec{d}_j$. Then, we can either minimize the norm in the original basis (minimizing the components of $\vec{l}_i$) or in this new basis (minimizing the λ_j). In both cases, since $\vec{l}_i \notin \mathcal{K}_i$ then, by definition of $\mathcal{K}_i$, the same is true if we subtract to it any linear combination of $(\vec{a}_j)_{j<i}$. This does

not change $F_i(\vec{a}_i)$, the subsequent extrusions, and the determinant of the lattice. This amounts to assume that $\lambda_j = 0$ for $j < i$. Then, if ρ_i is the common divisor of the $(\lambda_j)_{j \geq i}$, we can select $\vec{a}_i = (\sum_{j \geq i} \lambda_j \vec{d}_j)/\rho_i$ so that $\vec{l}_i = \rho_i \vec{a}_i$. Finally, as we did for enforcing unimodularity in Heuristic 1, we can then complete $(\vec{a}_j)_{j \leq i}$ into a unimodular matrix, with one simple computation of the Hermite form. The basis of the final lattice Λ is then given as a unimodular matrix (A) times a diagonal one (the ρ_i).

Things are simpler when unimodularity is enforced. A valid mapping can be easily defined by $\sigma = (A^{-1}, \vec{\rho})$. Indeed, $\sigma(\vec{x}) = \vec{0}$ iff $A\vec{x} = \vec{0} \bmod \vec{\rho}$ iff there exists $\vec{y} \in \mathbb{Z}^n$ such that $A\vec{x} = R\vec{y}$ where $R = \mathrm{diag}(\rho_1, \ldots, \rho_n)$, i.e., $\vec{x} = A^{-1}R\vec{y}$, which means $\vec{x} \in \Lambda$. When A is unimodular, it is also simpler to build the set of integral vectors in $\overline{\mathcal{K}_i}$ when $\mathcal{K}$ is, as in the isl library, expressed by a formula involving only integer variables and affine inequalities. Indeed, in this case, $\mathcal{K}_i = \mathcal{K} \cap \{\vec{x} \mid \vec{x} = \sum_{j<i} \alpha_j \vec{a}_j,\ \alpha_j \in \mathbb{R}\} = \mathcal{K} \cap \{\vec{x} \mid \vec{x} = \sum_{j<i} \alpha_j \vec{a}_j,\ \alpha_j \in \mathbb{Z}\}$ is also defined with integer variables and affine inequalities, so expressible with isl. The set of integral vectors in its complement can then be computed by isl. It will also be described with integer variables and affine inequalities.

4.2.3 Handling Parameters

As we saw in Section 4.1, Heuristic 1 can be extended with no difficulties to the handle the case where $\mathcal{K}$ depends affinely on parameters. The final mapping may depend on these parameters, but not the matrix M, only the modulo vector $\vec{b}$. This is more complicated for Heuristic 2. If all vectors $(\vec{l}_i)_{1 \leq i \leq n}$ are constant or equal to $\lambda \vec{a}_i$ where only λ depends on the parameters, the previous construction can be performed and a valid mapping can be computed from the lattice. However, if this is not the case, i.e., if the direction of $\vec{l}_i$ depends on the parameters, then computing $\mathcal{K}_{i+1}$ involves quadratic constraints (α_i is multiplied by a parameter in the definition of $\mathcal{K}_{i+1}$), which is problematic. Similarly, defining $\vec{a}_i$ and completing the vectors $(\vec{a}_j)_{j \leq i}$ into a basis will need knowledge on the arithmetic properties (gcd) of the parametric components of $\vec{l}_i$.

An easy situation is when all vectors $\vec{a}_i$ built during the optimization process are constant, except possibly the last one. Indeed, for $i = n$, there is no $\mathcal{K}_{i+1}$ to build and no basis completion to perform. This is the case for the example of Section 2, where we find successively $\vec{l}_1 = \rho_1 \vec{a}_1 = (2, 2)^T$, then $\vec{l}_2 = \rho_2 \vec{a}_2 = (2N - 1, 0)^T$, which corresponds to the 2D mapping $(x, y) \mapsto (y \bmod 2, x - y \bmod (2N - 1))$. See detailed explanations in Figure 3. By construction, the 1D mapping $\sigma'(x, y) = x - y + (2N - 1)y \bmod 2(2N - 1)$ is also valid [8].

Finally, let us point out that as long as, in the process, we seek a constant reuse vector $\vec{l}_i$, then $\vec{l}_i$ belongs to $\overline{\mathcal{K}_i}$ for any value of the parameters, i.e., the intersection of all $\overline{\mathcal{K}_i}$ when the parameters vary, i.e., the complement of the union of all $\mathcal{K}_i$ when the parameters vary. This means that we can first project out the parameters in $\mathcal{K}$ (equivalent to the union when the parameters vary), then compute the successive extrusions and complement sets. In the example of Section 2, projecting out the parameter N in $\mathcal{K}$ gives the non-parametric complement set $\overline{\mathcal{K}}$ defined by $\{(x, y) \mid (x \geq 2,\ y \geq 2) \text{ or } (x \leq -2,\ y \leq -2)\}$, from which we easily find $\vec{l}_i = (2, 2)$. In general, we can thus first build a non-parametric set $\mathcal{K}$, by projecting out the parameters, and work with it as long as we find a constant reuse vector.

