Regular Partitioning for synthesizing fixed-size systolic arrays*

Alain Darte
Laboratoire LIP-IMAG
Ecole Normale Supérieure de Lyon
69364 LYON Cedex 07
e-mail: darte@lip.ens-lyon.fr

April 1991

Abstract

Extending the projection method for the synthesis of systolic arrays, we present a procedure for the design of fixed-size systolic arrays using a technique called "locally sequential globally parallel" (LSGP) partitioning. Our main result, which gives a necessary and sufficient condition to characterize the boxes in which cells can be merged without conflict, is the key to the procedure presented here.

1 Introduction

Recent work has shown that the usual synthesis method for systolic arrays based upon a projection vector and a scheduling vector can be extended to generate systolic implementations on a fixed number of processors. The main idea of all these extensions is to merge many cells into a single processor, so as to compress the array. This step is called partitioning and can take two different forms: the LPGS (locally parallel globally sequential) form and the LSGP (locally sequential globally parallel) form.

The first approach, studied by Moldovan [MF86], is to partition the array into blocks whose size is the number of available processors, say p, and to compute each block one after the other. The different computation

*This work has been supported by the Project C3 of the French Council for Research CNRS, and by the ESPRIT Basic Research Action 3280 "Novel Algorithms for New Real-time VLSI Architectures" (NANA) of the European Economic Community

points in the current block are allocated and scheduled in the p processors, in accordance to the dependence constraints. This method requires small local memories, but a large external buffer is needed to store the data used by the next block.

The second approach, independently studied in [BDD90] and [DD90] is totally different. A virtual array is obtained by the usual method, and it is then partitioned into p blocks of virtual processors, each block being allocated to one physical processor. Of course, the different points allocated to the same processor have to be computed at different times in the array in such a way that they can be sequentially executed by the physical processor. This method permits to synthesize systolic arrays with a fixed number of cells and, as a particular case, permits to improve the efficiency of the cells in a systolic array obtained by the usual projection method. However, the two methods presented in [BDD90] and [DD90] have some weaknesses, especially because they give sufficient conditions for merging cells but not necessary conditions.

In section 2, we discuss several criteria to evaluate partitioning strategies and we show that the methods presented in [BDD90] and [DD90] do not fulfill these "quality" criteria. Then, in section 3, we extend the second method by presenting a new result which leads to an almost optimal partitioning. In section 4, we propose the automatic procedure, deduced from this result, to synthesize systolic arrays, and we exemplify it on the band matrix multiplication problem.

2 Qualities of a good partitioning for the LSGP approach

In the usual synthesis method, introduced in [Qui84] (see also [Kun88], [QR89] and the reference therein), the allocation and timing function are chosen linear. The domain of computation is projected along a vector, called projection vector, onto an hyperplane and this allocation plus the timing function describe the array. If we look at the resulting array more precisely, we can remark that the points projected along the projection vector onto the same cell are evaluated periodically, say with a period of c time units. This gives the intuition that c cells could be merged in a same physical processor. This should allow both to increase the activity of the cells, and to compress an array: indeed, we just have to slow down the algorithm, i.e. to increase c, and then we can compress the array by a factor c. A priori, we

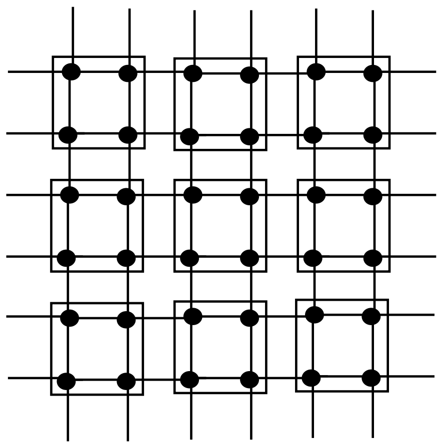

Figure 1: A paving with parallelipipeds with edges parallel to the links.

could merge any c cells in a same physical processor if they never are active at the same time. However, the best way seems to cluster cells belonging to parallelipipeds whose edges are parallel to the projection of dependence vectors, and this, for at least three reasons:

- In general, the array obtained after projection is a regular array with boundaries parallel to certain dependence vectors. Thus, a paving with such parallelipipeds will give a good paving, i.e. will require nearly the smallest number of processors.
- Second, using boxes with edges parallel to the projection of dependence vectors, as a pattern for the partitioning, will reduce the number of external communications, and the number of links.
- Third, merging neighboring cells is better if we want to pipeline many problems on the same array. Indeed, the delay between two problems would be longer if we merge into the same physical processor a cell active at the beginning together with a cell active much later, hence available for a second execution later too.

For example, for a square array, figures 1 and 2 show two different pavings and figures 3 and 4 show the resulting arrays.

