A reindexing based approach towards mapping of DAG with affine schedules onto parallel embedded systems

Clementin Tayou Djamegni, Patrice Quinton, Sanjay Rajopadhye, Tanguy Risset, Maurice Tchuente

A reindexing based approach towards mapping of DAG with affine schedules onto parallel embedded systems

Clémentin Tayou Djamegni[a,*] Patrice Quinton[b]
Sanjay Rajopadhye[c] Tanguy Risset[d]
Maurice Tchuenté[e,f]

[a] *Faculté des Sciences, B.P 069 Dschang, Cameroun*
[b] *IRISA, Campus de Beaulieu, 35042 Rennes-Cedex, France*
[c] *Colorado State University, Fort Collins CO, USA 80523-1873*
[d] *LIP, ENS-Lyon, 69364 Lyon Cedex 07, France*
[e] *Faculté des Sciences, B.P 812 Yaoundé, Cameroun*
[f] *IRD, UR GEODES, 32 Avenue Henri Varagnat, 93143 Bondy Cedex, France*

Abstract

We address the problem of optimally mapping uniform DAGs to systolic arrays, given an affine timing function. We introduce an automatic allocation method based on a preprocessing by reindexing that transforms the initial DAG into a new one that enables the well known projection method to minimize the number of processors along a number of directions. We demonstrate its superiority to other methods, and establish the space-optimality of the proposed method. We also show an upper bound on the number of processors that corresponds to the best space complexity that both the projection method and the so-called grouping method can give for the initial DAG. We also describe how the new allocation method can be implemented in tools.

*Corresponding author: dtayou@{cril.univ-artois.fr, yahoo.com}

Keywords: parallel embedded systems, timing function, timing surface, allocation function, space-optimality, space-time complexity, reindexing, dependence graph, systolic arrays.

1 Introduction

Embedded systems are information processing devices that are integral parts of complex applications: consumers buy them for the functionality they provide, not because there is a computer inside. Embedded systems are becoming more and more important from both the economic and technical side. They are tailored to do a particular task that can be squeezed into an extremely compact space, and are often specialized to a given task. The proliferation of computational devices have pushed for specialization to meet constraints on speed, power, energy, and area. For example, a mobile phone contains a specialized embedded architecture consisting of several different processors such as a DSP for wireless channel processing, a micro-controller that is responsible for the user interface and communication protocol, as well as several custom hardware components e.g., in the receiver/transmitter. The performance of this heterogeneous architecture exceeds that of any existing PC at lower power consumption, size, weight, and production cost. It is specialization that makes the mobile phone feasible and economical. Embedded systems are also ubiquitous in automotive electronics, robotics, aerospace applications and video game platforms, to name just a few prominent examples.

Many candidates for embedded systems in signal processing and multimedia applications are compute-bound. In order to meet higher performance computing speeds, an important option is to use highly parallel architectures (VLSI processor arrays, FPGA, etc.) This is shown by the relative maturity achieved by academic research systems such as MMAlpha [22], Compaan [24], VASS [48], PARO [43, 1] as well as industrial tools such as PICO-N system [38] developed at HP Labs, and now commercialized as PICO Express by Synfora. In these systems, the target (virtual) architecture is usually a systolic array, i.e., a locally interconnected grid of elementary processors called Processing Elements (PEs) or cells. In addition to full custom ASICs, FPGAs and other reconfigurable platforms also serve as physical target architectures for such PE arrays.

In designing systolic arrays, one usually first seeks the fastest schedule,

and then chooses an allocation function that maps the computation to the PEs. The problem of minimizing the number of PEs, for a given optimal schedule, has been widely studied in the literature. The techniques proposed have been either linear allocation functions [29, 31, 33, 40, 47], called projection methods, as well as non-linear allocation functions, namely *grouping* [5, 7, 9], *instruction shifts* [8], *piling* [2, 7] and *partition* [37, 42]. Of these, the last three are special cases of *piecewise linear* allocation functions, where different linear allocations are applied to different regions of the domain. It has been conjectured that the optimal allocation function is *always* piecewise linear, but this has never been proved.

The projection method, the grouping method and the instruction shift method can, and have been implemented in automatic synthesis tools, but the others have not, primarily because there is no clear way of describing the "pieces." Moreover, all the existing methods require human insight and are therefore not automatic. None of the methods yields provably space-optimal arrays in general [12, 10]. Nevertheless, they have all been used to obtain space-optimal arrays for many specific problem instances. Of these, the piling method usually produces arrays with spiral interconnections, leading to somewhat poorer locality. For many problems, the instruction shift method and the partition method [2, 3, 4, 6, 8, 10] have proved to be superior to the piling method, yielding arrays with fewer PEs, although the power of these methods have not been formally compared previously.

In this paper, we resolve the standing open problem of deriving space-optimal arrays for a given uniform dependence DAG with a given affine schedule. Our method is constructive and consists of two steps applied recursively, dimension by dimension. The steps are: (i) a sequence of re-indexations applied to specific pieces of the domain (our method describes how to determine the pieces); and (ii) projections along canonical directions. Because of this, we are able to describe how the method may be automated and incorporated in a tool such as MMAlpha [22]. The proof of space optimality de facto proves the superiority of this method over the previous ad hoc techniques. We also systematically compare our method with the projection and grouping methods. The idea behind this allocation method was originally introduced, for specific problem instances, by Tayou Djamegni [11, 12, 19].

The rest of the paper is organized in the following way. Section 2 illustrates our method (without the dimension by dimension recursion) on a simple sorting example. In Section 3, we introduce the notation, summarize how parallel algorithms are synthesized from recurrence equations, and

present previous allocation methods. Section 4 describes our method, and in Section 5 we illustrate it on a number of examples. In Section 6, we compare the new allocation method with other methods. Section 7 describes how to resolve an important problem if one is to implement the method in tools like MMAlpha [22], and we conclude in Section 8.

2 Illustrative Example

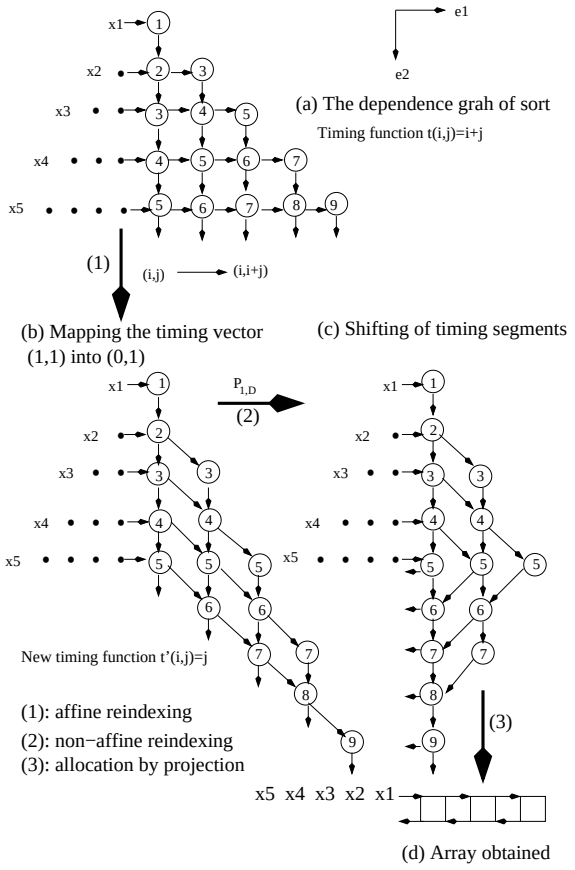

Figure 1: Illustration of the allocation method on the bubble sort

We first illustrate the main idea of the reindexing scheme through a simple sorting example. It has been deliberately chosen so that the recursion is

not needed (the problem is two-dimensional). Figure 1.a shows the bubble sort algorithm and its triangular shaped dependence graph, together with the optimal timing function, $t(i,j) = i + j$. The maximum number of active points is $\frac{n}{2}$, so an optimal allocation function should have $\frac{n}{2}$ PEs. All projections (including the standard ones—orthogonal along the axes, or along the diagonal) yield arrays with at least n PEs, and are suboptimal. However, note that the diagonal projection yields an array where every PE is active only on alternate clock cycles, and two adjacent PEs can be grouped into a single PE, with no slowdown. Thus, the projection method combined with grouping yields an optimal array. However, we do not know how to discover this projection automatically.

Another optimal allocation is obtained if we partition the triangle into a family of L-shaped lines and allocate all nodes along each such line to a single PE: the left and bottom boundary of the triangle would be computed on the leftmost PE, and so on. The corners of these L-shaped lines all lie on the line, $i + j = n$, and this allocation corresponds to the piecewise linear allocation function

$$a(i,j) = \begin{cases} i + j \leq n & : \quad i \\ i + j > n & : \quad n - j \end{cases}$$

This is the essence of the partition method, and the key difficulty is that discovering the $i + j = n$ line to cut the domain requires designer insight.

For our example, the instruction shift method and the piling method yield another array which is also optimal (both of them follow very similar reasoning for this example and produce the same array). To explain these methods, it is useful to first produce a suboptimal array, say by the canonical projection (as shown in Figure 1.b). Every vertical line in this "skewed" triangle is the temporal activity of a PE. We observe that PEs have different starting and ending times. In fact, the i-th PE starts at $2i - 1$ and finishes at $n + i - 1$. This means that the i'-th PE in the "right half of the array" (i.e., one whose coordinates are $\frac{n}{2} + i'$) starts just as the i'-th PE (in the left half) stops. Thus they can both be executed by a single physical PE.

The problem is that all these approaches need some human intuition. Our method finds the optimal allocation as follows. We first choose some arbitrary allocation, such as the one in Figure 1.b (the method is insensitive to this choice). Each row in this triangle represents the set of simultaneous activity of some subset of PEs. We translate each such line (i.e., reindex the domain) so that it is flush with the vertical axis. This yields the graph

shown in Figure 1.c, and corresponds to our optimal array (Figure 1.d). The partition method also gives us the same array.

The remainder of this paper formally proves that this approach is guaranteed to produce arrays with the minimal number of PEs, and that the method can be generalized to higher dimensions.

