

Practical Solutions for Counting Scalars and Dependences in ATOMIUM— A Memory Management System for Multidimensional Signal Processing

Florin Balasa, *Member, IEEE*, Francky Catthoor, *Member, IEEE*, and Hugo J. De Man, *Fellow, IEEE*

Abstract—Image and video processing applications involve large multidimensional signals which have to be stored in memory modules. In application-specific architectures for real-time multidimensional signal processing, a significant cost in terms of chip area and power consumption is due to these background memory units. The multidimensional signals are usually modeled in behavioral descriptions with array variables. In the algorithmic specifications of our target applications, many of the array references cover large amounts of scalars. Therefore, the efficient handling of array references in the specifications for image and video processing is crucial for obtaining low cost memory allocation solutions.

This paper addresses a central problem which arises when handling the array variables in behavioral specifications: the computation of the number of scalars covered by an array reference. This problem is closely related to the computation of dependences in data-flow analysis. The novel algorithms proposed in this paper are embedded in the ATOMIUM environment—a memory management system for multidimensional signal processing.

Index Terms—Data dependence, data-flow analysis, digital signal processing (DSP), linearly bounded lattice, polytope.

I. INTRODUCTION

IN application-specific architectures for real-time multidimensional signal processing, memory modules are responsible for a significant cost in terms of chip and/or board area and of power consumption. This is due to the large multidimensional signals involved by most of the image and video processing applications. The behavioral specifications of these applications model the signals by array variables, with many references of the indexed variables covering large amounts of scalars. The efficient handling of array references in procedural or nonprocedural specifications for image and

video processing applications is crucial for obtaining low cost solutions for the memory allocation task.

The computation of the scalars covered by the array reference $A[j][i + N + 1]$, defined within the second loop nest¹ in the illustrative Silage [10] example of Fig. 1(a), requires as operands signals covered by the array references $A[j][i + N]$ ($j < i$) and $A[j][i + N]$ ($j \geq i$), residing in the regions indicated in Fig. 1(b).

It must be noticed that only a small part of the signals A produced in the first loop nest are still necessary. These are the signals $A[j][N]$ ($0 \leq j \leq N - 1$, j even) belonging both to the array reference $A[j][i + 1]$ from the first loop nest and to the operand $A[j][i + N]$ ($j \geq i$) from the second loop nest. The rest of the signals A produced in the first loop nest, *independent of their internal order of computation*, are no longer necessary. The storage allocated for them can be freed before starting the execution of the second loop nest.

This example shows that data- and control-flows should influence the memory allocation solutions when the specifications are nonprocedural. Moreover, the amount of storage locations which is necessary in order to produce the signals in a certain loop nest, or which can be freed after the execution of the loop nest, depends on the number of scalars covered by the array references involved in the computation.

Approaching memory estimation/allocation issues from behavioral specifications containing indexed variables has recently gained attention in the digital signal processing (DSP) community. The PHIDEO system [12] employs a hierarchical stream model in order to handle multidimensional signals with complex affine indexes. However, the nested loops have only constant boundaries.

In the MeSA system [19], the array variables are clustered in memory modules, a strategy which leads to better allocation solutions than mapping each array to a separate memory or grouping all arrays into a single memory. The main drawback of the model is that life-time information is not exploited (as suggested by the example above) in order to reduce the storage requirements. The problem of computing the size of array references is, therefore, not needed, as memory is allocated for

Manuscript received April 19, 1995; revised November 19, 1996. This paper was recommended by Associate Editor, M. McFarland. This work was supported in part by the ESPRIT Basic Research Action 3281 (NANA II) Project of the European Community.

F. Balasa was with the Interuniversity Micro-Electronics Center (IMEC), Kapeldreef 75, B-3001 Leuven, Belgium. He is now with Rockwell Semiconductor Systems, Newport Beach, CA 92660 USA and the University of California, Irvine, CA 92664 USA.

F. Catthoor and H. J. De Man are with the Interuniversity Micro-Electronics Center (IMEC), Kapeldreef 75, B-3001 Leuven, Belgium.

Publisher Item Identifier S 0278-0070(97)03567-7.

¹The syntax of a loop statement is: (*iterator: lower_bound ..upper_bound .. step*);. The default value of the increment step is one.