In the method presented in [BDD90], $(c-1)$ vectors $(\vec{v}_1, \ldots, \vec{v}_{(c-1)})$ are chosen which indicate that the points $p, p+\vec{v}_1, \ldots, p+\vec{v}_{(c-1)}$ can be merged in the same processor. However, these vectors are found by enumeration

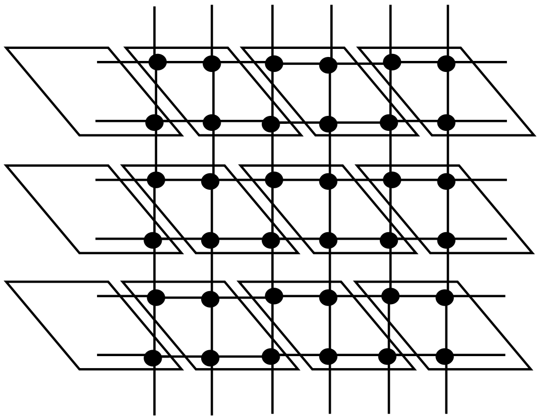

Figure 2: A paving with skewed parallelipipeds.

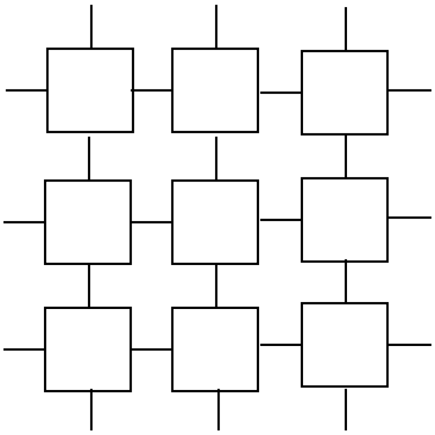

Figure 3: Only 9 cells and 4 links per cell.

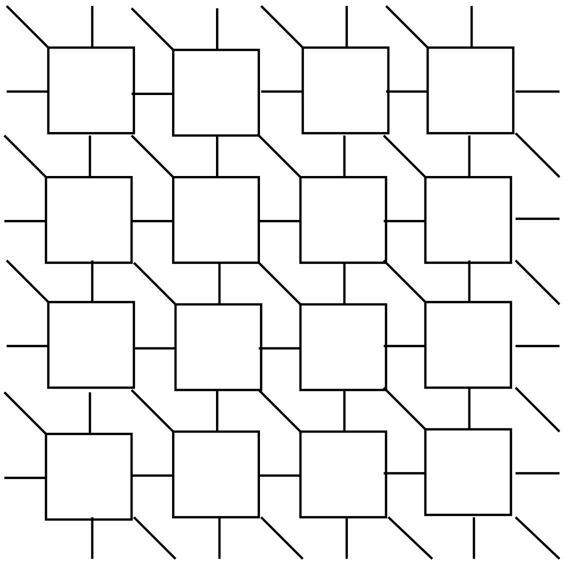

Figure 4: 16 cells and 6 links per cell.

and do not always allow to merge neighboring points. No information about the position of these vectors is given and thus, the qualities of the resulting partitioning are not ensured. The method presented in [DD90] has a similar weakness. It automatically gives a basis in which the points in a parallelipiped of the desired size can be merged, but nothing can be said about this basis. It can be a basis in which the array is very skewed, so the induced paving is not optimal. The main reason of these weaknesses is that nobody has ever proposed a necessary and sufficient condition which characterizes the parallelipipeds inside which all the points are computed at different times. This is the object of the next section.

3 Properties of an array obtained by the projection method

To describe all the properties of the resulting array in a precise framework, some mathematical tools must be introduced. They will allow to select the parallelipipeds that are suitable for a good partitioning.

3.1 Mathematical tools

The computation domain $\mathcal{D}$ is considered as a subset of $\mathcal{Z}^n$. Each point p of the domain will be denoted by the vector $\vec{p}$ of its coordinates. All the vectors used in the following are vectors with integral components and are treated as column vectors. The projection vector is noted $\vec{s}$ and chosen with relatively prime components. The timing function, a linear form, is given by the scalar product $\langle\vec{\tau}|.\rangle$ where $\vec{\tau}$, called the scheduling vector, is a vector with relatively prime components. The d dependence vectors are noted $(\vec{d_1}, \ldots, \vec{d_d})$. $\vec{\tau}$ must satisfy the following conditions:

- $\forall i, 1 \leq i \leq d, \langle\vec{\tau}|\vec{d_i}\rangle > 0$ (dependence constraints)
- $\langle\vec{\tau}|\vec{s}\rangle > 0$ (non conflict in a cell)

Furthermore, because there is a large freedom in the choice of these two vectors, $\vec{s}$ is usually chosen in such a way that the input data are projected on the boundaries of the resulting array and/or that the number of cells is minimized. If we want to compress the array by a factor c, we choose $\vec{\tau}$ among the vectors satisfying $\langle\vec{\tau}|\vec{s}\rangle = c$, as the one which minimizes the total time $\max_{\vec{p}\in\mathcal{D}}(\langle\vec{\tau}|\vec{p}\rangle) - \min_{\vec{p}\in\mathcal{D}}(\langle\vec{\tau}|\vec{p}\rangle)$.