3 Background

Algorithms that are good candidates for systolic implementations are characterized by regular and repetitive operations on large sets of data. They are usually written in the form of nested loops [30] and more generally in the form of a System of Recurrence Equations (SREs) [23, 36]. These formalisms and the problem they involve were first developed by Lamport [25] and by Karp et al. [23].

The standard methodology for systolic array synthesis developed over the past 25 years [29, 31, 33, 34, 35, 37, 40] proceeds in four steps.

- Specify the problem as an SRE or as a loop program.
- Uniformize/localize the dependences in the specification [33, 34, 35, 45, 46]. This leads to a so called System of Uniform Recurrence Equations (SURE) which has a *uniform* dependence graph: a Directed Acyclic Graph (DAG), $G = (D, U)$ where D is the set of tasks and U the set of edges. D is the set of integer points in a polyhedral region of $\mathcal{Z}^n$, and U is a set of constant n-vectors.
- Define a *timing function* (or *schedule*) $t : D \rightarrow \mathcal{N}$ that gives the computation date $t(v)$ of each task v of D, assuming that all tasks are of unit delay.
- Define an *allocation function* $D \rightarrow \mathcal{Z}^{n-1}$ that assigns the tasks to PE coordinates so as to avoid conflict, i.e., no two tasks with the same execution date are assigned to the same PE.

For the purpose of this paper, we assume that the first three steps have been performed, and we seek optimal allocation function(s).

Definition 1 *A timing function is said to be* optimal with respect to a dependence graph, G *iff it minimizes the number of time steps needed to execute*

all the tasks of G, i.e., the total execution time is equal to the length of the longest path of G. We restrict ourselves to affine *timing functions, i.e., functions of the form* $t(z) = \lambda^t z + \alpha$, *where* $z \in Z^n$, $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)^t$ *and* λ_1, λ_2, $\ldots$, λ_n, α *are integral constants. The vector* λ *is called the timing vector or scheduling vector.*

The schedule vector λ is orthogonal to $(n-1)$-dimensional hyperplanes of Z^n whose points have the same execution instant. These hyperplanes are called the *timing surfaces*. The maximum number of tasks on any timing surface defines the potential parallelism. To satisfy the causal dependencies, any affine timing function must satisfy the following dependence constraint.

Definition 2 *A vector* $v = (v_1, v_2, \ldots, v_n)^t$ *is said to be unimodular (or primitive) iff its components are relatively prime, i.e.,* $gcd(v_1, v_2, \ldots, v_n) = 1$. *An integral square matrix U is said to be unimodular iff* $|det(U)| = 1$. *The affine timing function* $t = [\lambda, \alpha]$ *is normalized iff its linear part* λ *is unimodular.*

Proposition 1 *An affine timing function specified by* $[\lambda, \alpha]$ *is valid if it satisfies* $\lambda^t d_z > 0$ *for any dependence vector* d_z *associated with the problem.*

A linear allocation is realized by projecting the iteration domain along a direction called *allocation direction* or *projection direction*. It is defined by a constant n-dimensional vector ξ.

Proposition 2 *Let* $[\lambda, \alpha]$ *be a valid affine timing function. A constant n-dimensional vector* ξ *defines a valid allocation direction if and only if* $\lambda^t \xi \neq 0$.

Definition 3 *Given a timing function, we call* potential parallelism *the maximum number of tasks having the same computation date. An allocation function is said to be* optimal with respect to a dependence graph G *and* a timing function *iff it minimizes the number of PEs, i.e., the PE count is equal to the potential parallelism of the timing function.*

Definition 4 *The minimum number of PEs over all possible allocation functions described by projections, defines the* potential parallelism by projection.

Throughout this paper we assume without loss of generality that the affine timing function $t = [\lambda, \alpha]$ is normalized and that vector ξ is unimodular. We now provide a number of definitions that allow us to study the activities of PEs and arrays.

Definition 5 *A working date of a PE is any date at which that PE executes a task of the iteration domain D. The working interval of a PE is the interval defined by the first and last working dates of that PE. An activation date of an array is a date at which at least one PE executes a task of D. The working rate of a PE is the ratio of the number of working dates of the PE and its working interval. The pipelining rate of an array is defined as the average of the working rates of its PEs.*

The working rate of a PE is 100% if it is active at all dates within its working interval. The pipelining rate of an array is 100% if the working rate of all its PEs is 100%. This is not often achieved. For example, the pipelining rates of both optimal sorting arrays described in the previous section, is 50%.

3.1 Minimizing the number of PEs

We now summarize previous work related to the minimization of the number of PEs in regular arrays.

1. *Projection method* [29, 31, 33, 40, 47]. This method corresponds to linear allocations. As stated earlier it is realized by projecting the dependence graph G along a direction ξ. All the tasks belonging to a same line of direction ξ are assigned to a same PE. The main drawback of this allocation technique is the low PE utilization occurring in the resulting array [8]. Wong and Delosme [47], Shang and Fortes [40], Ganapathy and Wah [21] use integer programming to get the best linear allocation possible. However, such a linear allocation may not correspond to an optimal allocation.

   For instance, the minimal number of PEs obtained by projecting the dependence graphs of the *Cholesky Factorization* (henceforth called CF) [8], *Triangular Matrix Inversion* (henceforth TMI) [27] and *Optimal String Parenthesization by Dynamic Programming* (DP) [26] is $N^2/2+\Theta(N)$ where N is the problem size. For square *Matrix Multiplication* (MM) the minimal number of PEs is N^2. None of these arrays is space optimal with respect to the earliest or the latest optimal timing functions. The potential parallelism of CF and MM are, respectively, $N^2/8 + \Theta(N)$ [8] and $3N^2/4 + \Theta(N)$ [6] for any optimal timing function. The potential parallelism of the TMI and DP are, respectively, $N^2/4 + \Theta(N)$ [37, 17] and $N^2/6 + \Theta(N)$ [26] for the latest optimal

timing functions and, $N^2/6+\Theta(N)$ [27] and $N^2/14+\Theta(N)$ [26] for the earliest optimal timing functions.

2. *Grouping (or clustering) method* [5, 7, 9]. This approach seeks to reduce the number of PEs of an initial array obtained by projection stemming from the analysis of PEs' activity. It consists in grouping $|\lambda^t\xi|$ neighboring PEs with distinct working dates. Typically, the PEs that are clustered are along a direction specified by a vector, γ. This approach permits to reduce the number of PEs by a factor of $|\lambda^t\xi|$. In addition, the resulting solution is a locally connected array that has a 100% pipelining rate. In The grouping method cannot guarantee that the design is space-optimal, because a 100% pipelining rate in the array does not necessarily imply space-optimality. Another drawback of the grouping method is that it applies only if $|\lambda^t\xi| \geq 2$.

   For the CF, TMI and DP the grouping method reduces the number of PEs by a factor of 2, i.e., from $N^2/2+\Theta(N)$ to $N^2/4+\Theta(N)$. For TMI this is optimal according to the latest optimal timing function [37, 17]. For MM it is easy to prove that the grouping method cannot do better than N^2 PEs.

3. *Instruction shift method* [8]. As in the grouping method, the starting point is an array obtained from the projection method. Then, the initial array is partitioned into a number of PEs segments parallel to a direction called *partition direction*. The tasks of each segment are reallocated so as to minimize the number of PEs on each segment. This amounts to minimizing the number of PEs on all intersection of the dependence graph with a plane generated by both the projection direction and the partition direction. This approach does not guarantee space-optimality. For the CF this technique reduces the number of PEs from $N^2/2+\Theta(N)$ to $N^2/6+\Theta(N)$ [8]. Although this represents a significant improvement, this design is not space optimal as the potential parallelism of the CF is $N^2/8+\Theta(N)$ [8].

4. *Piling method* [2, 7]. First, from a given dependence graph and affine schedule, a set M of tasks is found such that all tasks in the set are scheduled to be executed at the same time and the set size $|M|$ is maximal. Second, an allocation method is applied to assign the tasks of the dependence graph to PEs. Any PE that has not been assigned

to execute a task of M is piled to a PE that executes a task of M and has disjoint working intervals. However, piling PEs results in long communications such as spiral links, and increases irregularity for the resulting arrays. In this paper, we seek to avoid this. Another drawback of this method is that it is too general: as stated it can be applied to any task graph; exploiting it for systolic array synthesis requires human insight.

5. *Partition method* [37, 42]. The starting point is to partition the dependence graph G into a number of sub-graphs of lower dimension. Then, the tasks of each sub-graph are allocated to PEs, in an ad hoc manner, so as to minimize the number of PEs on each sub-graph. When the sub-graphs are of dimension two, this corresponds to the instruction shift method. Thus, the partition method can be seen as a generalization of the instruction shift method.

   In most of the cases where it has been applied in the literature [3, 4, 10, 16, 18, 11, 13, 26, 27, 37, 42] the sub-graphs correspond to two-dimensional domains that are obtained by considering the intersections of the dependence graph with a set of parallel planes. This allocation heuristic leads to an array of $N^2/8 + \theta(N)$ PEs for the DP [26]. In [18] the number of PEs is reduced to $N^2/10+\theta(N)$ by merging nodes of the dependence graph associated with the DP before applying the partition method. All these solutions are not space-optimal as they are based on the earliest optimal timing function whose potential parallelism is $N^2/14 + \Theta(N)$ [26]. The partition method is also used in [16] to derive a space-optimal array for the *Algebraic Path Problem* (APP) and in [3, 4] by Bermond et al. to derive various arrays for the *Gaussian Elimination* (GE). However these GE arrays are not space-optimal. A Space-time optimal array for the GE is proposed in [2]. For CF the partition method leads to an array of $N^2/6+\Theta(N)$ PEs [11]. For TMI, the partition method leads to space-time optimal arrays for both the earliest and latest optimal timing functions [37, 27]. The partition method also provides space-optimal arrays for MM. The main drawback of the partition method comes from the fact that the tasks of each sub-graph are not allocated in a systematic manner. Because of this the partition method is not suitable for tools that automatically generate application specific VLSI PEs.