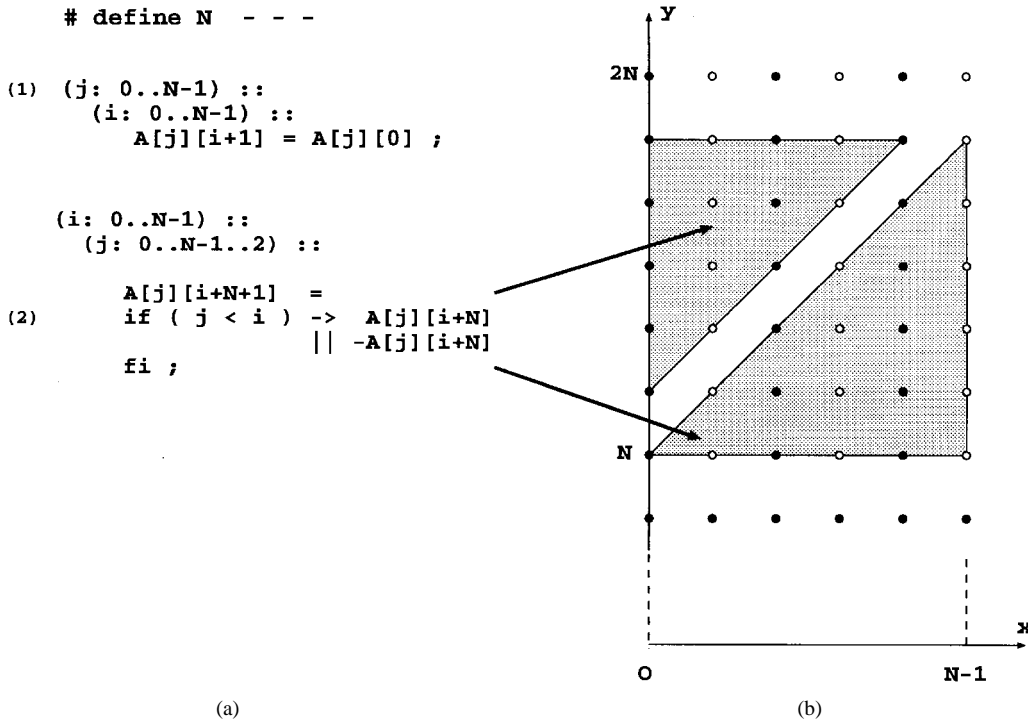


Fig. 1. Illustrative Silage code.

the entire array signals, a very expensive solution especially for image processing applications.

In more recent research, data dependence information provided by the Omega test [18] has also been applied in memory size estimation [25]. The approach is well suited for specifications where the loops have constant boundaries. For other applications, approximations of the loop boundaries by their extreme values are employed. This can lead to expensive results in terms of storage requirements, as justified by the example from Section II.

In [24], a polyhedral model for indexed signals is used to modify the loop hierarchy and the sequence of execution of the source code in order to optimize the storage requirements. In this approach, a crude but fast approximation is made for the size of array signals.

This paper addresses the problem of computing the number of scalars covered by array references with affine indexes, affine-type conditions, and in the scope of nested loops having as boundaries affine functions of loop iterators. This research has been carried out in the *context* of memory management for multidimensional signal processing systems which is not the *topic* of this work. Our memory estimation and allocation strategy has been presented [1], [2], and, therefore, it is beyond the scope of this paper.

Section II will present more formally the central problem tackled in this work. Afterwards, the organization of the paper will be described. In the sequel, matrices and vectors are denoted with bold characters, matrices being in capitals.

II. THE PROBLEM OF COMPUTING THE NUMBER OF SCALARS ADDRESSED BY AN ARRAY REFERENCE

Definitions: A *polyhedron* is a set of points $P \subset \mathbb{R}^n$ satisfying a finite set of linear inequalities: $P = \{\mathbf{x} \in \mathbb{R}^n \mid$

$\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}\}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. If P is a bounded set, then P is called a *polytope*. If $\mathbf{x} \in \mathbb{Z}^n$, then P is called an *integral polyhedron/polytope*. The set $\{\mathbf{y} \in \mathbb{R}^m \mid \mathbf{y} = \mathbf{A}\mathbf{x}, \mathbf{x} \in \mathbb{Z}^n\}$ is called the *lattice* generated by the columns of matrix \mathbf{A} .

Each array reference $A[x_1(i_1, \dots, i_n)] \dots [x_m(i_1, \dots, i_n)]$ of an m -dimensional signal A , in the scope of a nest of n loops having the iterators i_1, \dots, i_n , is characterized by an *iterator space* and an *index space*. The iterator space signifies the set of all iterator vectors $\mathbf{i} = (i_1, \dots, i_n) \in \mathbb{Z}^n$ in the scope of the array reference. The index space is the set of all index vectors $\mathbf{x} = (x_1, \dots, x_m) \in \mathbb{R}^m$ of the array reference.

If the loop boundaries are affine mappings with integer coefficients of the surrounding loop iterators, the increment steps of the loops² are ± 1 , and the conditions in the scope of the array reference are relational and/or logical operations between affine mappings of the loop iterators,³ then the iterator space can be represented by one or several disjoint integral (iterator) polytopes $\mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}$, where $\mathbf{A} \in \mathbb{Z}^{(2n+c) \times n}$ and $\mathbf{b} \in \mathbb{Z}^{2n+c}$. The first $2n$ linear inequalities are derived from the loop boundaries, and the last c inequalities are derived from the control-flow conditions.⁴ The size of the iterator space is $\text{Card}\{\mathbf{i} \in \mathbb{Z}^n \mid \mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}\}$.

If, in addition, the indexes of an array reference are affine mappings with integer coefficients of the loop iterators, the index space consists of one or several *linearly bounded*⁵

²This condition can be relaxed, as shown further.

³The polyhedral representation of the iterator space is still valid if the array reference scope also contains data-dependent (but iterator independent) conditions.

⁴This representation of the polytope $\mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}$ is usually not minimal (see Section III-B).