The vector $\vec{s}$ is completed into a unimodular matrix noted S. The vector $\vec{\tau}$ considered as a row vector is completed into a unimodular matrix noted T^{-1} (the inverse of T). Note that completing a vector $\vec{q}$ with relatively prime components into a unimodular matrix is easy: one only needs to compute the right Hermite form of this vector $q = QH$ and the desired matrix is Q. See [DD90] for more details. The domain of computation will be described in two different basis, whose vectors are the column vectors of S (the *space basis*) and T (the *time basis*): remember that if X denote the components of a point in the canonical basis, then its components in the basis formed by the column vectors of a matrix M are given by $X' = M^{-1}X$.

In the space basis, the first component of a point is the component along $\vec{s}$ and the $(n-1)$ other components are the components of the projection of this point, along the first one $\vec{s}$, onto the hyperplane given by the $(n-1)$ other column vectors. Thus, the allocation function is given by the lower submatrix of size $(n-1, n)$ in the matrix S^{-1}. The first component of a point p in the time basis is $\langle\vec{\tau}|\vec{p}\rangle$ and the $(n-1)$ last column vectors of T are orthogonal to $\vec{\tau}$. This means that the $(n-1)$ last column vectors of T form a basis of the hyperplane of the points computed at time 0 and the first one is the translation vector which brings the hyperplane of the points computed at time t onto the hyperplane of the points computed at time $t+1$. Then, we focus on the lower-right submatrix of size $(n-1, n-1)$ in the matrix $S^{-1}T$, which we denote by A (for active cells). Since the column vectors of $S^{-1}T$ are the components of the time basis in the space basis, this matrix A gives all the interesting properties about the activity of the cells in the resulting array, because its column vectors form a basis (in the $(n-1)$-dimensional array) of the submodule of the cells active at the same time. We call such a basis an *activity basis.*

Consider the following example:

$$\vec{s} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad \vec{\tau} = \begin{pmatrix} 9 \\ 4 \\ 3 \end{pmatrix}$$

$\vec{s}$ is completed into:

$$S = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

To show the different steps of the computation of an Hermite form, let us complete by hand the vector $\vec{\tau}$. At each step, the smallest component is chosen and then substracted to the others.

$$\begin{pmatrix} 9 \\ 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \\ 9 \end{pmatrix}$$

$$\begin{pmatrix} 3 \\ 4 \\ 9 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 3 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Thus,

$$\begin{pmatrix} 9 \\ 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 9 & 3 & 1 \\ 4 & 1 & 0 \\ 3 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

We take

$$T^{-1} = \begin{pmatrix} 9 & 4 & 3 \\ 3 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

and

$$T = \begin{pmatrix} 0 & 0 & 1 \\ 1 & -3 & 0 \\ -1 & 4 & -3 \end{pmatrix}$$

This leads to

$$S^{-1}T = \begin{pmatrix} 0 & 0 & 1 \\ 1 & -3 & -1 \\ -1 & 4 & -4 \end{pmatrix} \quad A = \begin{pmatrix} -3 & -1 \\ 4 & -4 \end{pmatrix}$$

3.2 Activity patterns in the array

Remember that $c = \langle \vec{r} | \vec{s} \rangle$ and that A is a matrix of size $(n-1, n-1)$. The determinant of A is the first component of the matrix $T^{-1}S$, i.e. c, the factor by which we want to compress the array. Note that we can obtain other activity basis by carrying out on A, operations such as permuting columns or adding a column to an other. As a particular case, such operations can lead to the $(n-1)!$ left Hermite forms of A, in which we are interested for the following theorems. For simplicity, we say that a box (a parallelipiped) is *valid* if it is a box of volume c in which all the cells are active at different times.

The following theorem gives a sufficient condition for a box to be valid.