We see that only the projection method, the grouping method and the instruction shift method are suitable for tools that (semi) automatically generate application specific VLSI PEs. Moreover, the allocation methods presented above do not guarantee space-optimality. The weakest allocation method, in terms of reducing the number of PEs, seems to be the projection method, while the most promising ones are the grouping method, the instruction shift method and the partition method. The partition method is a generalization of the instruction shift method. According to results related to CF, TMI, DP, GE and MM the most powerful method seems to be the partition method. However, this has not yet been confirmed by a systematic comparison. In others words, we are not aware of any procedure based on the partition method that systematically leads to results that are better than (or as good as) the best result that the grouping method (respectively, the projection method) can give.

This brief survey of previous allocation methods shows that the problem of minimizing the number of PEs occurring in regular arrays remains open despite considerable research.

3.2 Notation related to the new allocation method

The allocation strategy introduced in section 4 is based on reindexing transformations. We now show how these transformations act on a uniform dependence graph associated with an SURE. Let S be an $SURE$ defined over a domain $D \subset Z^n$ and associated with a dependence graph $G = (D, U)$. Assume that S admits a valid affine timing function $t = [\lambda, \alpha]$.

Definition 6 *A function $g : D \rightarrow g(D)$ is a valid reindexing transformation for S iff it is a bijection.*

The reindexing function g transforms system S into a new one S' that is is obtained by replacing in system S each index point z with its image by g, i.e., $g(z)$. The dependence graph G becomes $G' = (D', U')$ where $D' = g(D)$ and $U' = \{[g(z_1) \rightarrow g(z_2)] \mid [z_1 \rightarrow z_2] \in U, \text{ with } z_1 \in D \text{ and } z_2 \in D\}$. The equivalent timing function t' corresponding to the reindexing g assigns to $z' \in D'$ the computation date of its pre-image by g, i.e., $g^{-1}(z')$. This means that $t' = t \circ g^{-1}$.

Proposition 3 *When the reindexing g corresponds to an unimodular transformation, i.e., $g(z) = Uz + z_0$ where U is a constant unimodular matrix and*

z_0 is a constant, the schedule $[\lambda, \alpha]]$ is transformed to $[U^{-t}\lambda, \alpha - \lambda^t U^{-1} z_0]$. Furthermore, a reindexing maintains the potential parallelism under the new schedule [12].

Definition 7 *A segment $seg(z_1, z_2)$, $z_1 \neq z_2$, with extremities z_1 and z_2 is the set $\{z \in Z^n \mid z = \alpha_1 z_1 + \alpha_2 z_2,\ \alpha_1, \alpha_2 \geq 0, \alpha_1 + \alpha_2 = 1\}$.*

A domain D is said to be convex along an n-dimensional primitive vector s if its intersection with any segment parallel to s is either empty or a single segment. D is convex if it is convex along all directions.

If D is convex along s, the frontier $\mathcal{F}(D, s)$ of D along s*, is the subset of D defined by $\mathcal{F}(D, s) = \{z \in D \mid z + s \notin D\}$.*

Definition 8 *Given two linearly independent vectors v_1 and v_2, a $\langle v_1, v_2 \rangle$-slice of D is any intersection of D with a 2-hyperplane parallel to vectors v_1 and v_2, called supporting 2-hyperplane.*

Assume that D is convex along s. For some $z \in D$, say that $z_{s,D}$ (the "shadow" of z along s on the frontier of D) is the unique index point of $\mathcal{F}(D, s)$ that belongs to the line defined by vector s and point z. Similarly, let z_s denote the projection of point $z \in Z^n$ along s on the $(n-1)$-hyperplane orthogonal to s and containing point $(0,\ 0,\ 0, \ldots,\ 0)^T \in Z^n$.

Consider the array obtained by projecting domain D along ξ. $NPE(\xi, D)$ denotes the number of PEs in this array. The number of PEs obtained by applying the clustering method along γ is denoted by $NPE(\xi, D, \gamma)$, and the number of PEs obtained by applying the instruction shift method along γ is denoted by $NB(\xi, D, \gamma)$.

4 The new allocation method

We are now ready to present the new allocation method. As stated earlier, it is based on a preprocessing by reindexing that transforms the original domain into a new one that is more suitable for the application of the projection method. Given that the projection method assumes that points having the same computation date belong to the same $(n-1)$-hyperplane, we will consider only reindexing functions that preserve this property. Throughout this section, S denotes an $SURE$ over a bounded domain $D \subseteq \mathcal{Z}^n$, with a valid timing function $t = [\lambda, \alpha]$, where λ is normalized (otherwise, t would

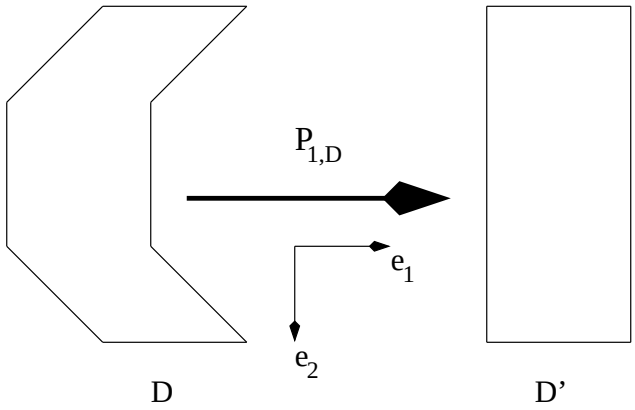

Figure 2: Illustration of the reindexing $P_{1,D}$. The timing vector is e_2

not be optimal). Let $\langle e_1, e_2, e_3, \ \ldots, e_n \rangle$ be the canonical basis of Z^n. As in all related work we assume that the timing surfaces are convex.

In order to simplify the presentation, we first develop the method on two-dimensional iteration domains. Then, results obtained from this simple case are generalized to n-dimensional domains.

4.1 Two-dimensional Domains

Here the timing surfaces are line segments. Depending on the direction of the timing vector λ, we consider two cases.

Case 1: *λ is collinear to a canonical basis vector e_l.*

We present here the reindexing that enables the projection method to provide an optimal allocation. Let e_h be the other canonical basis and choose s to be either e_h or $-e_h$. The key idea is to shift the timing segments along s so as to move the points of $\mathcal{F}(D, s)$ to the e_l axis. To do so, we use the reindexing function $P_{h,D}$ defined as follows:

$$\begin{aligned} P_{h,D}(z) &= \begin{cases} z \in \mathcal{F}(D,s) & : \ z_s \\ z \notin \mathcal{F}(D,s) & : \ z_s + z - z_{s,D} \end{cases} \\ &= z_s + e_h^t(z - z_{s,D})e_h \end{aligned} \tag{1}$$

It is easy to prove that $P_{h,D}$ is a valid reindexing transformation. Clearly $P_{h,D}(z) = (e_1^t(z - z_{s,D}), e_2^t z)^t$ for $h = 1$ and $P_{h,D} = (e_1^t z, e_2^t(z - z_{s,D}))^t$ for

$h = 2$. The reindexing $P_{1,D}$ is illustrated in figure 2. In this figure, the projection of domain $D' = P_{1,D}(D)$ along e_2 leads to an optimal allocation. In this example, the reindexing $P_{1,D}$ improves the potential parallelism by projection as the projection of the original domain along any direction does not lead to an optimal allocation.

The following proposition shows that the whole domain D' is located either at the left or at the right hand side of the e_l axis.

Proposition 4 *For all $z \in D$, either $e_h^t(z - z_{s,D}) \geq 0$ or $e_h^t(z - z_{s,D}) \leq 0$.*

PROOF. Assume that there exist two points z and z' of D such that $e_h^t(z - z_{s,D}) > 0$ and $e_h^t(z' - z'_{s,D}) < 0$. This implies that $z = z_{s,D} + ce_h$ and $z' = z'_{s,D} - c'e_h$ where c and c' are two strictly positive integers. Because D is convex along e_h, segments $Seg(z, z_{s,D})$ and $Seg(z', z'_{s,D})$ are in D. This implies that $z_{s,D} + e_h$ and $z'_{s,D} - e_h$ belong to D. This is in contradiction either with the definition of $z_{s,D}$ or of $z'_{s,D}$ as $s = e_h$ or $s = -e_h$. □

Proposition 5 *The projection of D' along e_l leads to an optimal allocation.*

PROOF. By definition, a point z' of D' is allocated to PE $[e_h^t z']$. Let $m = \max\{e_h^t z' \mid z' \in D'\} - min\{e_h^t z' \mid z' \in D'\} + 1$. Clearly, the number of PEs obtained by projecting D' along e_l is no more than m. Since all of D' is to one side of the e_l axis (by proposition 4), we know that $m = \max\{|e_h^t z'| \mid z' \in D'\} + 1$. Let z be a point of D' for which $|e_h^t z \mid = m - 1$. Since $z, z_s \in D'$, and D' is convex along e_h, $Seg(z_s, z) \subset D'$. This segment contains m integral points having the same computation date. Hence, the allocation is optimal. □

Note that projecting D' along direction e_l amounts to allocating point $z \in D$ to the PE numbered $|e_h^t z'| = |e_h^t(z - z_{s,D})|$, where $z' = P_{h,D}(z)$. In the corresponding array, input and output operations are performed by a PE located at an extremity of the array if all the input/output points belong to $\mathcal{F}(D, s)$. This property is suitable for VLSI implementation [12, 14].

Case 2: *No vector of the canonical basis is collinear to λ.*

We reduce the problem to the previous case. To do so we apply to the original domain D a reindexing function q that leads to a new timing vector

that is parallel to a vector e_l of the canonical basis. For this purpose, first note that $e_l = U^{-t}\lambda$ for some unimodular matrix U of order 2 as the columns of e_l^t and those of λ^t generate the same lattice [41]. Constructing matrix U is easy: one only needs to compute the Hermite form [41] of λ^t. From proposition 3 we can set $q : z \rightarrow Uz$. The number of PEs is minimized by applying to the new index domain $D' = q(D)$ the allocation strategy described for case 1. This is illustrated in figure 1.