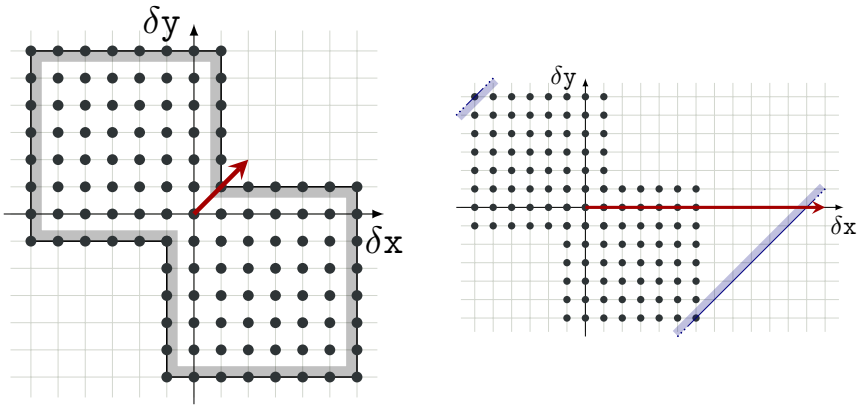


Figure 3: Optimized version of Heuristic 2 for the set $\mathcal{K}$ of differences of the live-out points in Figure 1. The selected basis is $\vec{a}_1 = (1,1)^T$, then $\vec{a}_2 = (1,0)^T$, or equivalently $(0,1)^T$. The figure to the left shows the basis vector of the lattice $\vec{l_1} = \rho_1\vec{a}_1 = (2,2)^T$, which is the shortest vector (a multiple of $\vec{a}_1$) that points outside of $\mathcal{K}$. In the next step, the set $\mathcal{K}$ is extruded along $\vec{a}_1$ and is now the infinite diagonal band shown in the right figure, leading to the second basis of the lattice: $\vec{l_2} = \rho_2\vec{a}_2 = (2N-1,0)^T$. The final mapping is given by the inverse of the matrix $[\vec{a}_1\vec{a}_2]^{-1} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & -1 \end{smallmatrix}\right]$ and the modulo factors defined by $\rho_1 = 2$ and $\rho_2 = 2N-1$, which is the mapping $(x,y) \mapsto (y \bmod 2, x-y \bmod (2N-1))$.

4.2.4 Star Shaping

There remains one potential problem, if $\mathcal{K}$ is not naturally star-shaped. In this case, $\mathcal{K}_i$ can have "holes" along a direction and the optimization of the norm in $\overline{\mathcal{K}_i}$ can lead to a small integral reuse vector $\vec{l_i} \in \overline{\mathcal{K}_i}$ such that $\lambda\vec{l_i} \in \mathcal{K}_i$ for some positive integer λ, resulting in an invalid lattice. A possibility is to ignore this problem and to check, a posteriori, that this does not happen. This can be done with integer linear programming if $\vec{l_i}$ is a constant vector. Another safer strategy is to first modify $\mathcal{K}$ into its star-shaped extension $\mathcal{K}^* = \{\vec{x} \mid \vec{x} = \lambda\vec{y},\ 0 \leq \lambda \leq 1, \vec{y} \in \mathcal{K}\}$. This however has to be done with care to avoid adding artificially integral points. Indeed, suppose that $\mathcal{K} = \{(0,0)\} \cup \{(x,y) \mid 2 \leq |y| \leq 3\}$. The star-shaped extension of $\mathcal{K}$ is an open set $\{(x,y) \mid 0 < |y| \leq 3\} \cup \{(0,0)\}$, while projecting naively λ in the expression of $\mathcal{K}^*$ would add the line $y = 0$. But, as we are interested only in integral vectors, we want to work with $\mathcal{K}^* = \{(0,0)\} \cup \{(x,y) \mid 1 \leq |y| \leq 3\}$. This can be done as follows. First suppose that $\mathcal{K}$ is the integral points in a polyhedron $\mathcal{P} = \{\vec{x} \mid A\vec{x} \leq \vec{b}\}$. We consider the set of integral points in $\mathcal{K}^* = \{\vec{x} \mid \exists\lambda,\ 0 < \lambda \leq 1,\ A\vec{x} \leq \lambda\vec{b}\}$ to which will be then added the vector $\vec{0}$. We then eliminate λ using the Fourier-Motzkin method, which leads to the following constraints:

$$\mathcal{K}^* = \{\vec{0}\} \cup \left\{ \vec{x} \;\middle|\; \begin{array}{ll} (A\vec{x})_i \leq b_i & \text{if } b_i > 0 \\ (A\vec{x})_i \leq 0 & \text{if } b_i = 0 \\ (A\vec{x})_i < 0 & \text{if } b_i < 0 \\ \dfrac{(A\vec{x})_i}{b_i} \leq \dfrac{(A\vec{x})_j}{b_j} & \text{if } b_j < 0 < b_i \end{array} \right\}$$

The inequality $(A\vec{x})_i < 0$ is then modified in $(A\vec{x})_i \leq -1$ since A and $\vec{x}$ have integral components. As the star-shaped extension of a union of polyhedra is the union of the star-shaped extension of each polyhedron, the previous technique gives an algorithm to star-shape any union of polyhedra. Note however that, as for the convex hull, it does not work, in general, for a parametric set as the result is not always affinely parametric. But, as we use this procedure to find constant reuse vectors, we can first project out the parameters from $\mathcal{K}$, then build the star-shaped extension $\mathcal{K}^*$ of this non-parametric $\mathcal{K}$ as we just explained. To illustrate this principle, suppose that $\mathcal{K} = \{(0,0)\} \cup \{(x,y) \in \mathbb{Z}^2 \mid 2 \leq |y| \leq 3, |x| \leq N\}$. Eliminating N produces $\{(0,0)\} \cup \{(x,y) \mid 2 \leq |y| \leq 3\}$, i.e., the set we discussed previously. Finally, with Fourier-Motzkin, we get the star-shaped extension $\{(0,0)\} \cup \{(x,y) \in \mathbb{Z}^2 \mid 0 < |y| \leq 3\}$, and, restricting to integer points, $\mathcal{K}^* = \{(0,0)\} \cup \{(x,y) \in \mathbb{Z}^2 \mid 1 \leq y \leq 3\}$ as expected. We can then compute the set of integer points not in $\mathcal{K}^*$ as $\{(x,y) \mid |y| \geq 4\} \cup \{(x,y) \mid |x| \geq 1, y = 0\}$.

4.2.5 Wrap Up

The optimized version of Heuristic 2 presented here can be viewed as a generalization of all approaches based on reuse/occupancy vectors: UOV [17], QUOV [22], AUOV [18], and even (pseudo)-projection methods [15]. See more details in Section 6.3. It is a strict generalization in the sense that our technique is the first *multi-dimensional* reuse vector technique, thanks to the concept of extrusion. Unlike Heuristic 1, it manipulates $\overline{\mathcal{K}}$, the complement of $\mathcal{K}$, and not $\mathcal{K}$ itself. This duality was also identified in the context of liveness analysis. In some specialized contexts (as in all previous work on reuse vectors) $\overline{\mathcal{K}}$, or an over-approximation of it, can be built directly, and not as the complement of $\mathcal{K}$.

In Figure 4, we describe how the different optimization criteria for the two heuristics favors one set of basis over the other. It is important to emphasize that the vectors for the two approaches have different meanings. One is the direction of the mapping, and the other is the reuse vector. Providing the equivalent mapping vectors that would be obtained from lattice-based approach, in the reverse order, i.e., $\vec{c}_1 = (1,-1)$ and $\vec{c}_2 = (0,1)$, to the Heuristic 1 yields the same modulos, but computed as widths, thus the same mapping.