Theorem 1 *If M is the matrix of an activity basis and is upper triangular with a positive diagonal $(l_1, \ldots, l_{(n-1)})$ then a box with edges parallel to the axis and of size $(l_1 \times \cdots \times l_{(n-1)})$ is valid.*

proof: Two cells p and q are active at the same time if $\vec{p} - \vec{q}$ is a linear integer combination of the column vectors of M. Furthermore, p and q are in a box with edges parallel to the axis and of size $(l_1 \times \cdots \times l_{(n-1)})$ if $\forall i, -l_i < (\vec{p} - \vec{q})_i < l_i$. Thus, if there are two cells p and q active at the same time in such a box then we have to solve an equation of the form $X = MY$ with $\forall i, -l_i < X_i < l_i$, which leads to $Y = \vec{0}$, and thus $\vec{p} = \vec{q}$. An other proof is given in [DD90].

□

From a theoretical point of view, it is interesting to know that any shape of valid box exist, but such a box requires a suitable basis, which means that, in the canonical basis, it can be a very skewed parallelipiped. This is shown by the following theorem:

Theorem 2 *Given* $(l_1 * \cdots * l_{(n-1)}) = c$ *any decomposition of* c *in integer factors, it is always possible to find a basis of the array so that a box with edges parallel to the axis and of size* $(l_1 \times \cdots \times l_{(n-1)})$ *is valid.*

proof: A constructive proof is given in [DD90] and is based upon the fact that the Smith form of A is $diag(1, 1, \ldots, 1, c)$. As a corollary, compressing along a line is always possible.

□

As said in section 2, we would like to characterize, in a given basis, which boxes with edges parallel to the axis are valid. Note $(\vec{e}_1, \ldots, \vec{e}_{(n-1)})$ the $(n-1)$ vectors of the basis which we are going to work with, and $(a_1, \ldots, a_{(n-1)})$ the activity period along each axis. This means that $a_i\vec{e}_i$ is the shortest vector $\vec{v}$ along the i^{th} axis such as the cells p and $p+\vec{v}$ are simultaneously active. For a given box with edges parallel to the axis, we will note $(l_1, \ldots, l_{(n-1)})$ the length of its edges. We will use the following theorem on finite abelian groups:

Theorem 3 *Let* G *be a finite abelian group. Note* $[g]_q = \{0, g, \ldots, (q-1)g\}$ *with* $g \in G$ *and* $1 < q \leq order(g)$*. If* $S_1, \ldots, S_k$ *are* k *subsets of* G*,* G *is said to be the direct sum of the* S_i*, and we note* $G = S_1 \oplus \cdots \oplus S_k$ *if the mapping* $(g_1, \ldots, g_k) \mapsto g_1 + \cdots + g_k$ *from* $S_1 \times \cdots \times S_k$ *to* G *is one-to-one. If* $G = [g_1]_{k_1} \oplus \cdots \oplus [g_r]_{k_r}$ *then at least one of the* $[g_i]_{k_i}$ *is a subgroup of* G*.*

proof: This result has been proved by Hajós in its works on one of the Minkowski's conjecture, see [Haj42].

□

Lemma 1 *If a box, with edges parallel to the axis, is valid then there exists* i *such as* $l_i = a_i$*.*

proof: Note $\mathcal{O}$ the submodule generated by the column vectors of A and consider the finite abelian group $G = \mathcal{Z}^{n-1}/\mathcal{O}$ of order c. Note $g_1, \ldots, g_{(n-1)}$ the canonical images of $0 + \vec{e}_1, \ldots, 0 + \vec{e}_{(n-1)}$ in G. Take B_0 the box defined as $\{t_1\vec{e}_1 + \cdots + t_{(n-1)}\vec{e}_{(n-1)} \,/\, \forall i,\ 0 \leq t_i < l_i\}$. If B_0 is valid then for all

$x \in \mathcal{Z}^{n-1}$, there exists $a \in \mathcal{O}$ and integers m_i, $0 \leq t_i < l_i$ such that $x = a + m_1\vec{e}_1 + \cdots + m_{(n-1)}\vec{e}_{(n-1)}$ and this decomposition is unique. This is just considering the point in B_0 computed at the same time (modulo c) as x and this means exactly that $G = [g_1]_{l_1} \oplus \cdots \oplus [g_{(n-1)}]_{l_{(n-1)}}$. The previous theorem shows that one of the g_i satisfies $l_i g_i = 0$, i.e. $0 + l_i\vec{e}_i \in \mathcal{O}$. Therefore, $l_i = a_i$.