As stated earlier in case 1, this mapping strategy allocates point $z' \in D'$ to the PE numbered $|e_h^t(z' - z'_{s,D'})|$ where s denotes either e_h or $-e_h$ with $h \neq l$. Proposition 6 shows that this amounts to allocate point $z \in D$ to the PE numbered $|e_h^t U(z - z_{s',D})|$ with $s' = U^{-1}s$.

Proposition 6 *Given two points $z, z' \in D$. If $z' = Uz$ then $z'_{s,D'} = Uz_{s',D}$ where $s' = U^{-1}s$.*

PROOF. First note that s is the direction of the timing segments of D'. This implies that the timing segments of D are of direction $s' = U^{-1}s$. In addition, the timing segments of D' are segments because the image of a segment by an unimodular transformation is also a segment. This means that D' is convex along direction s.

Now, assume that $z' = Uz$. We have $z' = z'_{s,D'} + cs$ and $z = z_{s',D} + c's'$, where c' and c are two negative integers. Because $z' = Uz$ we get:

$$z'_{s,D'} = (c' - c)s + Uz_{s',D} \tag{2}$$

As $z_{s',D} \in D$ and $q(D) = D'$ we have $Uz_{s',D} \in D'$. Thus, (2) implies that $(c - c')s + z'_{s,D'} \in D'$. Because $z'_{s,D'} + s \notin D'$ we have $(c - c') \leq 0$ as D' is convex along direction s. Similarly, as $z'_{s,D'} \in D'$ and $D = q^{-1}(D')$, (2) implies that $U^{-1}((c'-c)s+Uz_{s',D}) \in D$. This means that $(c'-c)s'+z_{s',D} \in D$. This implies that $(c' - c) \leq 0$ as D is convex along direction s'. Therefore $c = c'$, and as a consequence $z'_{s,D'} = Uz_{s',D}$. □

4.2 Generalizing to n-dimensional Domains

We now extend the method to higher dimensions. While the discussion above gives an insight on the method, it does not present a constructive approach, especially when we have more than two dimensions. This is done using "slice-based partitions" as described next.

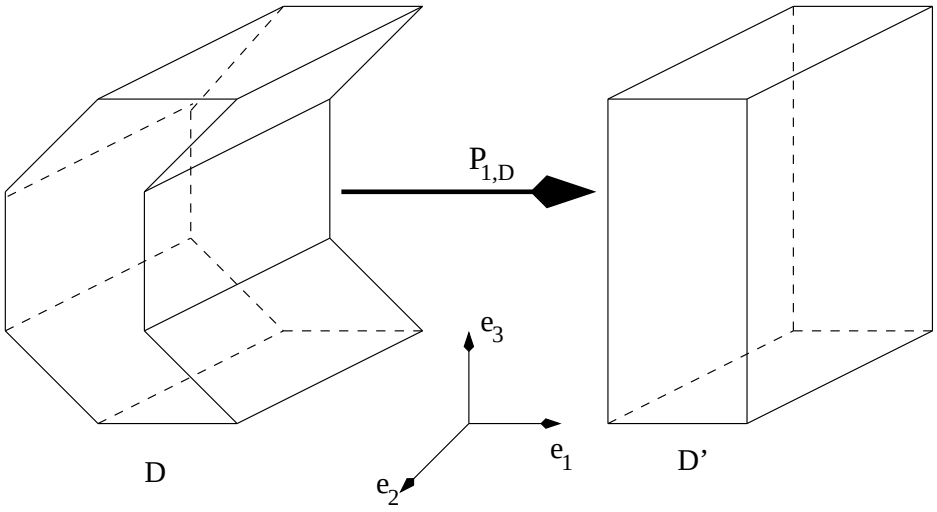

Figure 3: Illustration of the reindexing $P_{1,D}$ on a three-dimensional domain. The timing vector is e_3

4.2.1 Minimizing the number of PEs along one direction

Given two primitive and linearly independent vectors $v_1 = x_1e_1 + \ldots + x_ne_n$ and $v_2 = y_1e_1 + \ldots + y_ne_n$, we show how the tasks of the iteration domain D can be allocated to PEs so as to minimize the number of PEs along direction $\langle v_1, v_2\rangle$, i.e., so as to minimize the number of PEs on all $\langle v_1, v_2\rangle$-slice of D. Indeed, as in the partition method [37, 42], our starting point is to partition D into $\langle v_1, v_2\rangle$-slices denoted $D_1, D_2, \ldots, D_r$. However, we do not search for an allocation function that minimizes the number of PEs on all D_i "all at once" but proceed dimension b y dimension. Instead of using ad hoc approaches as in the partition method, we proceed in a systematic way. We perform a preprocessing by reindexing that provides a projection direction that separately allocates the tasks of each D_i so as to minimize the number of PEs on each D_i.

Note that if v_1 and v_2 are parallel to the timing surfaces the minimal number of PEs required by each $\langle v_1, v_2\rangle$-slice is equal to its cardinality. Because of this, we assume that the $\langle v_1, v_2\rangle$-slices are not parallel to the timing surfaces, i.e., $\lambda^t v_1 \neq 0$ or $\lambda^t v_2 \neq 0$. In what follows, we assume that $\lambda^t v_2 \neq 0$ without loss of generality. Depending on the direction of the timing vector λ we consider two main cases.

Case 1: *λ is collinear to some vector e_l of the canonical basis*

Sub-Case 1.1. The $\langle v_1, v_2\rangle$-slices are parallel to some e_h such that $h \neq l$. Let s be either e_h or $-e_h$. We now generalize the reindexing defined by equation (1). The main idea is again to shift the timing segments of all D_i along direction s so as to place the points of $\mathcal{F}(D_i, s)$ on the $(n-1)$-hyperplane of Cartesian equation $e_h^t z = 0$. To this end, we set:

$$\begin{aligned} P_{h,D}(z) &= P_{h,D}(z_{s,D}) + (z - z_{s,D}) \\ &= z_s + (e_h^t(z - z_{s,D}))e_h \\ &= (e_1^t z, \ldots, e_{h-1}^t z,\ e_h^t(z - z_{s,D}), e_{h+1}^t z, \ldots, e_n^t z)^t \end{aligned} \tag{3}$$

The reindexing $P_{h,D}$ maintains all points of D on their timing surface and on the supporting 2-hyperplane of its $\langle v_1, v_2\rangle$-slice. The points of

$$\mathcal{F}(D, s) = \bigcup_{k=1}^{k=r} \mathcal{F}(D_i, s)$$

are mapped into points belonging to the $(n-1)$-hyperplane of Cartesian equation $e_h^t z = 0$. This is illustrated in figure 3. The projection of domain $D' = P_{1,D}(D)$ along direction e_3 provides a space optimal array. In this example, the reindexing $P_{1,D}$ improves the potential parallelism by projection as the projection of the original domain along any direction does not provide a space optimal array.

The reindexing $P_{h,D}$ acts similarly on each $\langle v_1, v_2\rangle$-slice. It maps the points of $\mathcal{F}(D_i, s)$ into points belonging to the intersection of the supporting 2-hyperplane of D_i and the $(n-1)$-hyperplane of Cartesian equation $e_h^t z = 0$. This implies that $P_{h,D}(\mathcal{F}(D_i, s))$ is parallel to vector $v_2 - y_h e_h$. Hence, we have the following:

Proposition 7 *The projection of $D' = P_{h,D}(D)$ along direction $v_2 - y_h e_h$ separately allocates the points of all $\langle v_1, v_2\rangle$-slice=$\langle e_h, v_2 - y_h e_h\rangle$-slice so as to minimize the number of PEs on all $\langle e_h, v_2 - y_h e_h\rangle$-slice.*

Proof. The proof is similar to that of proposition 5. □

Proposition 7 shows that the projection of $D' = P_{h,D}(D)$ along direction $v_2 - y_h e_h$ leads to an array in which the number of PEs is minimized along direction e_h, i.e., for each line of PEs along direction e_h there is an instant for which all PEs are simultaneously active. As stated earlier in the case

where D is of dimension 2, the resulting array is such that input/output operations are performed by PEs located at the border of the array if all the input/output points belong to $\mathcal{F}(D, s)$, a property that is suitable for VLSI implementations.

Sub-Case 1.2. The $\langle v_1, v_2 \rangle$-slices are not parallel to some e_h with $h \neq l$. Here we perform a reindex that reduces the problem to sub-case 1.1. The idea behind this transformation is to locate all D_i parallel to some e_h with $h \neq l$ so as to leave unchanged the timing vector λ. To this end, we consider a unimodular vector v_0 that is collinear to vector $x_l v_2 - y_l v_1$. As the columns of v_0^t and those of e_h^t generate the same lattice we have $e_h = U_n v_0$ for some unimodular matrix U_n of order n [41]. Because $e_l^t v_0 = 0$ and $e_l^t e_h = 0$ we can assume that the l^{-th} column and the l^{-th} line of U_n are respectively e_l and e_l^t. For instance, if $l = n$ and $h = 1$ we have

$$U_n = \begin{pmatrix} U_{n-1} & 0_{n-1} \\ 0_{n-1}^t & 1 \end{pmatrix} \tag{4}$$

where U_{n-1} is an unimodular matrix of order $n-1$ and 0_{n-1} is the null vector of Z^{n-1}. Note that $U_n^t e_l = e_l$. This implies that $U_n^{-t} e_l = e_l$. Stemming from proposition 3, this implies that the timing vector λ is left unchanged by the reindexing $q : z \rightarrow U_n z$. Because the potential parallelism is invariant under valid reindexing functions [12], the problem of minimizing the number of PEs on each $\langle v_1, v_2 \rangle$-slice of D can be translated into the the problem of minimizing the number of PEs on each $\langle q(v_1), q(v_2) \rangle$-slice of $D' = q(D)$. This last problem can be solved with the allocation strategy proposed for sub-case 1.1.

Case 2: *No vector of the canonical basis is collinear to λ.* We perform a reindexing that reduces the problem to case 1. To do so we consider a reindexing function q that leads to a new timing vector that is parallel to some e_l. As e_l and λ are unimodular vectors we have $e_l = U_n^{-t} \lambda$ for some unimodular matrix U_n of order n. From proposition 3 we can set $q : z \rightarrow U_n z$. Now, we use the allocation strategy of case 1 to minimize the number of PEs on each $\langle q(v_1), q(v_2) \rangle$-slice of $D' = q(D)$.