Currently, our technique can be fully implemented only when one can guarantee that the reuse vectors produced are constant, i.e., do not depend on parameters. In contrast, Heuristic 1 is naturally adapted to parametric optimization. This motivates the development of a combination of the two heuristics as exposed in Section 4.3: we will first look for constant reuse vectors with Heuristic 2, then complete the lattice/mapping by optimizing the dimensions in the opposite order thanks to Heuristic 1, constrained by the reuse vectors already found by Heuristic 2.

Note that when p constant vectors are found, one could re-optimize in this vector space, thanks to a suitable enumeration of lattices using the Hermite normal form [9], to find the best one in this space. However, such a search is potentially expensive if the determinant is large. Also, the problem of finding the best constant strictly admissible sub-lattice in general seems open because we do not know in which subspace the search needs to be done.

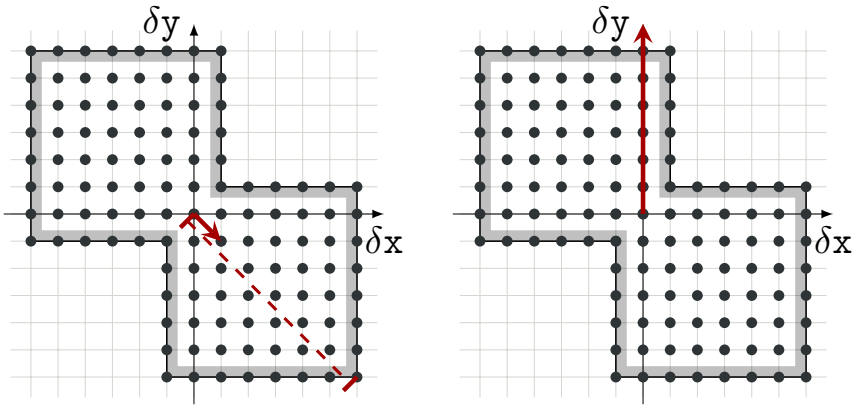


Figure 4: How the basis selection techniques for the two heuristics favor one basis over the other. The basis selection for the mapping space (Heuristic 1) favors, for the first vector, the canonical axis $(1, 0)$ (or equivalently $(0, 1)$) over $(1, -1)$ because the width along $(1, -1)$ is wider. As shown in the figure to the left, the width is defined by the vector $(N - 1, 1 - N)$ with a width equal to $2N - 2$, which is larger than $N - 1$ obtained from the canonical basis. In the lattice space, i.e., for Heuristic 2, the canonical basis leads to a much longer reuse vector (it has to be multiplied by $\rho = N$) as depicted in the right figure. Since a shorter (and constant) reuse vector is preferred in the basis selection, $(2, 2)$ is selected.

4.3 Combining the Two Approaches

The two previous approaches can be combined by understanding the connection between a valid mapping and an admissible lattice, in particular when the lattice has a "scaled" unimodular basis, i.e., is given by a unimodular basis A multiplied by a diagonal matrix $R = \text{diag}(\rho_1, \ldots, \rho_n)$. If $(\vec{a}_i)_{1 \leq i \leq n}$ are the columns of A, then the $\vec{l_i} = \rho_i \vec{a}_i$ form a basis of reuse vectors for the lattice. If $C = A^{-1}$ is the inverse of A, with row vectors $(\vec{c}_i)_{1 \leq i \leq n}$, then σ defined by $\sigma(\vec{x}) = C\vec{x} \bmod \vec{b}$ (following the notations of Heuristic 1) with $b_i = \rho_i$ is a valid mapping. Furthermore, if $\mathcal{K}$ is the set of integral points in a union of polyhedra, then if the ρ_i were computed, from $i = 1$ to n, following Heuristic 2 for the basis $(\vec{a}_i)_{1 \leq i \leq n}$, they are also equal to the b_i computed, from n to 1, following Heuristic 1 for the basis $(\vec{c}_i)_{1 \leq i \leq n}$. This can be proved the same way as for a single polyhedron [8], thanks to duality in linear programming.

Now, suppose that only the first i column vectors of A have been computed, for example because we were not able to find a constant vector $\vec{l}_{i+1}$ in the complement of $\mathcal{K}_{i+1}$. How can we complete them with $(n - i)$ parametric reuse vectors? These first i vectors do not fully constrain the final mapping, they only define a partial mapping with the first i vectors $(\vec{c}_j)_{j \leq i}$ of the inverse of A. However, they fully determine the vector space they generate, as well as its orthogonal, i.e., the vector space where the missing vectors $(\vec{c}_j)_{j > i}$ should lie. The idea is then to use Heuristic 1, constrained to this orthogonal, to optimize them and complete the mapping. This gives rise to the following combined heuristic.

Heuristic 3.

- *Project out the parameters in $\mathcal{K}$ to get a description $\mathcal{K}'$ of the union of constraints valid for all parameters.*
- *Build the star-shaped extension $\mathcal{K}'^*$ of this non-parametric $\mathcal{K}'$ as explained in Section 4.2.*
- *Use Heuristic 2 with $\mathcal{K}'^*$ to build optimized reuse vectors, until no reuse vector is found. If i is the number of vectors built, we get a unimodular matrix A with column vectors $(\vec{a}_j)_{1\leq j\leq n}$ such that for all $j \leq i$, $\vec{l}_j = \rho_j \vec{a}_j$ is the reuse vector that was built.*
- *Use Heuristic 1 with the initial set $\mathcal{K}$, restricting the search to the orthogonal defined by $(\vec{a}_j)_{j\leq i}$, to get $(n-i)$ optimized mapping vectors $(\vec{c}_j)_{n\geq j>i}$ (numbered in this decreasing order). Let $(w_j)_{n\geq j>i}$ be their corresponding successive widths.*
- *Define the matrix M with row vectors $(\vec{m}_j)_{1\leq j\leq n}$ such that $\vec{m}_j$ is the jth row of A^{-1}, if $j \leq i$, and $\vec{m}_j = \vec{c}_j$ if $j > i$. Define $\vec{b}$ with $b_j = \rho_j$ if $j \leq i$ and $b_j = w_j$ if $j > i$.*

Theorem 3. *The mapping $\sigma = (M, \vec{b})$ is a valid mapping for $\mathcal{K}$.*