⁵As the definition domain of the mappings is not \mathbb{Z}^n but a polytope, which is *bounded* by hyperplanes characterized by *linear* equations.

lattices (LBL) [23], the image of a vectorial affine function over the iterator polytopes

$$\{\mathbf{x} = \mathbf{T} \cdot \mathbf{i} + \mathbf{u} \mid \mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}, \mathbf{i} \in \mathbf{Z}^n\} \quad (1)$$

where $\mathbf{x} \in \mathbf{Z}^m$ is the index (coordinate) vector of the m -dimensional signal. The affine function is characterized by $\mathbf{T} \in \mathbf{Z}^{m \times n}$ and $\mathbf{u} \in \mathbf{Z}^m$. The size of the index space of an array reference is, therefore, $\text{Card}\{\mathbf{x} \in \mathbf{Z}^m \mid \mathbf{x} = \mathbf{T} \cdot \mathbf{i} + \mathbf{u}, \mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}, \mathbf{i} \in \mathbf{Z}^n\}$. The number of scalars addressed by the array reference is obviously the size of its index space.

Example: The index space of the operand $A[j][0]$ in loop nest (1) from the Silage code in Fig. 1(a) is represented by

$$\left\{ \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} [j] + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \mid \begin{bmatrix} 1 \\ -1 \end{bmatrix} [j] \geq \begin{bmatrix} 0 \\ -N+1 \end{bmatrix} \right\}.$$

In general, the index space may be a set of linearly bounded lattices, e.g., a conditional instruction *if* ($i \neq j$) determines two LBL's for the array references within the scope of the condition with one corresponding to $i \geq j + 1$ and another corresponding to $i \leq j - 1$. As the iterator space of an array reference can be decomposed into a set of disjoint polytopes and the index space into disjoint linearly bounded lattices [3], it is assumed in the sequel, without any loss of generality, that each iterator space is represented by a single polytope, and correspondingly, each index space is represented by a single linearly bounded lattice.

Several memory-related research works (e.g., estimating the amount of necessary storage [25] or tackling the problem of multiport memory allocation subject to bandwidth constraints [12]) assume that all loops in the specifications have constant boundaries. If this is not the case, approximations by taking the extreme values of the boundaries are considered.

However, such approximations can significantly increase the amount of necessary storage in multidimensional signal processing. Employing the terminology introduced above, the number of scalars addressed by an array reference is approximated by the size of the hypercube encompassing the iterator space of that array reference. This approximation is good only if, in addition, the loops in the array scope have unitary steps, if there are no iterator-dependent conditions in the scope, and if the affine mapping $\mathbf{T}\mathbf{i} + \mathbf{u}$ in (1) is injective. When doing the approximation mentioned above, the number of scalars is associated to the size of the *iterator space* rather than to the size of the *index space*. But, in many cases, the size of the index space can be significantly lower.

Example:

$$(i : 8..15) :: (j : i - 8..i + 8..2) :: \dots A[i + j] \dots$$

As iterator j takes values between zero and 23, the hypercube encompassing the iterator space is $[8, 15] \times [0, 23]$, and it contains 192 points. The size of the index space corresponding to the array reference $A[i + j]$ is, however, 16, therefore, 12 times less than the approximation. The error is due both to the incorrect evaluation of the iterator space size, which contains

72 points rather than 192 and to the fact that the affine mapping $f(i, j) = i + j$ is not injective [e.g., signal $A[12]$ is addressed by several pairs of iterators (i, j) : (8, 4), (9, 3), (10, 2)].

Concluding, for the general affine case, the approximation mentioned above may be very crude, being improper to use for memory estimation/allocation as it can lead to an exaggerated number of storage locations. Section IV will present correct and efficient practical solutions to the problem of index space size determination.

It must be emphasized that enumerative techniques can always be applied to compute the number of scalars in an array reference. These approaches are obviously simple and extremely efficient for array references with “small” iterator spaces. However, in image and video processing applications, most of the array references are characterized by huge iterator spaces. An enumerative technique, although very simple, will be too computationally expensive to use in such applications (see Table I).

It must be mentioned also that loops having increment steps different from one

$$(i : m..M \text{ .. Step}) :: \dots A[f(i)] \dots$$

can be easily “normalized” with the affine transformation $i = i' \cdot \text{Step} + m$, thus, being equivalent to

$$\left(i' : 0.. \left\lfloor \frac{M - m}{\text{Step}} \right\rfloor \right) :: \dots A[f(i')] \dots$$

For instance, the nest of loop in the example above is equivalent to

$$(i : 8..15) :: (j : 0..8) :: \dots A[2 * i + 2 * j - 8] \dots$$

It is assumed along this paper that all loops have been “normalized” in a preprocessing phase. Loop boundaries containing the *floor* $\lfloor \cdot \rfloor$ and *ceiling* $\lceil \cdot \rceil$ functions create no special problem, as the routines of our system are designed to take into account only the points with integer coordinates.

Section III thoroughly presents a practical solution for computing the size of the iterator space of an array reference with affine indexes, within nested loops having affine boundaries and affine-type conditions. Afterwards, Section IV will present two exact numerical solutions for the major problem addressed in this work: the computation of the size of the index space. The first solution is based on an appropriate modification of the affine mapping. A second solution to the problem is based on the effective construction of the index space of an array reference. Section V deals with two other related problems encountered in the ATOMIUM system, a memory management system for multidimensional signal processing [17]: detection of the equivalence of array references and data-dependence issues between array references. Examples illustrating the novel techniques, along with comparative implementation results, will be presented in Section VI, followed by the conclusions of this work in Section VII.