4.2.2 Space-optimality

The allocation technique presented in section 4.2.1 is based on reindexing transformations that compress the initial iteration domain along a given direction in order to construct a new DAG that leads to an array which is such that the number of PEs is minimized along the given direction. This technique can further be optimized stemming from propositions 8 and 9. The idea behind these propositions is that any array with a 100% pipelining rate whose number of PEs is minimized along any direction is space-optimal. Indeed, this idea can be exploited by successively compressing the initial iteration domain along a number of directions, using reindexing transformations introduced in section 4.2.1. As the new obtained iteration domain is compressed along many directions, it enables the projection method to provide an array whose number of PEs is minimized along many directions. Note that the 100% pipelining rate criterion is easy to satisfy. It is easy to see that if the iteration domain is convex along a projection vector ξ, that satisfies $|\lambda^t \xi| = 1$, then the pipelining rate of the resulting array is 100%. Recall that the new allocation method applies the projection method with $\xi = \lambda \in \{e_1,\ e_2,\ e_3, \ldots,\ e_n\}$. Therefore, propositions 8 and 9 guarantee the space-optimality of the new allocation technique.

Proposition 8 *Any array with a* 100% *pipelining rate that is such that the intersection of working intervals of any pair of PEs is not empty is space-optimal.*

Proof. Denote np the number of PEs, pp the potential parallelism and t a date at which the maximum number of PEs are working. Clearly $np \geq pp$. Assuming that $np > pp$, let's denote $C = \{c_1,\ c_2, \ldots, c_{pp}\}$ the set of cells scheduled to work at date t, $f(c)$ (resp. $l(c)$) the first working date (resp. last working date) of cell c. There is then at least one PE $p \notin C$ that is not active at date t. Because the array has a 100% pipelining rate, each PE works at all activation date between its first and last working dates. This implies that either $t < f(p)$ or $l(p) < t$. On the other hand we have $I(c) = [f(c),\ l(c)] \cap [f(p),\ l(p)] \neq \emptyset$ for all $c \in C$ as the intersection of the working intervals of any pair of PEs is not empty. This implies that $f(p) \in I(c)$ for all c if $t < f(p)$ and $l(p) \in I(c)$ for all c if $t < l(p)$. It follows that the PEs of C are simultaneously working at date $f(p)$ or at date $l(p)$. Therefore $pp + 1$ PEs are working at the same date. Hence a contradiction.

□

The proof of the following proposition is simple.

Proposition 9 *In any array in which the number of PEs is minimal along all direction, the intersection of activation intervals of every pair of PEs is not empty.*

5 Examples

Here we illustrate the method on Matrix Multiplication (MM) and the Cholesky Factorization (CF). We derive asymptotically space-time optimal arrays for MM and CF.

Matrix multiplication [13, 19, 15]

The computation of the product $C = AB$ of two square matrices A and B of order N can be defined by the following SURE:

Initialization:

For $0 \leq i \leq N-1,\ 0 \leq j \leq N-1,\ 0 \leq k \leq N-1$

$A(i,-1,k) = A_{i,k}$

$B(-1,j,k) = B_{k,j}$

$C(i,j,-1) = 0$

Computation:

For $0 \leq i \leq N-1,\ 0 \leq j \leq N-1,\ 0 \leq k \leq N-1$

$A(i,j,k) = A(i,j-1,k)$

$B(i,j,k) = B(i-1,j,k)$

$C(i,j,k) = C(i,j,k-1) + A(i,j-1,k)B(i-1,j,k)$

Output:

For $0 \leq i \leq N-1,\ 0 \leq j \leq N-1$

$C_{i,j} = C(i,j,N-1)$

The iteration space is $D^{(0)} = \{(i,j,k) \in Z^3 \mid 0 \leq i \leq N-1,\ 0 \leq j \leq N-1,\ 0 \leq k \leq N-1\}$. The dependence vectors are the vectors of the canonical basis of Z^3: e_1, e_2 and e_3. A corresponding optimal timing function

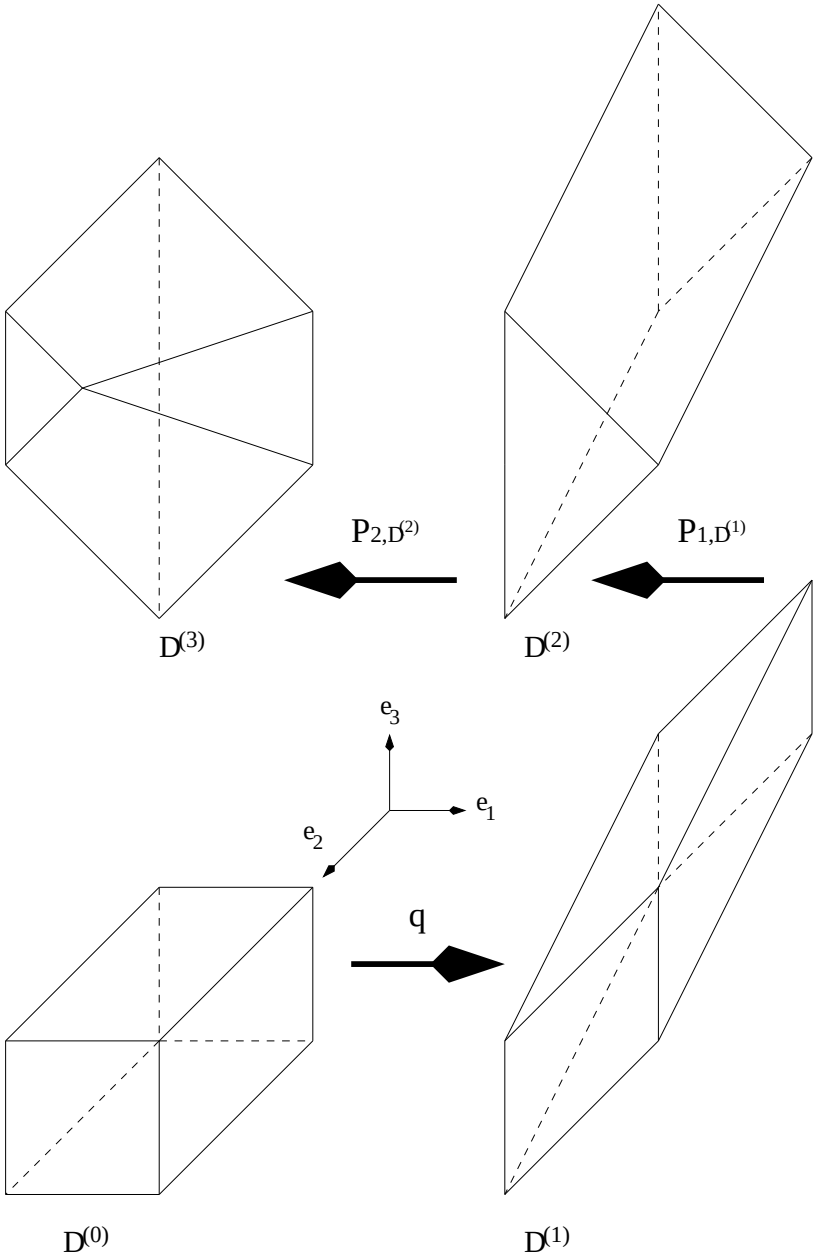

Figure 4: Reindexing process for matrix multiplication

is $t(i,j,k) = i + j + k$. The projection of $D^{(0)}$ along vector e_3 leads to an $N \times N$ array of N^2 PEs. To reduce the number of PEs we proceed as follows. First, the timing vector $e_1 + e_2 + e_3$ is mapped into e_3 with the reindexing q:

$$q(i,j,k) = (i,j,t(i,j,k) = (i,j,i+j+k) \tag{5}$$

This leads to a new iteration domain $D^{(1)} = q(D^{(0)}) = \{(i,j,k) \in Z^3 \mid 0 \leq i \leq N-1,\ 0 \leq j \leq N-1,\ 0 \leq -i-j+k \leq N-1\}$ [13, 19, 15]. This is illustrated in figure 4. Second, $D^{(1)}$ is compressed along direction e_1 with the reindexing $P_{1,D^{(1)}}$ to obtain $D^{(2)} = P_{1,D^{(1)}}(D^{(1)}) \subseteq \{(i,j,k) \in Z^3 \mid 0 \leq i \leq N-1,\ 0 \leq j \leq N-1,\ i+j-k \leq 0,\ i-j+k \leq 2(N-1)\}$ [13, 19, 15]. Similarly, $D^{(2)}$ is compressed along direction e_2 with the reindexing $P_{2,D^{(2)}}$ to obtain $D^{(3)} = P_{2,D^{(1)}}(D^{(2)}) \subseteq \{(i,j,k) \in Z^3 \mid 0 \leq i \leq N-1,\ 0 \leq j \leq N-1,\ 2i+j \leq 2(N-1)$ [13, 19, 15]. Stemming from the results of section 7 we have:

$$P_{1,D^{(1)}}(i,j,k) = \begin{cases} (i,j,k) & \text{if } -j+k-(N-1) \leq 0 \\ ((N-1)+i+j-k,j,k) & \text{if } -j+k-(N-1) > 0 \end{cases} \tag{6}$$

$$P_{2,D^{(2)}}(i,j,k) = \begin{cases} (i,j,k) & \text{if } i+k-2(N-1) \leq 0 \\ (i,-i+j-k+2(N-1),k) & \text{if } i+k-2(N-1) > 0 \end{cases} \tag{7}$$

Third, $D^{(3)}$ is projected along direction e_3 to obtain an asymptotically space-time optimal array of $\frac{3}{4}N^2 + \Theta(N)$ [19, 13] PEs.