□

Theorem 4 *A box with edges parallel to the axis and of size $l_1, \ldots, l_{(n-1)}$ is valid if and only if there exists a left Hermite form of A with diagonal $l_1, \ldots, l_{(n-1)}$.*

proof: We have already proved the sufficient condition. Now, consider a valid box of size $l_1, \ldots, l_{(n-1)}$. The previous lemma gives an index i such that $a_i = l_i\vec{e}_i$. Note $\tilde{\mathcal{O}}$ (resp. $\tilde{B}_0$) the projection of $\mathcal{O}$ (resp. B_0) along $\vec{e}_i$ onto $\mathcal{Z}^{(n-2)}$. Obviously, $\tilde{\mathcal{O}}$ is a submodule of $\mathcal{Z}^{(n-2)}$ and $\tilde{B}_0$ a valid box for this submodule, thus there is an activity vector of the form $l_j\vec{e}_j + t\vec{e}_i$. By repeating this process, we obtain $(n-1)$ activity vectors and the matrix H formed by these vectors is triangular with diagonal $l_1, \ldots, l_{(n-1)}$ (by eventually permuting the indices). We just have to show now that these vectors form a basis of the module generated by A. By adding suitable combinations of the columns vectors of H to any vector $x \in \mathcal{O}$, we get $x = H\lambda + a$ where $\forall i$, $0 \leq a_i < l_i$. Because there is only one element of $\mathcal{O}$ in B_0 (which is 0), we have $a = 0$ and thus, x is a linear combination of the column vectors of H. Furthermore, this decomposition is unique because H is triangular. So, H is the matrix of a basis of $\mathcal{O}$, this means that there exists a unimodular matrix Q such that $A = HQ$, and this completes the proof.

□

For the example in section 2, the two left Hermite form of A are

$$H_1 = \begin{pmatrix} 1 & 0 \\ 4 & 16 \end{pmatrix} \quad H_2 = \begin{pmatrix} 1 & 4 \\ 4 & 0 \end{pmatrix}$$

which means that in the basis given by S, the only valid boxes are the boxes of size 1×16 and 4×4, see figure 5 and 6.

4 A procedure for partitioning

We will now present the automatic procedure deduced from the previous results, in order to build S, T and to choose the best box for partitioning.

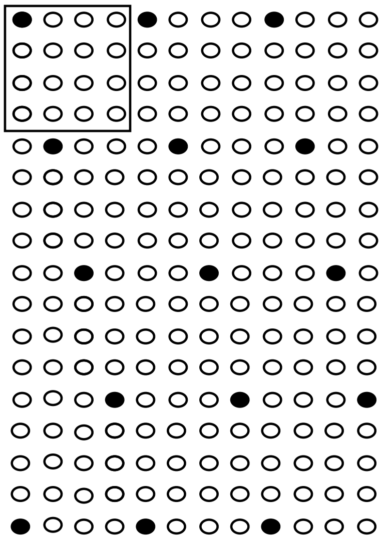

Figure 5: valid box 4×4

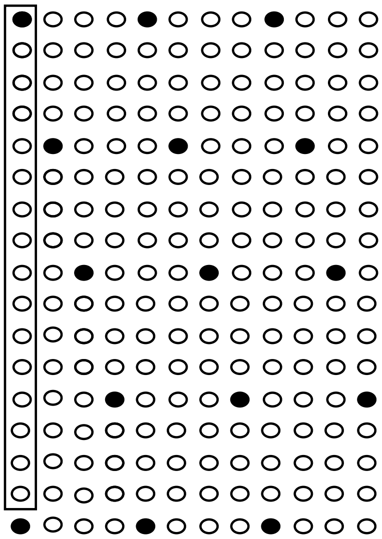

Figure 6: valid box 1×16

We assume that the domain of computation $\mathcal{D}$ is known as well as the dependence vectors $(\vec{d}_1, \ldots, \vec{d}_d)$.

1.Construction of the space basis: Choose a projection direction $\vec{p}$ of the domain with integer components. Compute the right Hermite form of $\vec{p}$, $\vec{p} = SH$. This defines the space matrix S.

2.Selection of the compression factor: Evaluate the number of cells in the virtual array obtained in the space basis S and choose a compression factor c.

3.Construction of the time basis: First select among the vectors satisfying $\langle \vec{\tau} | \vec{s} \rangle = c$, those which minimize the total time $\max_{\vec{p} \in \mathcal{D}} (\langle \vec{\tau} | \vec{p} \rangle) - \min_{\vec{p} \in \mathcal{D}} (\langle \vec{\tau} | \vec{p} \rangle)$. Compute the left Hermite form of $\vec{\tau}$, $\vec{\tau} = QH$. This defines the space matrix T, which is the inverse of the transpose of Q. For each of them, follow the next steps, and finally select the one which minimizes the number of cells and/or the number of links.

4.Construction of an activity basis: Compute $S^{-1}T$ and note A_0 the lower right submatrix of dimension $(n-1)$.