III. COMPUTING THE SIZE OF THE ITERATOR SPACE

As shown in Section II, the iterator space of an array reference, in the scope of affine-type conditions and nested

loops having affine boundaries, consists of a set of integral polytopes.

The computation of the number of points inside a polytope, having integer coordinates, has been tackled long ago for the three-dimensional (3-D) case [15], when the importance of Dedekind sums was revealed in this context. Recently, Dyer has proven that Dedekind sums can be computed in polynomial time, employing an algorithm similar to Euclid's [8]. The outcome is that also our problem can be solved in polynomial time up to the four-dimensional (4-D) case. However, these results appear to be of rather theoretical interest. First, Dyer's algorithm requires a preprocessing step, a decomposition of a special form of the given polytope. Although the decomposition can be achieved in polynomial time, it can affect the practical efficiency of the method. Second, it is not clear whether Dyer's algorithm, based on Dedekind sums, can be extended to higher dimensions. In addition, there is no polynomial-time algorithm available for evaluating Dedekind sums of higher order.

Recent work in the field of data-flow analysis and code parallelization also focused on polytope size determination. Tawbi introduced an algorithm of exponential complexity for counting the number of iterations in nested loops with parametrizable bounds. However, the polytopes have restricted representations in this case [22].

Tackling a related problem, Eisenbeis *et al.* proposed a "fast" but still exponential method for counting the integer solutions of linear systems of equations, when variables have constant bounds [9]. Employing rational fractions of polynomials, the approach is used to determine the number of dependences between array references but only within the scope of loops having constant boundaries. Unfortunately, the method is not able to handle the case of loop nests with affine boundaries, and furthermore, the practical efficiency of the approach degrades quickly with the number of variables.

As accurate solutions for evaluating the size of array references and the number of dependences in complex affine specifications were both needed in the ATOMIUM system [17], a general framework based on the well-known Fourier–Motzkin pairwise elimination (e.g., [6]) has been developed. As this latter technique for reducing the number of variables in a system of linear inequalities is known to be "quite time consuming" [4] for problems in many variables, Sections III-A and III-B will mainly deal with practical improvements which enhance the speed of the approach.

A. Practical Improvements for Efficient Size Computation of Integral Polytopes

The basic idea of the Fourier–Motzkin elimination, briefly described below for the sake of consistency, represents actually the direct way of solving by hand systems of linear inequalities.

A variable, say x_n , may be eliminated from a given system of inequalities

$$\sum_{j=1}^n a_{ij}x_j \geq b_i, \quad i = 1, \dots, m \quad (2)$$

partitioning (2) into three sets of inequalities according to whether the coefficients of x_n are positive, negative, or zero

$$\begin{cases} x_n \geq D_1(\bar{x}) \\ \vdots \\ x_n \geq D_p(\bar{x}) \end{cases} \quad \begin{cases} x_n \leq E_1(\bar{x}) \\ \vdots \\ x_n \leq E_q(\bar{x}) \end{cases} \quad \begin{cases} 0 \leq F_1(\bar{x}) \\ \vdots \\ 0 \leq F_r(\bar{x}) \end{cases} \quad (3)$$

where $D_i(\bar{x}), E_j(\bar{x}), F_k(\bar{x})$ are affine functions of $\bar{x} = (x_1, \dots, x_{n-1})$. The given system (2) may be solved by first solving the *reduced* system (with one variable less)

$$\begin{cases} D_i(\bar{x}) \leq E_j(\bar{x}) & i = 1, \dots, p \quad j = 1, \dots, q \\ 0 \leq F_k(\bar{x}) & k = 1, \dots, r \end{cases} \quad (4)$$

and then finding the values of x_n satisfying $\max_i D_i(\bar{x}) \leq x_n \leq \min_j E_j(\bar{x})$. If $p = 0$ or $q = 0$, then the range of x_n is unbounded, and the existence of an \bar{x} satisfying (4) entails an infinite number of solutions for system (2). In this case, system (2) does not represent a polytope, as the polyhedron is not bounded.

As (4) is also a system of linear inequalities with one variable less, the elimination procedure can be applied to x_{n-1} , and so on until all but a single variable, say x_1 , are gone. The original system (2) is solvable if and only if the final system obtained by the successive elimination of $n - 1$ variables has a solution. The method is usually employed to decide the (in)consistency of a system of linear inequalities.⁶

The routine *CountPolytopePoints*(\mathbf{A}, \mathbf{b}) is described below, denoting the columns of matrix \mathbf{A} by subscripted vectors: $\mathbf{A} = [\mathbf{a}_1 \ \mathbf{a}_2 \ \dots]$. The main idea of the algorithm [1] is the following. The number of points of integer coordinates inside the given n -dimensional polytope $\mathbf{A}\mathbf{x} \geq \mathbf{b}$, with the first coordinate fixed x_1^0 , is equal to the number of points inside the polytope $\mathbf{a}_2x_2 + \mathbf{a}_3x_3 + \dots \geq \mathbf{b} - \mathbf{a}_1x_1^0$ which has one dimension less. The required result is obtained by accumulating this number over the entire discrete range of x_1 . This range is determined by the Fourier–Motzkin method, when x_1 is the last eliminated variable.