Cholesky factorization [8, 11, 13]

The CF is defined as follows: Given a $N \times N$ symmetric positive definite matrix A, the CF calculates a lower triangular matrix L such that $A = LL^t$. It is defined by the following well known affine recurrence equations:

For $(i,j,k) \in D^{(0)} = \{1 \leq j \leq i \leq N \ \wedge\ 0 \leq k \leq j\}$

$$L(i,j,k) = \begin{cases} A_{j,i} & \text{if } 1 \leq j \leq i \leq N \wedge k = 0 \\ \frac{L(i,j,k-1)}{L(j,j,j-1)} & \text{if } 1 \leq i \leq N \wedge 1 \leq j \leq i-1 \wedge k = j \\ L(i,j,k-1)^{1/2} & \text{if } 1 \leq i \leq N \wedge j = i \wedge k = i \\ L(i,j,k-1) - & \text{if } 1 \leq j \leq i \leq N \wedge 1 \leq k \leq j-1 \\ \quad L(i,k,k)L(j,k,k) & \end{cases} \tag{8}$$

Regarding the dependencies of (8) a uniform version can be obtained with e_1, e_2 and e_3 as the dependence vectors [33], and this without changing the domain $D^{(0)}$ of equations (8). An optimal timing function corresponding to such a uniformization is $t(i,j,k) = i + j + k$. By projecting the domain $D^{(0)}$ along vector $e_1 + e_2$, we obtain a triangular orthogonally connected array of $N(N+1)/2$ PEs. The number of PEs can be reduced as follows. First, the timing vector $e_1 + e_2 + e_3$ is mapped into e_1 with the reindexing $q(i,j,k) = (i+j+k, j, k)$. Second, $D^{(1)} = q(D^{(0)}) \subseteq \{(i,j,k) \in Z^3 \mid -i + 2j + k \leq 0,\ i - j - k - N \leq 0,\ -k \leq 0,\ -j + k \leq 0\}$ is mapped into $D^{(2)} = P_{3,D^{(1)}}(D^{(1)}) \subseteq \{(i,j,k) \in Z^3 \mid j + k - N \leq 0,\ -k \leq 0,\ i - 2j + k - N \leq 0,\ -i + 2j + k \leq 0,\ -j + k \leq 0\ \}$ and $D^{(2)}$ is in turn mapped into $D^{(3)} = P_{2,D^{(2)}}(D^{(2)})$ [11, 13]. Third, a asymptotically space-time optimal array of $\frac{N^2}{8} + \Theta(N)$ [11, 13] PEs is obtained by projecting $D^{(3)}$ along vector e_1. This improves the results obtained from the instruction shift method [8] and the partition method [11, 13]. They lead to arrays of $\frac{N^2}{6} + \Theta(N)$ PEs. Note that the projection of $D^{(2)}$ along vector e_1 leads to an array of $\frac{N^2}{4} + \Theta(N)$ [11, 13] PEs. Therefore the numbers of PEs is reduced by a factor of two at each of the last two steps of the reindexing process. Stemming from the results of section 7 we have:

$$P_{3,D^{(1)}}(i,j,k) = \begin{cases} (i,j,-i+j+k+N) & \text{if } (i,j,k) \in D_1^{(1)} \\ (i,j,k) & \text{if } (i,j,k) \in D_2^{(1)} \end{cases}$$

where

$$D_1^{(1)} = \{(i,j,k) \in D^{(1)} \mid -i + j + N < 0\}$$
$$D_2^{(1)} = \{(i,j,k) \in D^{(1)} \mid -i + j + N \geq 0\}$$

$$P_{2,D^{(2)}} = \begin{cases} (i, -\frac{1}{2}i + j - \frac{1}{2}k + \frac{N}{2}, k) & \text{if } (i,j,k) \in D_{1,0}^{(2)} \\ (i, -\frac{1}{2}i + j - \frac{1}{2}k + \frac{N+1}{2}, k) & \text{if } (i,j,k) \in D_{1,1}^{(2)} \\ (i, j-k, k) & \text{if } (i,j,k) \in D_2^{(2)} \end{cases}$$

where

$$D_{1,0}^{(2)} = \{(i,j,k) \in D^{(2)} \mid -i + j + N < 0 \wedge (i + k - N) (\text{mod } 2) = 0\}$$

$$D_{1,1}^{(2)} = \{(i,j,k) \in D^{(2)} \mid -i + j + N < 0 \wedge (i + k - N) (\text{mod } 2) = 1\}$$

$$D_2^{(2)} = \{(i,j,k) \in D^{(2)} \mid -i + j + N \geq 0\}$$

6 Comparison with other methods

In this section, we compare our allocation method with the projection method, the grouping (or clustering) method, the instruction shift method and the partition method. Our method is sustained by a space-optimality result (section 4.2.2) while the methods mentioned above do not guarantee space-optimality. The space-optimality is achieved by compressing the initial iteration domain along all directions with reindexing transformations introduced in section 4.2.1. In what follows, we establish that the lowest space complexity that both the projection method and the grouping method can give for the initial iteration domain is an upper bound on the space complexity of our method, i.e., our method can do no worse.

6.1 Preliminary result

Let λ be a timing vector and let $v_1 = x_1e_1 + \ \ldots \ + x_ne_n$ and $v_2 = y_1e_1 + \ldots + y_ne_n$,$v_1$ be two unimodular linearly independent vectors such that $\langle v_1, v_2\rangle$-slices are not parallel to timing surfaces.

We have the following result.

Proposition 10 *If λ is collinear to some e_l and $\langle v_1, v_2\rangle$-slices are parallel to some e_h with $l \neq h$ then $NPE(v_2 - y_he_h, P_{h,D}(D)) \leq NPE(v_2, D, e_h) \leq NPE(v_2, D)$.*

Proof. Because the $\langle v_1, v_2\rangle$-slices are not parallel to timing surfaces, i.e., $\lambda^t v_1 \neq 0$ or $\lambda^t v_2 \neq 0$, we can assume without loss of generality that $\lambda^t v_2 \neq 0$.

It is easy to see that $NPE(v_2, D, e_h) \leq NPE(v_2, D)$. $NPE(v_2, D) = \sum_{i=1}^{i=r} NPE(v_2, D_i)$ as points allocated to a same PE belong to a same line of direction v_2, and obviously to a same $\langle v_1, v_2\rangle$-slice. Denote $\xi_0 = v_2 - y_he_h$ and $D'_i = P_{h,D}(D_i))$. We have $NPE(v_2, D, e_h) = \sum_{i=1}^{i=r} NPE(v_2, D_i, e_h)$ and $NPE(\xi_0, D') = \sum_{i=1}^{i=r} NPE(\xi_0, D')$. On the other hand, the reindexing $P_{h,D}$ retains the potential parallelism on each $\langle v_1, v_2\rangle$-slice [12], i.e., the potential parallelism of D'_i is equal to that of D_i. In addition, according to proposition 7, the potential parallelism of D'_i is equal to $NPE(\xi_0, D'_i)$. This implies that $NPE(\xi_0, D'_i) \leq NPE(v_2, D_i)$ and $NPE(\xi_0, D'_i) \leq NPE(v_2, D_i, e_h)$. Therefore $NPE(\xi_0, D') \leq NPE(v_2, D, e_h) \leq NPE(v_2, D)$. □

6.2 Two strong comparison results

Here, we establish two strong comparison results. We first establish a strong comparison result between our allocation method and the projection method. Then, we establish another strong comparison result between our allocation method and the clustering method.

To simplify the presentation, we assume without loss of generality that the timing vector $\lambda = e_n$. Thus $t = [e_n, \alpha]$.

Denote $D^{(0)} = D$, $D^{(h)} = P_{h,D^{(h-1)}}(D^{(h-1)})$ with $h \in \{1,\ 2, \ldots, n-1\}$ and assume that $D^{(h-1)}$ is convex along direction e_h. Consider an unimodular vector $\xi = \xi_1 e_1 + \xi_2 e_2 + \ldots + \xi_n e_n$ that corresponds to a valid projection vector. As $\lambda = e_n$, we get $\xi_n \neq 0$. Denote $\xi^{(h)}$, $h \in \{0,\ 1,\ 2, \ldots, n-1\}$, an unimodular vector that is collinear to vector $\xi_{h+1}e_{h+1} + \xi_{h+2}e_{h+2} + \ \ldots + \xi_n e_n$.

Corollary 1 *We have* $NPE(D^{(n-1)}, \lambda) \leq NPE(D^{(0)}, \xi)$ *for all vector* ξ *such that* $\lambda^t \xi \neq 0$.

Proof. From proposition 10 we get

$$NPE(\xi^{(h)}, D^{(h)}) \leq NPE(\xi^{(h-1)}, D^{(h-1)}),\ h \in \{1, 2, 3, \ldots, n-1\}$$

This implies that $NPE(e_n, D^{(n-1)}) \leq NPE(\xi, D^{(0)})$. □

As stated earlier, Wong and Delosme [47], Shang and Fortes [40], Ganapathy and Wah [21], have developed an approach based on integer programming to get the best projection direction possible. But for large problem instances the time spent to solve such programs cannot be neglected [9]. Here we propose another approach based on reindexing transformations that permits to do better. A straightforward consequence of corollary 1 is that it provides a procedure based on a preprocessing by reindexing that systematically leads to arrays whose number of PEs is bounded from above by the number of PEs obtained from the best projection direction. We also have the following.

Theorem 1 *We have* $NPE(D^{(n-1)}, \lambda) \leq \frac{1}{|\lambda^t \xi|} NPE(D^{(0)}, \xi) + \sum_{h=0}^{h=n-2} \Theta(nl_h)$ *for all vector* ξ *such that* $\lambda^t \xi \neq 0$ *where* nl_h *denotes the number of distinct* $\langle e_{h+1}, \xi^{(h)} \rangle$*-slices of* $D^{(h)}$.