5.Selection of a work basis of the virtual array: Note $\vec{r}_i$ the projection of the i^{th} dependence vector, i.e. $S^{-1}\vec{d}_i$ without considering the first component. Select $\min((n-1), d)$ vectors among these $\vec{r}_i$ and note M the matrix formed by these vectors as column vectors. Compute the right Hermite form of M, $M = QH$ and if H has no null column vectors (which means that the selected vectors were linearly independent), take Q as the basis which we are going to work with. Thus, take as activity basis the matrix $A = Q^{-1}A_0$.

6.Selection of the pattern for partitioning: Compute the $(n-1)!$ left Hermite forms of A, which gives all the possible valid boxes in the basis Q. Select the box which allows the paving with the fewest number of cells. Reducing the number of external communications can be a criteria, too. This gives a size $l_1 \times \cdots \times l_{(n-1)}$.

7.Allocation function and timing function: Note M the matrix of size $(n-1)n$ formed by the $(n-1)$ last rows of S^{-1}. If Q is the selected basis and $l_1 \times \cdots \times l_{(n-1)}$ the selected size for the pattern, note $D = diag(1/l_1, \ldots, 1/l_{(n-1)})$. Then the allocation function without compressing is given by the matrix $C = Q^{-1}M$ and the final allocation fuction is given by DC. A point p is computed in the cell $\lfloor DCp \rfloor$.

The notation $\lfloor p \rfloor$ means the vector with components $\lfloor p_i \rfloor$. The timing function is, as usually, given by $\langle \vec{\tau} | . \rangle$.

Note that the complexity of the procedure only depends upon n and d which are very small in general ($n = 3$ for example). Let us examine an example, an array for the band matrix multiplication problem. To simplify, we will follow the different steps, but only for one vector $\vec{\tau}$. $\mathcal{D}$ is a subset of the domain for the matrix multiplication problem and is usually projected along

$$\vec{s} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

in such a way that the size of the resulting virtual array only depends upon the size of the bands and not upon the size of the matrices which have to be multiplied. If we choose $c = 16$, the matrices S, T and A are those given at the end of the section 2. The dependence vectors are:

$$\vec{d_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \vec{d_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \vec{d_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Thus, after projection they are:

$$\vec{r_1} = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \quad \vec{r_2} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \vec{r_3} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The different work basis are:

$$Q_1 = \begin{pmatrix} -1 & 1 \\ -1 & 0 \end{pmatrix} \quad Q_2 = \begin{pmatrix} -1 & 0 \\ -1 & 1 \end{pmatrix} \quad Q_3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

with the associated activity basis:

$$A_1 = \begin{pmatrix} -4 & 4 \\ -7 & 3 \end{pmatrix} \quad A_2 = \begin{pmatrix} 3 & 1 \\ 7 & -3 \end{pmatrix} \quad A_3 = \begin{pmatrix} -3 & -1 \\ 4 & -4 \end{pmatrix}$$

The different Hermite forms are:

$$H_{1,1} = \begin{pmatrix} 4 & 0 \\ 3 & 4 \end{pmatrix} \quad H_{1,2} = \begin{pmatrix} 12 & 16 \\ 1 & 0 \end{pmatrix}$$

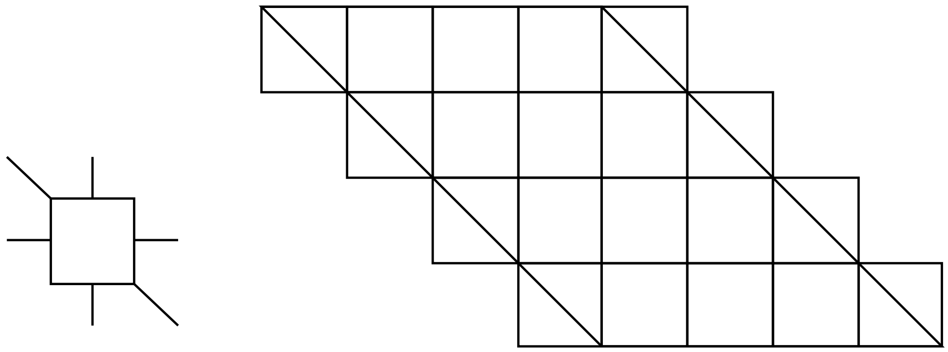

Figure 7: links and paving 1 in basis 1

$$H_{2,1} = \begin{pmatrix} 1 & 0 \\ 13 & 16 \end{pmatrix} \quad H_{2,2} = \begin{pmatrix} 5 & 16 \\ 1 & 0 \end{pmatrix}$$

$$H_{3,1} = \begin{pmatrix} 1 & 0 \\ 4 & 16 \end{pmatrix} \quad H_{3,2} = \begin{pmatrix} 1 & 4 \\ 4 & 0 \end{pmatrix}$$

The different paving are shown figures 7 to 12 for an example for which the projection of the domain gives a square of size 32×32.