The emptiness of an integral polytope can be checked similarly: $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ is not empty if and only if at least one of the polytopes $\mathbf{a}_2x_2 + \mathbf{a}_3x_3 + \dots \geq \mathbf{b} - \mathbf{a}_1x_1^0$ is not empty.

The worst-case time complexity of the algorithm *CountPolytopePoints* is clearly exponential. This results both from the worst-case increase rate of the number of inequalities during the pairwise elimination and from the recursivity of the algorithm. For instance, the Fourier–Motzkin procedure applied to the system (2) could yield in an unfavorable case $(m/2)^2$ inequalities after eliminating the first variable and, eventually, could yield $4(m/4)^{2^m}$ inequalities after eliminating the last variable.

In order to obtain an acceptable efficiency of the algorithm (see Section VI), several enhancing techniques presented in the sequel are employed in our current implementation.

- 1) The algorithm *CountPolytopePoints*(\mathbf{A}, \mathbf{b}) is not applied directly on the iterator polytope $\mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}$. It must

⁶That is, the existence (or not) of at least one real solution.

be noticed that linear inequalities are derived from the constraints on loop boundaries and control-flow conditions. In practice (e.g., some loop boundaries are often constant), the system of inequalities may have a block decomposition of the form

$$\begin{bmatrix} \mathbf{A}_1 & \vdots & \mathbf{0} & \vdots & \mathbf{0} \\ \dots & & \dots & & \dots \\ \mathbf{0} & \vdots & \ddots & \vdots & \mathbf{0} \\ \dots & & \dots & & \dots \\ \mathbf{0} & \vdots & \mathbf{0} & \vdots & \mathbf{A}_s \end{bmatrix} \mathbf{i} \geq \begin{bmatrix} \mathbf{b}_1 \\ \dots \\ \vdots \\ \dots \\ \mathbf{b}_s \end{bmatrix}$$

after an eventual reordering of the iterators. Therefore, the routine *CountPolytopePoints* is usually applied on *reduced* constraint matrices $\mathbf{A}_k \cdot \mathbf{i} \geq \mathbf{b}_k$, $k = 1, \dots, s$, the results being multiplied at the end. In particular, the computation of the iterator space size for an array reference inside a nest of loops with constant boundaries is reduced to a simple multiplication of the iterator ranges.

- 2) If the iterator space representation also contains equalities, then the dimensions of the initial system $\mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}$ is reduced (similar to [18] and [9]). An equality is solved for one variable which is substituted afterwards in the other relations. The process is continued till the total elimination of the equalities. The number of variables decreases correspondingly.

The following techniques aim to decrease the number of inequalities during the Fourier–Motzkin elimination.

- 3) As shown above, the elimination of the variable x_n from system (2) generates a new system (4) of $pq + r$ inequalities, where p, q , and r are the numbers of positive, negative, and zero coefficients of x_n in the initial system (2), respectively. The order of variable elimination is important for keeping the number of inequalities as low as possible. As a consequence, a reasonably good heuristic is to select the next variable to be eliminated among those for which the expression $pq + r$ is minimum. The only difference in the routines presented above is that the Fourier–Motzkin elimination will yield the range of some element x_k of \mathbf{x} (rather than the range of x_1 , the first element), and the modifications are straightforward.
- 4) By numbering distinctly the initial inequalities and the newly generated ones during the pairwise elimination process, the set of initial inequalities from which a certain inequality is derived can be easily determined. If after the elimination of t variables, an inequality is derived from $t + 2$ initial inequalities or more, then the inequality is redundant as proven in [7]. It has to be emphasized that the condition above is only sufficient, and, therefore, not all the redundant inequalities are detected and eliminated with this simple technique.

As a consequence of the Farkas Lemma (e.g., [14]), an inequality is redundant if and only if it is a nonnegative combination of the other inequalities. After each variable

elimination, it is possible to detect *all* the redundant inequalities. However, this latter technique, explained in Section III-B, proves to be expensive when it is embedded in the Fourier–Motzkin elimination (see Section VI).

B. Minimizing the Representation of Polytopes

The procedure employed in the ATOMIUM system [17] for minimizing systems of linear inequalities stems from the polyhedral library developed at IRISA, France [26]. The approach is based on the *double description of polyhedra*.

A *vertex* of a set \mathcal{S} is a point in \mathcal{S} which cannot be expressed as a convex combination of any other distinct points in \mathcal{S} . A vector \mathbf{r} is a *ray* of \mathcal{S} , if $\mathbf{x} \in \mathcal{S}$ implies that $\mathbf{x} + \alpha \mathbf{r} \in \mathcal{S}$, $\forall \alpha \geq 0$. When the same property holds for all α , then \mathbf{r} is a *line* or *bidirectional ray*. A ray is *extreme* if it cannot be expressed as a positive combination of any other two distinct rays.