Proof. From the definition of $\xi^{(h)}$, $h \in \{0,\ 1,\ 2,\ \ldots,\ n-2\}$, there exists an integer number l_h such that $e_n^t \xi^{(h)} = e_n^t \xi^{(h+1)} l_h$. Denote A_h the

array obtained by projecting domain $D^{(h)}$ along direction $\xi^{(h)}$. In this array, all PE works only once over $|\lambda^t \xi^{(h)}| = |e_n^t \xi^{(h)}|$ successive activation dates. Now, let's examine the working dates of PEs belonging to a same line of direction e_{h+1} denoted L_h. All the tasks allocated to L_h belong to a same $\langle \xi^{(h)}, e_{h+1} \rangle$-slice=$\langle e_{h+1}, \xi^{(h+1)} \rangle$-slice of $D^{(h)}$. The computation dates d_1 and d_2 of two arbitrary tasks belonging to a same $\langle e_{h+1}, \xi^{(h+1)} \rangle$-slice are such that $(d_1 - d_2) \bmod |e_n^t \xi^{(h+1)}| = 0$. This implies that there are $|l_h|$ activation dates of L_h over $|e_n^t \xi^{(h)}|$ successive activation dates of A_h as $e_n^t \xi^{(h)} = e_n^t \xi^{(h+1)} l_h$. It follows that all PE of L_h works only once over $|l_h|$ successive activation dates of L_h. Thus by applying the clustering technique to any line of PEs parallel to vector e_{h+1}, the space complexity of array A_h is reduced by a factor of $|l_h|$, i.e.,

$$NPE(\xi^{(h)}, D^{(h)}, e_{h+1}) = \frac{1}{|l_h|} NPE(\xi^{(h)}, D^{(h)}) + \Theta(n l_h)$$

On the other hand from proposition 10 we get

$$NPE(\xi^{(h+1)}, D^{(h+1)}) \leq NPE(\xi^{(h)}, D^{(h)}, e_{h+1})$$

for all $h \in \{0,\ 2,\ 3, \ldots,\ n-2\}$. This implies that

$$NPE(\xi^{(h+1)}, D^{(h+1)}) \leq \frac{1}{|l_h|} NPE(\xi^{(h)}, D^{(h)}) + \Theta(n l_h)$$

for all $h \in \{0, 1, 3, \ldots, n-2\}$. It follows that

$$NPE(\xi^{(h+1)}, D^{(h+1)}) \leq \frac{1}{|l_0 l_1 l_2 \ldots l_h|} NPE(\xi^{(0)}, D^{(0)}) + \sum_{i=0}^{i=h} \Theta(n l_i)$$

for all $h \in \{0, 1, 3, \ldots, n-2\}$. Moreover for $h = n-2$ we have $|l_0 l_1 l_2 \ldots l_h| = |\lambda^t \xi|$. Hence the result. □

A straightforward consequence of theorem 1 is that the new allocation method can be applied so as to systematically derive arrays whose space complexities are bounded from above by the lowest space complexities that the grouping method can give. Compared to other allocation methods, we are not aware of any similar strong comparison result between the instruction shift method and the grouping method, nor between the partition method and the grouping method. No similar strong comparison result exists even between the instruction shift method and the projection method, nor between partition method and the projection method.

7 Closed form of the allocation function

We now investigate the implementation of the new allocation method in tools for (semi) automatic derivation of VLSI arrays. In particular, we consider the MMAlpha [22] system based on the functional language Alpha [28]. Alpha programs are almost identical to the systems of equations that we have used in this paper. Each equation is defined over a possibly parameterized index domain that is a finite union of polyhedra [32]. Such domains constitute an abstract data type with closure under certain operations, notably intersection, union, and image by bijective affine index transformations. Such an index transformation is valid if every point of the index domain to which it is applied has a unique integral image. In particular, any unimodular transformation is always a valid index transformation.

In order to implement our method in MMAlpha, we only need to show how to implement allocation function (3) since it is the only function that may not correspond to an obvious unimodular transformation. The function, repeated below for convenience, is a piecewise affine function, but specified implicitly.

$$P_{h,D}(z) = (e_1^t z, \ldots, e_{h-1}^t z,\ e_h^t(z - z_{s,D}), e_{h+1}^t z, \ldots, e_n^t z)^t$$

Let s denote either e_h or $-e_h$, where e_h is the h^{-th} vector of the canonical basis. To simplify the presentation, we assume that $D = Z^n \bigcap P$ where P is a single convex[1] polytope, Q^n defined by an irredundant set of integral inequalities $\{z \in Q^n \mid Az \leq b\}$. Therefore, any row A_i of matrix A defines a facet $F_i = \{z \in P \mid A_i z - b_i = 0\}$ [41] of D. Among the F_i's, we choose those that satisfy $A_i s > 0$. We assume without loss of generality that these facets are $F_1,\ F_2,\ F_3, \ldots, F_r$ for some integer r. Proposition 11 shows that such facets exist.

Proposition 11 *There exists an integer i such that $A_i s > 0$.*

PROOF. Suppose that $A_i s \leq 0$ for all i. This implies that $A_i(z + \alpha s) \leq b_i$ for any positive rational number α and for any z belonging to P. This implies that $z + \alpha s \in P$, which in turn implies that s is a ray of P. This contradicts the fact that P is bounded. □

[1] In the general case, we would require a preprocessing step to first decompose the domain into a number of sub-domains.

Step 1: We first partition D into dense sub-domains [32]. Let $\sigma_i(z)$ denote the projection of point z on the supporting hyperplane of facet F_i along direction s. Let $P_i = \{z \in P \mid \sigma_i(z) \in F_i\}$ and $D_i = Z^n \bigcap P_i$ for $i = 1, 2, \ldots, r$. The following proposition provides us a constructive method to determine the dense sub-domains.

Proposition 12 *We have*

$$P = \bigcup_{k=1}^{k=r} P_k$$

and each P_k is a convex polytope. As a result, $D = \bigcup_{k=1}^{k=r} D_k$.

PROOF. Let $z \in P$. Denote $\alpha_z = \min\{\frac{b_k - A_k z}{A_k s} \mid k = 1, 2, \ldots, r\}$, i.e., $\alpha_z = \frac{b_i - A_i z}{A_i s}$ for some i. We have $A(z + \alpha_z s) \leq b$ and $A_i(z + \alpha_z s) - b_i = 0$. This implies that $\sigma_i(z) = z + \alpha_z s \in F_i$, which in turn implies that $z \in P_i$.

Now consider two points z_1 and z_2 of a given P_i. Let $z_3 \in Seg(z_1, z_2)$. Because P is convex we have $z_3 \in P$. In addition, a simple linear algebraic argument shows that $\sigma_i(z_3) \in Seg(\sigma_i(z_1), \sigma_i(z_2))$, and this implies that $\sigma_i(z_3) \in F_i$ because F_i is convex. Thus $z_3 \in P_i$, and $Seg(z_1, z_2) \subseteq P_i$. Therefore P_i is convex. □

Step 2: Next, we partition the D_i's into sparse *periodic* domains.

Proposition 13 *Let z be a point of P_i and let α be a positive rational number. We have:*

$$z + \alpha s \in P_i \text{ iff } A_i(z + \alpha s) \leq b_i \tag{9}$$

PROOF. Let us assume that $A_i(z + \alpha s) \leq b_i$. From the definition of σ_i we get $\sigma_i(z) = z + \left(\frac{b_i - A_i s}{A_i s}\right) s$. This implies that $z + \alpha s \in Seg(z, \sigma_i(z))$ as $0 \leq \alpha \leq \frac{b_i - A_i z}{A_i s}$. We have $Seg(z, \sigma_i(z)) \subseteq P$ as domain P is convex along direction s. Thus $z + \alpha s \in P$. In addition, $\sigma_i(z + \alpha s) = \sigma_i(z)$. Therefore $z + \alpha s \in P_i$. The necessary condition is easy to check. □

Let $z \in D_i$. Clearly, $z_{s,D} = z + \alpha s$ for some integer number α. We have $A_i(z + \alpha s) \leq b_i$. From proposition 13, we get $A_i(z + (\alpha + 1)s) > b_i$ as $z_{s,D} + s \notin P_i$. Thus

$$\frac{b_i - A_i z}{A_i s} - 1 < \alpha \leq \frac{b_i - A_i z}{A_i s}.$$

This implies that

$$\alpha = \left\lfloor \frac{b_i - A_i z}{A_i s} \right\rfloor,$$

which means that

$$\alpha = \frac{b_i - A_i z - j}{A_i s} \tag{10}$$

where $(b_i - A_i z) \bmod A_i s = j$.

Stemming from (10), we consider a partition $D_{i,j} = \{z \in D_i \mid (b_i - A_i z) \bmod A_i s = j\}$, $j = 0,\ 1,\ 2,\ \ldots,\ A_i s - 1$ of D_i. $D_{i,j}$ is a sparse domain if $j \neq 1$.

Step 3: Analytic expression of the re-indexing function $P_{h,D}$

Assuming that $z \in D_{i,j}$, we have from (3)

$$P_{h,D}(z) = \left(e_1^t z,\ \ldots, e_{h-1}^t z,\ \frac{e_h^t s}{A_i s}(-b_i + A_i z + j),\ e_{h+1}^t z,\ \ldots,\ e_n^t z \right)^t \tag{11}$$

For $h = 1$ this is equivalent to

$$P_{h,D}(z) = \begin{pmatrix} 1 & B_i \\ 0_{n-1} & I_{n-1} \end{pmatrix} z + \begin{pmatrix} \frac{(j-b_i)(e_1^t s)}{A_i s} \\ 0_{n-1} \end{pmatrix}$$

where 0_{n-1} is the null vector of Z^{n-1}, I_{n-1} the identity matrix of order $n-1$ and

$$\frac{e_1^t s}{A_i s} A_i = \begin{pmatrix} 1 & B_i \end{pmatrix}.$$

This shows that the restriction of $P_{h,D}$ to each $D_{i,j}$ corresponds to an affine transformation. If $A_i s = 1$ we have $D_i = D_{i,0}$ and the restriction of $P_{h,D}$ to D_i corresponds to an unimodular transformation. If $A_i s > 1$ then the $D_{i,j}$'s are sparse periodic domains [32] and the reindexing $P_{h,D}$ corresponds to a piecewise affine transformation. In this particular case, the resulting array may correspond to a piecewise regular array [43].