Proof. Let $\vec{x} \in \mathcal{K}$ such that $M\vec{x} = \vec{0} \bmod \vec{b}$. With the same argument used in the proof of Theorem 1, and by definition of $(\vec{m}_j)_{j>i}$ in Heuristic 3 and of $(b_j)_{j>i}$ in Heuristic 1, we first get $\vec{c}_n.\vec{x} = 0$, then successively $\vec{c}_j.\vec{x} = 0$ for all j from n to $(i+1)$. Thus $\vec{x}$ belongs to the orthogonal of $(\vec{c}_j)_{j>i}$, i.e., the vector space spanned by $(\vec{a}_j)_{j\leq i}$. As A is unimodular, we can write $\vec{x} = \sum_{j\leq i} \lambda_j \vec{a}_j$ with integers λ_j. As the first i rows of M are those of the inverse of A, $\vec{m}_j.\vec{x} = \lambda_j$ for $j \leq i$. Thus λ_j is a multiple of $b_j = \rho_j$ as defined in Heuristic 2. With the same arguments used in the proof of Theorem 2, we conclude that $\vec{x} = \vec{0}$, which means that the modular mapping defined by M and $\vec{b}$ is valid. □

To force the search in the orthogonal of the $(\vec{a}_j)_{j\leq i}$, we can look for an integer linear combination of the last $(n-i)$ rows of A^{-1} only. Then, using the same principles as for Heuristic 1, we can impose that the vectors $(\vec{c}_j)_{j>i}$ form a unimodular basis of this orthogonal, which, combined with the first i rows of A^{-1}, will form a $n \times n$ unimodular matrix M. There is thus in this case a complete correspondence between the b_i found through Heuristic 1 for the row vectors of M, starting from the last one, and the ρ_i found through Heuristic 2 for the column vectors of M^{-1} starting from the first one. But what is important here is that by enforcing the search of the mapping vectors $(\vec{c}_j)_{j>i}$ to the orthogonal of small reuse vectors $(\vec{a}_j)_{j\leq i}$, we find potentially larger widths for the first $(\vec{c}_j)_{j>i}$. But this is on purpose, to make sure we do not miss a good directions of reuse later. This is what happens on the example of Section 2 if we apply the combined heuristic. We first find the reuse vector $(2, 2)$, then we limit the search to the orthogonal direction. Here, in 2D, we directly get the mapping vector $(1, -1)$. In general, this remains a heuristic as the orthogonal of the reuse vector may not be the right direction in terms of width, although it guarantees the right order of magnitude for the size of the mapping. This concludes the formal description of our combined optimization.

5 Evaluation

We validated our technique by running the different optimization steps, as a script, with the iscc calculator [20]. As iscc does not provide sufficient genericity, the script has to be tailored to each example but a generic implementation could be done using isl [19].

5.1 Reverse-L Shaped Region and Optimizing Scripts

Our illustrating example can be solved using the following script:

```
Kp := {[N]->[x,y]: N>=2 and
    (-1<=-x,y<N or -1<=x,-y<N)};
K_1 := range Kp;
nK_1 := {[x,y]} - K_1;
Norm := { [x,y]->[m,-x,-y]: m >= x,y,-x,-y };
a_1 := (Norm^-1)(lexmin (Norm(nK_1)));
```

`Kp` represents $\mathcal{K}$, with the exception that its parameters are variable (to circumvent iscc's impossibility of projecting parameters out). We then build `K_1` by projecting those parameters. A star-shaping step should also take place (but again is not easy to do with iscc) but would give the same polyhedron as it is already star-shaped (and $\vec{0}$ is also in each basic set). We then compute `nK_1` as the complement of `K_1`. The `Norm` map provides the $||.||_\infty$ norm, which we want to minimize (the $||.||_1$ norm would give the same result here as the optimal solution is a vertex of `nK_1`). The inversion of the x and y coordinates is cosmetic and is so that we choose positive coefficients for equivalent vectors as a side effect of minimization. We then proceed to find the smallest vector, which, in our example, leads to $\vec{l_1} = (2,2)^T$ as expected, providing our first lattice vector $\vec{a}_1 = (1,1)^T$ and its associated modulo 2. The search for the next vector will be unfructuous but is executed as follows:

```
Extr_1 := {rat: [x,y]->[x',y']:
    (exists e: x'=x+1*e and y'=y+1*e)};
Unimod_1 := range {[t]->[1*t,0*t]: t>0};
K_2 := coalesce Extr_1(K_1);
nK_2 := {[x,y]}-K_2;
a_2 := (Norm^-1)(lexmin (Norm(nK_2*Unimod_1)));
```

where `Extr_1` is an extrusion operator along $\vec{a}_1$ (notice the use of a rational map) and `Unimod_1` is a constraint enforcing the unimodularity of the future basis. It is obtained by looking for vectors colinear to the vectors (here only one, $(1,0)^T$) that complete $\vec{a}_1$ into a unimodular basis. Here, `K_2` is now the whole space, due to the parameter projection, so `nK_2` is empty, which shows that we cannot find a second constant reuse vector to complete the first. If we did not project the parameters (and detected that `K_2` was already star-shaped) then we would have found $\vec{l_2} = (2N-1,0)$, which is a valid parametric reuse vector (see again Figure 3).

Now, let us assume that we did not find that last vector because we stopped Heuristic 2 as soon as it does not find a constant vector. We complete the vector $\vec{a}_1$ into a unimodular basis as previously, for example with the vector

$(1,0)$, then we compute its inverse: $\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{-1} = \begin{pmatrix}0 & 1\\ 1 & -1\end{pmatrix}$. This already gives the first vector $\vec{c_1} = (0,1)$ of the mapping (first row) and its associated modulo equal to 2. We then continue with Heuristic 1:

```
K := range ([n]->{[n]->[x,y]} * Kp);
fK := (unwrap coefficients K)^-1;
Opt := {[c_cst,c_n]->[c_n,c_cst]};
Ortho_1 := range {[t]->[-1*t,1*t]: t>0};
b := (fK.Opt)*Ortho_1;
b_min := lexmin range b;
c_2 := sample (b^-1)(b_min);
```

The set `K` is the parameterized `Kp` and `fK` is the dual of `K` in the sense of Farkas lemma. `Opt` is used to order the coefficients in the correct order for the minimization: we want a small coefficient for N, then a small constant coefficient. `Ortho_1` constraints the vector to be in the vector space that completes $\vec{c_0}$ into a unimodular matrix (so here it is the second row of the inverse we computed). `b` combines those operators and is therefore a map that gives, for a given vector, the width of the symmetric conflict set in this direction. For example, `lexmin (fK.Opt)([0,1])` would gives `[1,0]` which represents $1 * N + 0$ in the Farkas space. We then compute the smallest width into `b_min`, and proceed to find a vector $\vec{c_2}$ that attains it. The smallest modulo here is $2N - 1$ (as `b_min` $= [2, -2] \mapsto 2N - 2$ and we need a modulo strictly larger).