In 1896, Minkowski showed that any polyhedron defined by $\mathcal{P} = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{C}\mathbf{x} \geq \mathbf{d}\}$ has an equivalent dual parametric representation [21, sec. 8.9]

$$\mathcal{P} = \left\{ \mathbf{x} \mid \mathbf{x} = \mathbf{V}\nu + \mathbf{L}\lambda + \mathbf{R}\mu, \mu, \nu \geq 0, \sum \nu_i = 1 \right\}$$

characterized by a convex combination of vertices (the columns of matrix \mathbf{V}), a linear combination of lines (the columns of matrix \mathbf{L}), and a positive combination of extreme rays (the columns of matrix \mathbf{R}). In 1936, Motzkin proved the *Decomposition Theorem for Polyhedra*, according to which any polyhedron can be *uniquely* decomposed into a polytope represented parametrically by a convex combination of vertices $\mathcal{V} = \{\mathbf{x} = \mathbf{V}\nu, \nu \geq 0, \sum \nu_i = 1\}$, and a *polyhedral cone*, i.e., a polyhedron having only a single vertex.⁷ In its turn, the cone can be partitioned into a *lineality space* $\mathcal{L} = \{\mathbf{x} = \mathbf{L}\lambda, \lambda \in \mathbb{R}\}$, the largest linear space contained in the cone, and a *pointed cone* $\mathcal{R} = \{\mathbf{x} = \mathbf{R}\mu, \mu \geq 0\}$, represented parametrically by a positive combination of extreme rays [21, ch. 8].

An algorithm to compute the dual representations of a polyhedron \mathcal{P} has been proposed in [16]. Given \mathbf{A} , \mathbf{b} , \mathbf{C} , and \mathbf{d} , it determines the matrices \mathbf{V} , \mathbf{L} , and \mathbf{R} and reciprocally, has been proposed in [16]. A similar algorithm has been also implemented in the polyhedral library of IRISA [26].

The elimination of redundant inequalities in (2) or in any of the reduced systems can be achieved by computing the double representation of the respective inequality set.

- 1) By selecting a basis of the lineality space, the redundant equalities can be detected and eliminated as the system may contain also equalities embedded, and these are generating the lineality space.
- 2) The irredundant inequalities must be saturated⁸ by at least n extreme vertices/rays, where n is the space dimension.

⁷A polyhedral cone, having the vertex at the origin, has the implicit description of the form $\{\mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{0}, \mathbf{C}\mathbf{x} \geq \mathbf{0}\}$ and a parametrical representation $\{\mathbf{x} \mid \mathbf{x} = \mathbf{L}\lambda + \mathbf{R}\mu, \mu \geq 0\}$.

⁸A ray \mathbf{r} saturates the inequality $\mathbf{a}^T \mathbf{x} \geq 0$ when $\mathbf{a}^T \mathbf{r} = 0$.

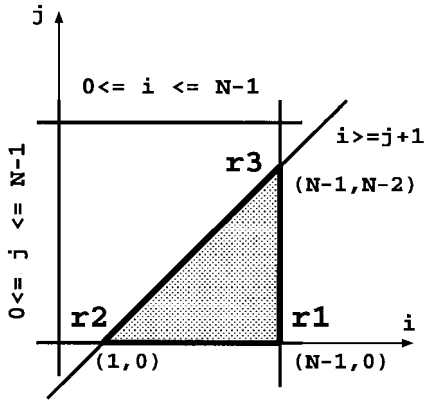


Fig. 2. Elimination of redundant inequalities.

Example: Disregarding the step of the j loop, the iterator space of the array reference $A[j][i+N]$ in line (2) from the Silage code in Fig. 1(a) is represented by the system of inequalities $N-1 \geq i \geq 0$, $N-1 \geq j \geq 0$, $i \geq j+1$, where the parameter $N > 2$. The goal is to eliminate the redundant inequalities (if any) from this two-dimensional (2-D) iterator polytope of the form $\mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}$.

The transformation $\mathbf{x} \rightarrow \begin{pmatrix} \mathbf{x} \\ \xi \end{pmatrix}$, $\xi \geq 0$ changes the initial n -dimensional polytope represented as an inhomogeneous system of inequalities into an $(n+1)$ -dimensional polyhedral cone (see the previous footnote): $\xi \mathbf{A} \cdot \mathbf{i} - \xi \mathbf{b} \geq 0$, which is a homogeneous system. By means of the above preprocessing step, both the vertices and the rays of the initial inhomogeneous polyhedron have a unified representation as rays $[\xi i \ \xi j \ \xi]^T$ in the 3-D polyhedral cone (2-D vertices when $\xi > 0$, and 2-D rays when $\xi = 0$).

Applying the double description algorithm [16] to the homogeneous set of constraints, it results that the lineality space is the null space, and the ray space is generated by the vectors

$$\mathcal{R} = \{r_1 = [N-1 \ 0 \ 1]^T, r_2 = [1 \ 0 \ 1]^T, r_3 = [N-1 \ N-2 \ 1]^T\}.$$