8 Conclusion

In this paper, we have formally introduced a new allocation method that is based on a preprocessing by reindexing that leads to a new index domain that permits to systematically derive arrays in which the number of PEs is minimized along a number of directions. In addition, the preprocessing permits to improve the potential parallelism by projection of the initial domain. It also permits to allocate input/output nodes to PEs located at the border of the array. This can permit to eliminate eventual additional steps for data loading and result unloading that may force a slowdown of the algorithm [16, 11, 12, 13, 14]. This new allocation approach is simple, suitable for software tools and could be included in a language manipulating recurrence equations like the ALPHA language.

References

[1] M. Bednara and J. Teich, Interface Synthesis for FPGA Based VLSI Processor Arrays, In Proc. of *The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA02)*, Las Vegas, Nevada, U.S.A., (June 2002) 24-27.

[2] A. Benaini and Y. Robert, Space-minimal systolic arrays for Gaussian elimination and the algebraic path problem, *Parallel Comput.*, 15 (1990) 211-225.

[3] J.C. Bermond, C. Peyrat, I. Sakho and M. Tchuenté, Parallelization of the Gaussian elimination on systolic arrays, *J. of Parallel and Distrib. Comput.*, 33 (1996) 69-75.

[4] J.C. Bermond, C. Peyrat, I. Sakho and M. Tchuenté, Parallelization of the Gauss elimination on systolic arrays, *Internal Report*, LRI, 430 (1988).

[5] J. Bu and E.F. Deprettere, Processor clustering for the design of optimal fixed-size systolic arrays, Proceedings of the *Sixth International Parallel Processing Symposium*, (Mar. 1992) 275-282.

[6] P. Cappello, A processor-time minimal systolic array for cubical mesh algorithms, *IEEE Trans. on Parallel and Distrib. Systems*, 3 (1) (1992) 4 -13.

[7] P. Clauss, C. Mongenet and G. R. Perrin, Calculus of space-optimal mappings of systolic algorithms on processor arrays, *IEEE Int. Conf. on Application Specific Array Processors*, (1990) 591-602.

[8] P. Clauss and G.R. Perrin, Optimal mapping of systolic algorithms by regular instructions shifts, *Int. Conf. on Application-Specific Array Processors*, ASAP'94, San Francisco, CA, IEEE Computer Society Press, Silver Spring, MD, (1994) 224-235.

[9] A. Darte, T. Risset and Y. Robert, Synthesizing systolic arrays: Some recent developments, *Int. Conf. on Application Specific Array Processors*, IEEE Computer Soc. Press, (1991) 372-386.

[10] C.T. Djamegni, Contribution to the synthesis of optimal algorithms for regular arrays, *Thèse de Doctorat, Department of Computer Science, University of Yaoundé I-Cameroun*, December 1997.

[11] C.T. Djamegni, Synthesis of space-time optimal systolic algorithms for the Cholesky factorization, *Discrete Mathematics and Theoretical Computer Science*, 5 (2002) 109-120.

[12] C.T. Djamegni, Mapping rectangular mesh algorithms onto asymptotically space-optimal arrays, *J. Parallel and Distrib. Comput.*, 64 (3) (2004) 345-359.

[13] C.T. Djamegni, Méthodes d'optimisation pour la synthèse d'architectures régulières, *Thèse de Doctorat d'Etat, University of Yaoundé I-Cameroun*, December 2005.

[14] C.T. Djamegni, Matrix product on modular linear systolic arrays for algorithms with affine schedule, *J. Parallel and Distrib. Comput.*, 66 (3) (2006) 323-333.

[15] C.T. Djamegni, Exécution d'un graphe cubique de tâches sur un réseau 2D et asymptotiquement optimal, *African Revue in Informatics and Mathematics Applied* (ARIMA),4, 53-65 (2006), http://www.cari-info.org.

[16] C.T. Djamegni, P. Quinton, S. Rajopadhye and T. Risset, Derivation of systolic algorithms for the algebraic path problem by recurrence transformations, *Parallel Comput.* 26 (2000) 1429-1445.

[17] C.T. Djamegni and M. Tchuenté, Scheduling of the DAG associated with pipeline inversion of triangular matrices, *Parallel Process. Lett.*, 6, (1) (1996) 13-26.

[18] C. T. Djamegni and M. Tchuenté, A new dynamic programming algorithm on two dimensional arrays, *Parallel Process. Lett.*, 10 (1) (2000) 15-27.

[19] C.T. Djamegni and M. Tchuenté, Derivation of 2D space-optimal regular arrays for 3D cubical mesh algorithms, CARI'02, *6-th African Conference on Research in Computer Science*, Yaoundé-Cameroun, 2002.

[20] F. Dupont de Dinechin, P. Quinton, S. Rajopadhye and T. Risset, First steps in Alpha, *Technical Report*, IRISA, 1244 (1999).

[21] K.N. Ganapathy and B.W. Wah, Optimal synthesis of algorithm-specific lower-dimensional processor arrays, *IEEE Trans. on Parallel and Distributed System*, 7 (3) (1996) 274-287.

[22] A-C. Guillou, F. Quilleré, P. Quinton, S. Rajopadhye and T. Risset, Hardware Design Methodology with the Alpha Language, *Forum on Design Languages*, Lyon, France (Sept. 2001).

[23] R. M. Karp, R. E. Miller and S. Winograd, The organization of computations for uniform recurrence equations, *Journal of ACM*, 14 (3) (1967) 563-590.

[24] B. Kienhuis, E. Rijpkema and E. Deprettere, Compaan: Deriving process networks from Matlab for embedded signal processing architectures, *Proceedings of the 8th Int. Workshop on Hardware/Software Codesign, CODES'2000*, San Diego, ACM, (May 2000) 13-17.

[25] L. Lamport, The parallel execution of Do loops, *Communication of ACM*, 17 (2) (1974) 83-93.

[26] B. Louka and M. Tchuenté, Dynamic programming on two-dimensional systolic arrays, *Information Process. Lett.*, 29 (2) (1988) 97-104.

[27] B. Louka and M. Tchuenté, Triangular matrix inversion on systolic arrays, *Parallel Comput.*, 14 (2) (1990) 223-228.

[28] C. Mauras, ALPHA: Un langage équationnel pour la conception et la programmation d'architectures parallèles et synchrones, *Ph.D Thesis, Université de Rennes I, IRISA*, 1990.

[29] D.I. Moldovan, On the analysis and synthesis of VLSI algorithms, *IEEE Trans. Comput.*, C-31 (11), (November 1982) 1121-1126.

[30] J.K. Peir and R. Cytron, Minimum distance:a method for partitioning recurrences for multiprocessors, *IEEE Transactions on Computers*, 38 (8) (August 1989)1203-1211.

[31] P. Quinton, Automatic synthesis of systolic array from uniform recurrence equations, Proceedings of *IEEE 11th Annual International Conference on Computer Architecture*, Ann Arbor, (1984) 208-214.

[32] P. Quinton, S. Rajopadhye and T. Risset, Extension of the ALPHA language to recurrences on sparse periodic domains, *IEEE Int. Conf. on Application Specific Array Processors*, (1996) 391-401.

[33] P. Quinton and V.V. Dongen, The mapping of linear equations on regular arrays, *J. VLSI Signal Process.* 1 (2) (1989) 95-113.

[34] S.V. Rajopadhye, Synthesizing systolic arrays with control signal from recurrence equations, *Distrib. Comput.* 3 (1989) 88-105.

[35] S.V. Rajopadhye, S. Purushothaman, and R.M. Fujimoto, Synthesizing systolic arrays from Recurrences Equations with Linear Dependencies, *Proceedings, Sixth Conference on Foundations of Software Technology and Theoretical Computer Science*, Springer Verlag, LNCS 241, (1986) 488-503.

[36] S. Rao and T. Kailath, Regular iterative algorithms and their implementation on processors arrays. *Proceedings of IEEE*, 76 (4) (1988) 259-269.

[37] I. Sakho and M. Tchuenté, Méthode de conception d'algorithmes parallèles pour réseaux réguliers, *Tech. Sci. Inform.*, 8 (1989) 63-72.

[38] R. Schreiber, S. Aditya, B. R. Rau, V. Kathail, S. Mahlke, S. Abraham and G. Snider, High-Level Synthesis of Nonprogrammable Hardware Accelerators, Proceedings of the IEEE *Int. Conf. on Application Specific Systems, Architectures, and Processors*, ASAP'2000, Boston, Massachusetts, October, 2000

[39] W. Shang and J.A.B. Fortes, Time optimal linear schedules for algorithms with uniform dependencies, *IEEE Trans. on Computers*, 40 (1991) 723-742.

[40] W. Shang and J.A.B. Fortes, On mapping of uniform dependence algorithms into lower dimensional processors arrays, *IEEE Trans. on Parallel and Distributed Systems*, 3 (1992) 350-363.

[41] A. Schrijver, Theory of Linear and Integer Programming, John Wiley and Sons, New York, 1986.

[42] M. Tchuenté, Parallel Computation on regular arrays, *Manchester University Press*, Manchester and Wiley, NY, USA, 1992.

[43] J. Teich and L. Thiele, Partitioning of processor arrays: a piecewise regular approach. *INTEGRATION, The VLSI Journal*, 14 (1993) 297-332.

[44] L. Thiele, Resource constrained scheduling of uniform algorithms. *Journal of VLSI Signal Processing*, 10 (3) (August 1995) 295-310.

[45] L. Thiele and V. Roychowdhury, Systematic design of local processor arrays for numerical algorithms, in: E.F. Deprettere, A.J. Van der Veen (eds.), *Algorithms and Parallel VLSI Architectures*, Vol. A: Tutorials, Elsevier, Amsterdam, (1991) 329-339.

[46] Y. Wong and J.M. Delosme, Transformation of broadcasts into propagations in systolic algorithms, *J. Parallel and Distrib. Comput.*, 14 (1992) 121-145.

[47] Y. Wong and J.M. Delosme, Space-optimal linear processor allocation for systolic arrays synthesis, Proceedings of the *Sixth International Parallel Processing Symposium*, (1992) 275-282.

[48] J.W. Yeh, W.J. Cheng and C.W. Jen, VASS a VLSI array system synthesizer, *J. of VLSI Signal Processing*, 12 (1996) 135-158.