5.2 Blur Filter

On the blur filter with an interleaved schedule [2] we find the optimal allocation at the condition that we optimize both for the Maximum distance $||.||_\infty$ and the Manhattan distance $||.||_1$, as explained in Section 4.2.1. Indeed, as shown in Figure 5, $(1,2)$ is equivalent to $(2,2)$ for $||.||_\infty$ and to $(0,3)$ for $||.||_1$. With this optimization, we get the desired allocation of $blurx[(y - 2x) \bmod (2N+1)]$.

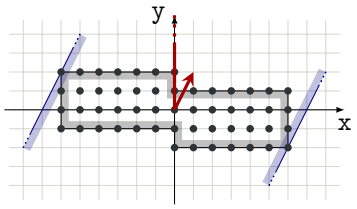


Figure 5: Blur filter w. interleaved schedule and conflict differences

$$\begin{aligned} N \to \{\, (x,y) \mid\ & (x \geq 1 \wedge x < N \wedge y \leq 1 \wedge y \geq 0) \\ & \vee (x \geq 0 \wedge x < N \wedge y \leq -1 \wedge y \geq -2 \wedge y > -N) \\ & \vee (x \leq 0 \wedge x > -N \wedge y \geq x \wedge y \leq 2 \wedge y \geq -1 \wedge y < N) \,\} \end{aligned}$$

5.3 Diamond Tiling

Another interesting example is the diamond tiling example [2] whose conflict set is shown in Figure 6. The technique of Bhaskaracharya et al., analyzed

in Section 6.2, finds an allocation of size $6B - 5$ with the mapping allocation $A_B[(t - 3i) \bmod (6B - 5)]$, where B is the tile size. Our algorithm finds the mapping $A_B[t \bmod 2, i \bmod (2B - 1)]$, a mapping aligned with the canonical basis that Lefebvre and Feautrier [14] thus also find (or equivalently Heuristic 1) if i is the first canonical dimension, and t the second. In our case, our combined heuristic does not fall in the trap: the first lattice vector found is $(0, 2)$, i.e., $\vec{a}_1 = (0, 1)$ and a modulo of 2, then the second vector is a mapping vector $(1, 0)$ with a modulo of $2B - 1$, so the canonical basis in the right order is found.

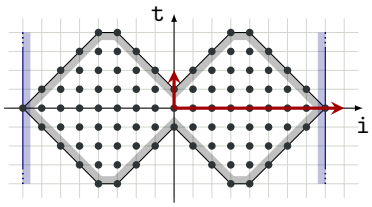


Figure 6: Diamond tiling, the conflict differences are

$$
\begin{aligned}
B \to \{\, (i,t) \mid\ & (i \le 0 \wedge t \le 1 - i \wedge t \ge -1 + i \wedge t \le 2B + i \wedge t \ge -2B - i) \\
& \vee (i \ge 0 \wedge t \le 1 + i \wedge t \ge -1 - i \wedge t \le 2B - i \wedge t \ge -2B + i) \\
& \vee (i \le B \wedge i \ge -B \wedge t \le 1 - i \wedge t \ge -1 - i \wedge t \le B \wedge t \ge -B) \\
& \vee (i \le B \wedge i \ge -B \wedge t \ge -1 + i \wedge t \le 1 + i \wedge t \le B \wedge t \ge -B)\, \}
\end{aligned}
$$

5.4 Other Examples

To compare our technique with the state of the art, we report in Table 1 the different mappings found on a set of examples previously introduced to evaluate the technique of Bhaskaracharya et al. [2]. We provide the mappings found by the successive modulo (for the canonical basis) of Lefebvre-Feautrier [14], by their technique, and by ours. The last column gives, in order of magnitude in terms of the parameters, the factor of improvement (the larger, the better), as compared to Lefebvre-Feautrier (thus it is 1 for the first row for each example). In all cases, we find the same or a better factor. Note that for `heat-2d-tile`, we took the tiling along the mapping $(t, i, j) \mapsto (t, t + i, t + i + j)$ but this may not be the same.

6 Related Work

There are a number of existing techniques for memory allocation in the polyhedral model, i.e., for programs on which code analysis and optimizations based on manipulations of polyhedra and linear programming techniques can be applied [2, 8, 14, 15].

This work extends the work by Darte et al. on lattice-based memory allocation [8, 9]. As developed in Section 4, we extend this framework both to handle (non-convex) unions of polyhedra and to improve the choice of projections, i.e., mapping functions. The work by Lefebvre and Feautrier [14] can be viewed as a special case of lattice-based allocation (more precisely in the form of Heuristic 1,

Table 1: Comparison between the "successive modulo" technique (LeFe), Bhaskara. & al. algorithm (BBA) and our approach (Lattice).

Example	Algo.	Mapping	Ratio
stencil-2d-5x5	LeFe	$A[t \bmod 2, x \bmod N, y \bmod N]$	1
	BBA	$A[(3x-2y) \bmod (5N-4)]$	$2N/5$
	Lattice	$A[y \bmod 2, x-y \bmod (2N-1)]$	$N/2$
blur-interleaved	LeFe	$A[y \bmod 3, x \bmod N]$	1
	BBA	$A[(2x-y) \bmod (2N+1)]$	3/2
	Lattice	$A[(y-2x) \bmod (2N+1)]$	3/2
blur-tiled	LeFe	$A[tx, ty, x \bmod B, y \bmod B]$	1
	BBA	$A[tx, ty, (y-2x) \bmod (3B-2)]$	$B/3$
	Lattice	$A[tx, ty, (y-2x) \bmod (3B-2)]$	$B/3$
LBM-D2Q9	LeFe	$A[t \bmod 2, i \bmod N, j \bmod N]$	1
	BBA	$A[(i-2t) \bmod (N+2), j \bmod N]$	2
	Lattice	$A[(i-t) \bmod (N+1), (j-t) \bmod (N+1)]$	2
LBM-D3Q19	LeFe	$A[t \bmod 2, i \bmod N, j \bmod N, k \bmod N]$	1
	BBA	$A[(i-2t) \bmod (N+2), j \bmod N, k \bmod N]$	2
	Lattice	$A[j \bmod N, (i-t) \bmod (N+1), (k-t) \bmod (N+1)]$	2
LBM-D3Q27	LeFe	$A[t \bmod 2, i \bmod N, j \bmod N, k \bmod N]$	1
	BBA	$A[(i-2t) \bmod (N+2), j \bmod N, k \bmod N]$	2
	Lattice	$A[(j-t) \bmod (N+1), (i-t) \bmod (N+1), (k-t) \bmod (N+1)]$	2
diamond-tile	LeFe	$A[t \bmod B, i \bmod (2B-1)]$	1
	BBA	$A[(t-3i) \bmod (6B-5)]$	$B/3$
	Lattice	$A[t \bmod 2, i \bmod (2B-1)]$	$B/2$
heat-2d-tiled	LeFe	$A_B[t \bmod B, i \bmod B, j \bmod B]$	1
	BBA	$A_B[(i-2t) \bmod (4B-3), (t-2j) \bmod (5B-4)]$	$B/20$
	Lattice	$A_B[j \bmod (2B-1), i-t \bmod (3B-2)]$	$B/6$