As the resulting three rays have the general form $[\xi v_x \ \xi v_y \ \xi]^T$ with $\xi > 0$, they all represent vertices of coordinates (v_x, v_y) in the initial polytope (see Fig. 2). As none of these vertices saturates the constraints $i \geq 0$ and $N-1 \geq j$ (denoted I_1, I_2), while the other three inequalities are saturated by exactly two vertices r_k each, it follows that I_1 and I_2 are redundant. The minimal representation of the given iterator space is, therefore, $N-1 \geq i, j \geq 0, i \geq j+1$ (denoted I_3, I_4 , and I_5). As mentioned in Section III-A, the two redundant inequalities (I_1 , and I_2) are nonnegative combinations of the other three ones

$$(I_1) = \frac{1}{N-2}(I_3) + \frac{N-1}{N-2}(I_4) + \frac{N-1}{N-2}(I_5)$$

and

$$(I_2) = \frac{N-1}{N-2}(I_3) + \frac{1}{N-2}(I_4) + \frac{N-1}{N-2}(I_5).$$

IV. COMPUTING THE SIZE OF THE INDEX SPACE

When the vectorial affine mapping $\mathbf{f} : \mathbf{Z}^n \rightarrow \mathbf{Z}^m$, defined by $\mathbf{f}(\mathbf{i}) = \mathbf{T}\mathbf{i} + \mathbf{u}$ is injective, the index space and the iterator

space are equal in size, as any n -dimensional point in the iterator space is mapped to a *distinct* m -dimensional point in the index space. Therefore, when \mathbf{f} is injective, our problem is reduced to the iterator space size determination (Section III).

Computing the size of linearly bounded lattices is at least as complex as computing the size of integer polytopes. As mentioned in the introduction of Section III, the complexity of the latter problem is, to the best of our knowledge, still unknown when $n > 4$ [8].

There are two difficulties, related in this context: first, concerning the decision of whether a vectorial affine function is injective or not, and second, concerning the proper computation of the index space size when the mapping is not injective.

Two numerical solutions for the index space problem are proposed in the sequel. The basic idea of the first solution (Section IV-A) is to transform the affine mapping embedded in the array reference and to obtain an equivalent linearly bounded lattice whose size is easier to compute. The second solution is based on the analytic determination of the index space, followed by the direct computation of its size (Section IV-B). Both the proposed solutions have a worst-case exponential complexity. However, their global behavior within the ATOMIUM system [17] proves the effectiveness of the techniques presented in the paper (see Section VI).

A. Transformation of the Affine Mapping

Lemma: Let $\mathbf{U} \in \mathbf{Z}^{n \times n}$ be a unimodular matrix [21]. Then the linearly bounded lattices $\{\mathbf{x} \in \mathbf{Z}^m \mid \mathbf{x} = \mathbf{T} \cdot \mathbf{i} + \mathbf{u}, \mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}, \mathbf{i} \in \mathbf{Z}^n\}$ and $\{\mathbf{y} \in \mathbf{Z}^m \mid \mathbf{y} = \mathbf{T}' \cdot \mathbf{j} + \mathbf{u}, \mathbf{A}' \cdot \mathbf{j} \geq \mathbf{b}, \mathbf{j} \in \mathbf{Z}^n\}$ are equal, where $\mathbf{T}' \stackrel{\text{def}}{=} \mathbf{T} \cdot \mathbf{U}$ and $\mathbf{A}' \stackrel{\text{def}}{=} \mathbf{A} \cdot \mathbf{U}$.

Proof: If \mathbf{x} belongs to the first lattice, there is an n -dimensional vector \mathbf{i} such that $\mathbf{x} = \mathbf{T}\mathbf{i} + \mathbf{u}$ and $\mathbf{A}\mathbf{i} \geq \mathbf{b}$. Since matrix \mathbf{U} is unimodular, \mathbf{U}^{-1} exists and has integer elements, and consequently, $\mathbf{j} \stackrel{\text{def}}{=} \mathbf{U}^{-1}\mathbf{i} \in \mathbf{Z}^n$. It follows immediately that $\mathbf{x} = \mathbf{T}'\mathbf{j} + \mathbf{u}$ and $\mathbf{A}'\mathbf{j} = \mathbf{A}\mathbf{i} \geq \mathbf{b}$. The inverse inclusion is proven similarly. \square

By post-multiplying matrix \mathbf{T} with a sequence of unimodular matrices $\mathbf{S} = \mathbf{U}_1 \cdot \mathbf{U}_2 \cdots$, a reduced Hermite normal form [21], as in Fig. 3, is obtained where all nonzero elements are on or below the main diagonal.⁹ The goal of this transformation is to obtain an equivalent lattice whose size is easier to compute.

Let \mathbf{j}' and \mathbf{j}'' be two distinct points in the transformed iterator polytope $\mathbf{A}' \cdot \mathbf{j} \geq \mathbf{b}$ (where $\mathbf{A}' = \mathbf{A} \cdot \mathbf{S}$), and suppose that $\mathbf{j}'_1 = \mathbf{j}''_1, \dots, \mathbf{j}'_{k-1} = \mathbf{j}''_{k-1}$, and $\mathbf{j}'_k \neq \mathbf{j}''_k$.

- If $k \leq r \stackrel{\text{def}}{=} \text{rank } \mathbf{T}$, then the points $\mathbf{x}' = \mathbf{T}'\mathbf{j}' + \mathbf{u}$ and $\mathbf{x}'' = \mathbf{T}'\mathbf{j}'' + \mathbf{u}$ in the index space differ from each other at least at some i th coordinate, where the i th row of \mathbf{T}' has the k th element nonzero and the last $n-k$ elements equal to zero [see Fig. 3(d)]. As $x'_i \neq x''_i$, the images of \mathbf{j}' and \mathbf{j}'' are distinct.
- If $r < k \leq n$, then $\mathbf{x}' = \mathbf{x}''$ as all their coordinates result to be equal, and, therefore, \mathbf{j}' and \mathbf{j}'' are mapped to the same point in the index space.