The best array is the last one. It has the fewest number of cells and 6 links like the others.

5 Conclusion

We have shown that the problem of LSGP partitioning can be well described by the use of unimodular matrices, Hermite form and Smith form: the matrices of the space basis, the time basis and the activity basis contain all the useful informations about the behavior of the array and can be computed very easily and automatically. This new approach for describing systolic arrays seems to be very powerful. By using these mathematical tools, we have shown how to find automatically the allocation function by just completing the projection vector into a unimodular matrix. We have proved that any shape of valid box could be found, in a suitable basis. Above all, we have given a necessary and sufficient condition to characterize the boxes of maximal volume in which cells could be merged without collision, which is very simple to express in terms of Hermite forms. The examples show the importance of good partitioning strategies to avoid bad pavings with too many

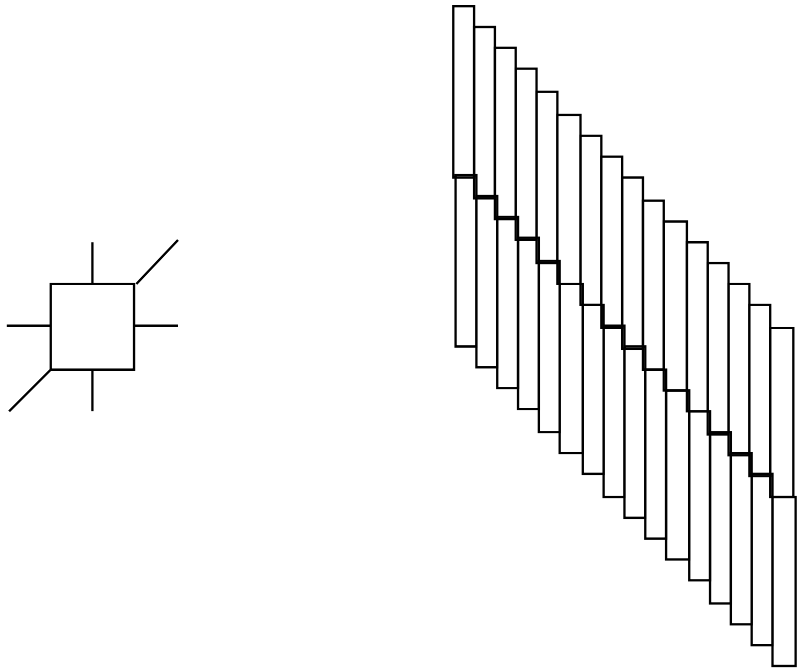

Figure 8: links and paving 1 in basis 2

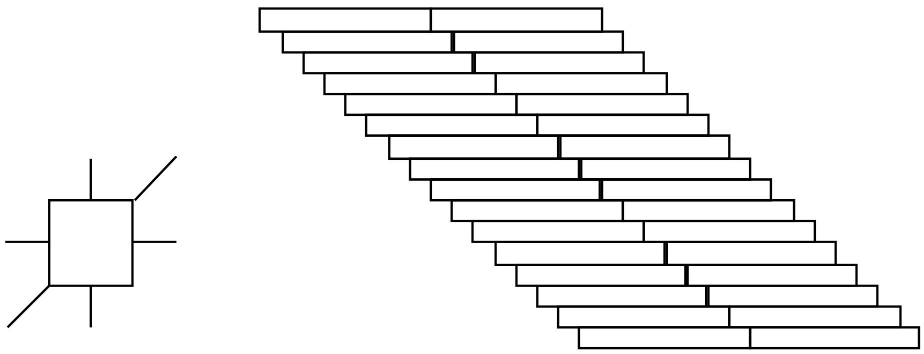

Figure 9: links and paving 2 in basis 1

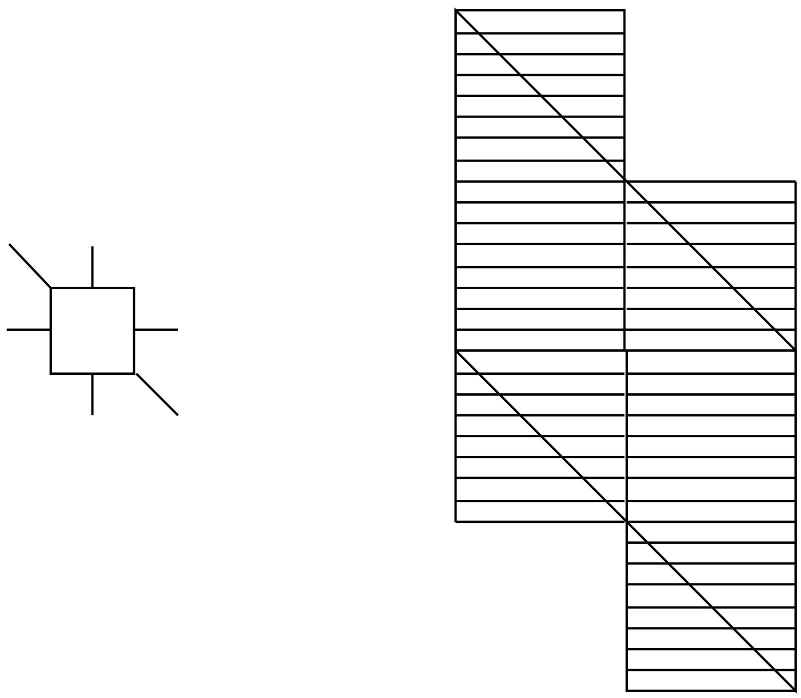

Figure 10: links and paving 2 in basis 2

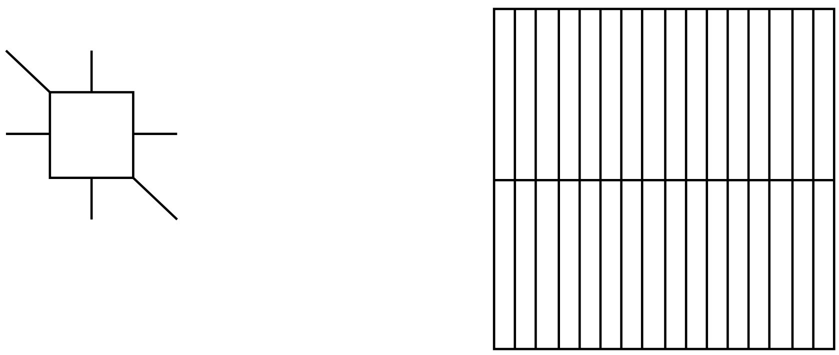

Figure 11: links and paving 1 in basis 3

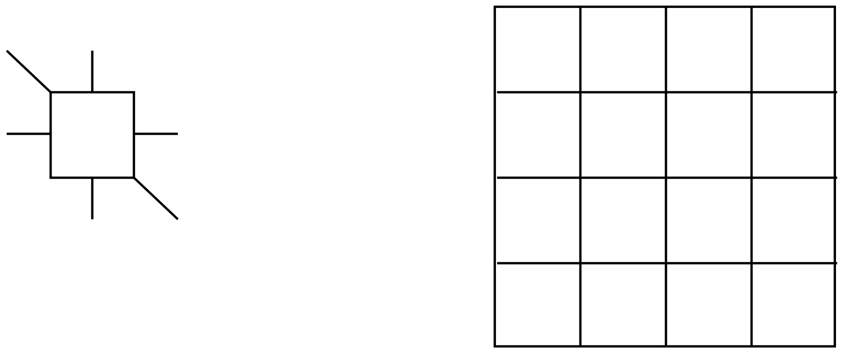

Figure 12: links and paving 2 in basis 3