i.e., in the space of mappings), where only a subset of the mapping space is used (and not optimized), i.e., it works for a fixed basis.

The technique proposed by Quilleré and Rajopadhye [15], on the other hand, is more similar to Heuristic 2, in the sense that it explores the lattice space, characterizing legal projective allocations (possibly with modulo reuse), including projections along non-canonical directions. However, their primary objective is in minimizing the dimensionality of the resulting projection, and no algorithm is available to search among legal projections. Also, it can be used only with strong hypotheses, in particular for multi-dimensional schedules. Nevertheless, in this limited context, it does find constant reuse vectors when the conflict set is "flat", i.e., not fully-dimensional. Our optimized technique with successive extrusions generalizes this approach in a broader context.

Our technique has strong links with the search for multi-dimensional affine functions for scheduling, in particular for detecting tiling bands in nested loops (see Section 6.1). Our optimized version of Heuristic 1 uses similar linear programming techniques, as does the recent work of Bhaskaracharya et al. for intra-array reuse [2]. This latter work has the same objectives as ours, but it uses a different approach to tweak Heuristic 1 and try to avoid a N^2 mapping in the main example of Section 2. We explain in Section 6.2 why this approach still has some weaknesses, at least for intra-array reuse. The way we tweak Heuristic 1, with the help of Heuristic 2, reveals new connections with early work on universal occupancy (reuse) vectors as we explain in Section 6.3.

6.1 Link with Multi-Dimensional Scheduling and Tiling

It is interesting to note the strong link with the algorithm used in the Pluto compiler [4, 1] to build code transformations enabling tiling. In Pluto, operations in nested loops, each captured by a textual statement S and a loop counter (or iteration) vector $\vec{i}$, are reordered by defining affine mappings $\sigma_S(\vec{i})$ such that $\sigma_T(\vec{j}) - \sigma_S(\vec{j}) \geq 0$ whenever there is a dependence from $(S, \vec{i})$ to $(T, \vec{j})$. The objective function is to minimize the distance between these operations (this tends to improve data reuse), i.e., the maximum of all $\sigma_T(\vec{j}) - \sigma_S(\vec{i})$, as we do for the width computation in Heuristic 1.

However, there are three main differences compared to the scheduling algorithm. The first difference is that we assume a single σ, i.e., $\sigma_T = \sigma_S$, so that we can work with the difference $\vec{j} - \vec{i}$. But we could also apply Heuristic 1 to map different arrays in the same memory space, each with a possibly different mappings, as explored in SMO [3]. The second difference is that, at each step, we remove all pairs such that $\sigma(\vec{j}) - \sigma(\vec{i}) = \vec{0}$ (as in the search for maximal parallelism [12]) while, in Pluto, dependences need to be kept for defining other dimensions for tiling. The third difference is that there are no constraints such as $\sigma(\vec{j}) - \sigma(\vec{i}) \geq \vec{0}$ as conflicting differences can have any sign. In other words, the technique is similar but we will get a smaller width at each step (because of fewer constraints).

We can import all tricks used in Pluto [4] and Pluto+ [1], in particular for enforcing linear independence but, as our situation is simpler, the techniques we use are sufficiently cheap in our case. Nevertheless, this shows a strong link between multi-dimensional scheduling/tiling and array mapping: dependences are used for the first, pairs of conflicting elements for the latter, the main difference is that these pairs are not directed and can thus be "satisfied" by a mapping (i.e., $\sigma(\vec{i}) \neq \sigma(\vec{j})$) either as a positive or a negative value.

6.2 Link with Bhaskaracharya et al. Intra-Array Reuse

Heuristic 1 is the natural extension of Lefebvre-Feautrier successive modulo technique [14] with a greedy optimization of the basis. As we mention in the above, it has some similarities with multi-dimensional scheduling and tiling.

While optimizing the basis is usually not needed when $\mathcal{K}$ is convex (at least in order of magnitude) [9], this is not the case anymore when $\mathcal{K}$ is a union of polyhedra, and different bases can give different mapping sizes in order of magnitude. In particular, Heuristic 1 may suffer from the fact that, at each step, the optimization is equivalent to considering the convex hull of the intersection of $\mathcal{K}$ with the orthogonal of the previously-built vectors. Also, minimizing the width to get a small modulo may induce a bad choice for the next steps. This is our motivation, shared with Bhaskaracharya et al. [2], to find a mechanism to force Heuristic 1 to *not* choose an hyperplane with smallest width, i.e., smallest modulo. Our result is Heuristic 2, and we now highlight the differences with respect to the work by Bhaskaracharya et al. [2] in the following.

Their approach is formulated with a relation describing pairs of conflicting elements $(\vec{i}, \vec{j})$ instead of conflicting differences $\vec{i} - \vec{j}$ as we do. For intra-array reuse, only one σ is searched, so this is equivalent. We thus rather explain their technique with conflicting differences to simplify the discussions and visualizations.

As we do, they assume that the conflict set $\mathcal{K}$ is given by the integer points in a union of polyhedra $\mathcal{K} = \cup_{1 \leq j \leq r} \mathcal{P}_j$, but $\mathcal{K}$ is only half of the conflict set to exclude $\vec{0}$ (which can never be separated by any hyperplane) and to make it asymmetric[1]. Then, instead of searching for a hyperplane with minimal width, they search for a hyperplane that "separates" as many polyhedra $\mathcal{P}_j$ of the union as possible, thus fully on one side of the hyperplane, i.e., with either $\sigma(\vec{x}) \geq 1$ for all $\vec{x} \in \mathcal{P}_j$ or $\sigma(\vec{x}) \leq -1$ for all $\vec{x} \in \mathcal{P}_j$. Among such solutions, the one with smallest width is chosen. If such a hyperplane is found for all j, it forms a valid one-dimensional allocation. When such a hyperplane does not exist, several hyperplanes are successively applied, focusing on the intersection of $\mathcal{K}$ with the orthogonal to the first hyperplane (as in Heuristic 1) until all the conflicts are resolved with the separating hyperplanes.

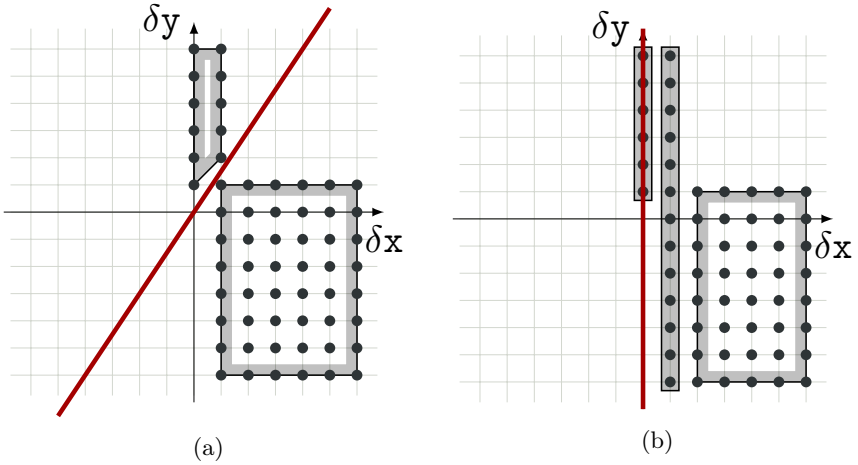
Although their algorithm may work well in many cases, careful look reveals certain situations where their approach misses good allocations, sometimes even using higher-dimensional array than necessary. The two main problems is that, as for Heuristic 1, minimizing the moduli in such a greedy fashion is not always good, and, more importantly, the resulting allocation is dependent on the way the conflict set is decomposed into a union of polyhedra, due to their primary objective defined at the granularity of the constituting polyhedra. This makes the approach quite unstable.

Figure 7 illustrates the problem with their heuristic, and how the decomposition may influence the allocation. In contrast, our approach finds better allocations for these examples by using a different objective function that does not rely on the way the conflict set is represented as a union of polyhedra. The key point is that, despite the link between multi-dimensional scheduling and affine mapping explained in Section 6.1, there is one fundamental difference. In scheduling, due to the constraint $\sigma(\vec{i}) - \sigma(\vec{j}) \geq 0$ all dependences are made nonnegative (weakly separating hyperplane) and some are made positive $\sigma(\vec{i}) - \sigma(\vec{j}) \geq 1$ (separating hyperplane). If a first schedule σ_i separates $\mathcal{P}_i$ and a second schedule σ_j separates $\mathcal{P}_j$, then $\sigma_i + \sigma_j$ is a also a schedule that separates both $\mathcal{P}_i$ and $\mathcal{P}_j$. This is the reason why a greedy separating approach [12] is optimal for detecting maximal parallelism. Here, this is not true because separation can be ≥ 1 or ≤ -1, which breaks the analogy.

6.3 Link with Universal Occupancy Vectors

An occupancy vector is a vector that points to another iteration that can safely reuse the memory location. Universal occupancy vectors (UOVs) are those that are legal for any legal schedule of a program [17]. For UOVs, the formulation is as follows. An iteration point $\vec{v}$ may overwrite a value produced at another iteration $\vec{u}$ if $\vec{v}$ (transitively) depends on all uses of $\vec{u}$. Since all uses of $\vec{u}$ must be executed for $\vec{v}$ to execute without violating dependences, the allocation is valid for all possible schedules. Constraints for expressing UOVs are exactly the complement of constraints for expressing conflicts, in the context of schedule-independent mappings. The theory of UOV is however rather limited, as only codes with uniform dependences and constant UOVs are captured. Otherwise, the set of legal UOV is hard to define. Thies et al. [18] designed an algorithm

[1] How this is performed is not given. We assume that it is done by taking the intersection with the strict lexicographic order.



(a) (b)

Figure 7: How the technique by Bhaskaracharya et al. [2] encounters difficulties with the example from Figure 1. Figures 7a and 7b are two possible decompositions of the asymmetric version of the conflict set. In Figure 7a, a hyperplane exists that does not intersect with the entire conflict polyhedra, satisfying their primary objective function. This leads to N extra storage as we illustrated in Section 2. Figure 7b is another possible decomposition where no hyperplane can partition the entire conflict set. In this case, their primary objective to maximize the number of polyhedra (in the disjoint union) fully on one side of the hyperplane, in combination with the secondary objective to minimize the width of a dimension in the final array, gives the vertical line. This produces N^2 storage in the end, which is worse by one full dimension that what we achieve.

to handle affine dependences, but they had to restrict to the set of legal affine one-dimensional schedules. In this case, the set of all constant valid constant occupancy vectors (AUOV, for affine UOV) can be defined.

Although allocations legal for all possible schedules may seem too conservative, UOVs can still give efficient allocations for many programs. In fact, it is no coincidence that UOV-based allocations (and in particular QUOV-based allocations [22], designed for tiling) find good allocations for live-out values of tiled stencil programs [2]. If we look at an allocation for a tile, the live-out values are those that must be preserved at the end of the execution of a tile. These values should never be overwritten since there are other uses outside of a tile. Thus, even if the tile itself as a specific schedule, live-out variables are captured well with schedule-independent mappings, e.g., with UOV-based allocations. These exploit possible reuse within the tile while keeping the live-out values.

Partially due to the restriction of UOVs that only use one projection (hence using $d-1$ dimensional array for d dimensional iteration space), the primary heuristic used in the search for a UOV is the length of the UOV [17]. The gcd of the UOV is directly connected to the memory consumption through modulo factors along the projection. Increases in the UOV length that do not influence the gcd often increase the memory usage by making the projection to be more steeply angled. For example, projecting a $N \times N$ domain along the vector $(1,0)^T$ gives a line of length N, while a diagonal projection along $(1,1)^T$ gives $2N-1$. If such a projection is along some boundary of the domain, it may decrease the memory usage, but the length of the UOV is a good approximation otherwise. This shares the same idea with the heuristic used in our approach.