links or too many cells, thus these results had to be found to develop a good partitioning procedure which we wanted to be automatic and without long enumeration.

It has been shown in [DD90] that this method could be extended to project the computation domain onto smaller dimensional spaces but the extension was not optimal in time. Some work has still to be done to give a partitioning procedure for linear arrays, for example.

References

[BDD90] J. B., E.F. Depreterre, and P. Dewilde. A design methodology for fixed-size systolic arrays. In S.Y. Kung and E. Swartzlander, editors, *International Conference on application Specific Array Processing*, pages 591–602, Princeton, New Jersey, September 1990. IEEE Computer Society.

[DD90] A. Darte and J.M. Delosme. Partitioning for array processors. Technical Report 90-23, LIP, ENS-Lyon, 46 allee d'Italie, 69364 Lyon cedex 07, France, October 1990.

[Haj42] G. Hajos. Uber einfache und mehrfache bedeckung des n-dimensionalen raumes mit einem wurfelgitter. *Math. Zeitschrift*, 47:427–467, 1942.

[Kun88] S.Y. Kung. *VLSI array processors.* Prentice-Hall, 1988.

[MF86] D.I. Moldovan and J.A.B Fortes. Partitioning and mapping of algorithms into fixed size systolic arrays. *IEEE Trans. on Computers*, 35(1):1–12, January 1986.

[QR89] P. Quinton and Y. Robert. *Algorithmes et architectures systoliques*. Masson, 1989.

[Qui84] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *Proc IEEE 11-th Int. Sym. on Computer Architecture*, pages 208–214, Ann Arbor, MI, USA, 1984.