7 Conclusion and Future Work

In this paper, we have extended the lattice-based memory allocation in two important directions: unions of polyhedra, and better objective functions to find the basis vectors. The key insight is taken from the two dual approaches in the original lattice-based method, the first that works with hyperplanes on one side and the second with reuse vectors on the other side. We have shown that reuse vector based selection of the basis of the mappings finds more compact mappings with different examples.

Several research directions directly follow the work presented in this paper.

- One immediate future work is to explore how the Heuristic 2 can be made fully parametric.
- Reasoning about the optimality of the resulting mapping. Can we guarantee that we obtain the mapping that is dimension-wise optimal?
- While Heuristic 1 can be generalized to inter-array optimizations where different statements have different mappings, as done by Bhaskaracharya for their intra-array mapping [3], it is unclear how to extend our lattice-based method with reuse vectors.
- Although we have focused on compactness of the allocation in this paper, it is not always the best for performance. All prior work on memory allocation in the polyhedral literature have never taken locality into account. An important direction to explore is how we can explore memory layout transformations for improved performance, e.g., through more cache-friendly layouts, and in particular cases where introducing redundant storage leads to better overall performance.

In this paper, we have separated the construction of conflict sets as an orthogonal component, since our method works for conflict sets that comes from many different inputs. In addition to the classical conflict sets computed based on multi-dimensional affine schedules, we may take those from other specifications, including explicitly-parallel specifications (such as OpenMP, X10, OpenStream, and so on), or when we consider kernel offloading with overlapped computation and I/O. Memory reuse analysis is important in many different scenarios, and now we have provided the theory that seamlessly generalizes across different cases.

Acknowledgements

We thank Paul Feautrier for fruitful discussions, in particular for the Fourier-Motzkin trick to compute the star-shaped extension of $\mathcal{K}$.

References

[1] Aravind Acharya and Uday Bondhugula. PLUTO+: Near-complete modeling of affine transformations for parallelism and locality. In *Proceedings of the 20th ACM/SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP'15, pages 54–64, San Francisco, February 2015.

[2] Somashekaracharya G. Bhaskaracharya, Uday Bondhugula, and Albert Cohen. Automatic intra-array storage optimization. Technical Report IISc-CSA-TR-2014-3, Indian Institute of Science, November 2014.

[3] Someshekaracharya Bhaskaracharya, Uday Bondhugula, and Albert Cohen. SMO: An integrated approach to intra-array and inter-array storage optimization. In *Proceedings of the 43rd Annual Symposium on Principles of Programming Languages*, POPL'16, St Petersburg, Florida, January 2016.

[4] Uday Bondhugula, Muthu Manikandan Baskaran, Sriram Krishnamoorthy, J. Ramanujam, Atanas Rountev, and P. Sadayappan. Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model. In *Proceedings of the 17th International Conference on Compiler Construction*, CC'08, pages 132–146, 2008.

[5] Alessandro Cilardo and Luca Gallo. Improving multibank memory access parallelism with lattice-based partitioning. *ACM Transactions on Architecture and Code Optimizations (TACO)*, 11(4):45:1–45:25, 2014.

[6] Alain Darte. Optimal parallelism detection in nested loops. In David Padua, editor, *Encyclopedia of Parallel Programming*. Springer, 2011.

[7] Alain Darte, Leonid Khachiyan, and Yves Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1(2):73–81, 1991.

[8] Alain Darte, Rob Schreiber, and Gilles Villard. Lattice-based memory allocation. In *Proceedings of the 6th ACM International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, CASES'03, pages 298–308, San Jose, CA, USA, October 2003.

[9] Alain Darte, Robert Schreiber, and Gilles Villard. Lattice-based memory allocation. *IEEE Transactions on Computers*, 54(10):1242–1257, October 2005.

[10] Eddy De Greef, Francky Catthoor, and Hugo De Man. Memory size reduction through storage order optimization for embedded parallel multimedia applications. *Parallel Computing*, 23:1811–1837, 1997.

[11] Paul Feautrier. Some efficient solutions to the affine scheduling problem, I, one-dimensional time. *International Journal of Parallel Programming*, 21(5):313–348, 1992.

[12] Paul Feautrier. Some efficient solutions to the affine scheduling problem, part II: Multi-dimensional time. *International Journal of Parallel Programming*, 21(6):389–420, December 1992.

[13] Paul Feautrier. The power of polynomials. In *Proceedings of the 5th International Workshop on Polyhedral Compilation Techniques*, IMPACT '15, Amsterdam, The Netherlands, January 2015.

[14] Vincent Lefebvre and Paul Feautrier. Automatic storage management for parallel programs. *Parallel Computing*, 24:649–671, 1998.

[15] Fabien Quilleré and Sanjay Rajopadhye. Optimizing memory usage in the polyhedral model. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22(5):773–815, 2000.

[16] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, 1986.

[17] Michelle Mills Strout, Larry Carter, Jeanne Ferrante, and Beth Simon. Schedule-independent storage mapping for loops. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '98, pages 24–33, 1998.

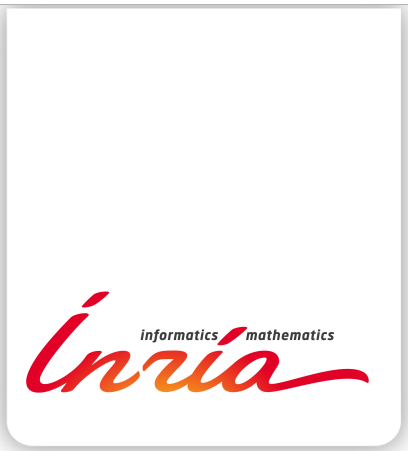
[18] William Thies, Frédéric Vivien, and Saman P. Amarasinghe. A step towards unifying schedule and storage optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(6), 2007.

[19] Sven Verdoolaege. isl: An integer set library for the polyhedral model. *Mathematical Software–ICMS 2010*, pages 299–302, 2010. Corresponding library: `isl.gforge.inria.fr`.

[20] Sven Verdoolaege. Counting affine calculator and applications. In *Proceedings of the 1st International Workshop on Polyhedral Compilation Techniques*, IMPACT '11, Chamonix, France, April 2011.

[21] Jingling Xue. *Loop Tiling for Parallelism*. Kluwer Academic Publishers, 2000.

[22] Tomofumi Yuki and Sanjay Rajopadhye. Memory allocations for tiled uniform dependence programs. In *Proceedings of the 3rd International Workshop on Polyhedral Compilation Techniques*, IMPACT '13, pages 13–22, January 2013.



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