⁹Broken lines in the figure.

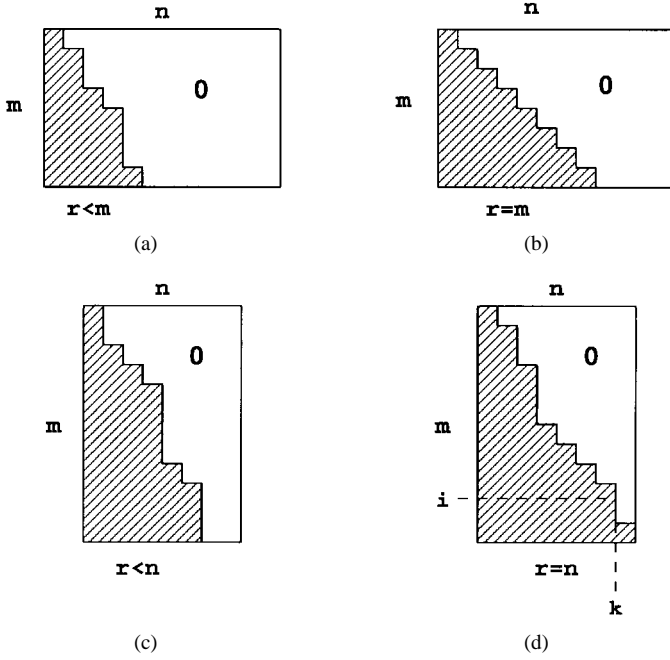


Fig. 3. Unimodular transformation $\mathbf{T}' = \mathbf{T} \cdot \mathbf{S}$ of the affine mapping in an array reference.

Consequently, the size of the index space (i.e., the number of distinct \mathbf{x}) is at most the size of the r -dimensional polytope $pr_r(\mathbf{A}'\mathbf{j} \geq \mathbf{b})$, the projection of $\mathbf{A}'\mathbf{j} \geq \mathbf{b}$ on \mathbf{Z}^r along with the first r coordinates. $pr_r(\mathbf{A}'\mathbf{j} \geq \mathbf{b})$ can be easily computed, by eliminating the last $n - r$ variables in $\mathbf{A}'\mathbf{j} \geq \mathbf{b}$ with the Fourier–Motzkin technique (see Section III-A).

The affine mapping \mathbf{f} is injective (independent of the iterator space) when $r = n$, because $\mathbf{j}' \neq \mathbf{j}''$ implies $\mathbf{x}' \neq \mathbf{x}''$, as shown at case a). In addition, \mathbf{f} may be injective relative to the iterator space when every point in $pr_r(\mathbf{A}'\mathbf{j} \geq \mathbf{b})$ is the projection of at most a single point in $\mathbf{A}'\mathbf{j} \geq \mathbf{b}$, which prevents case b).

It must be noticed that not necessarily all the points of integer coordinates in $pr_r(\mathbf{A}'\mathbf{j} \geq \mathbf{b})$ represent projections of points in $\mathbf{A}'\mathbf{j} \geq \mathbf{b}$. For instance, the polytope $j_1 + 4 \geq 8j_2 \geq -j_1 + 4$, $3 \geq j_1$ has the projection $3 \geq j_1 \geq 0$, but none of the points $(0, 1, 2, 3)$ is the projection of a lattice point in the polytope.¹⁰ These invalid projections are detected easily. It is sufficient to replace the coordinates of the projection point in the polytope $\mathbf{A}'\mathbf{j} \geq \mathbf{b}$ and to check whether the resulting $(n - r)$ -dimensional integer polytope is empty or not.

Examples:

$$(i : 0 .. 9) :: (j : 0 .. 3) :: \dots A[10 * i + 3 * j] \dots$$

The unimodular matrix $\mathbf{S} = \begin{bmatrix} 1 & -3 \\ -3 & 10 \end{bmatrix}$ transforms the mapping $\mathbf{T} = [10 \ 3]$ to $\mathbf{T}' = [1 \ 0]$, having a form as in Fig. 3(b). The iterator polytope corresponding to the new mapping is $P' = \{[j_1 \ j_2]^T \mid 9 \geq j_1 - 3j_2 \geq 0, 0 \geq 3j_1 - 10j_2 \geq -3\}$, and its projection polytope is $pr_1 P' = \{j_1 \mid 99 \geq j_1 \geq 0\}$. Only 40 points in $pr_1 P'$ are projections of points in P' . For instance, $j_1 = 1$ is not a valid projection. Replacing this value in P' , there is no integer solution for j_2 . The size of the index

¹⁰ Although the *real* polytope is not empty [it contains, e.g., the point $(2, 0.5)$], the *integer* polytope is empty.

space is 40, and the mapping is injective relative to the iterator space¹¹ $\{[i \ j]^T \mid 9 \geq i \geq 0, 3 \geq j \geq 0\}$, the size of which is also 40.

In the “normalized” example from Section II, the unimodular transformation $\mathbf{S} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$ brings matrix $\mathbf{T} = [2 \ 2]$ to a form as in Fig. 3(b). The iterator polytope corresponding to the new mapping is $P' = \{[j_1 \ j_2]^T \mid j_2 + 15 \geq j_1 \geq j_2 + 8, 8 \geq j_2 \geq 0\}$, and its projection is $pr_1 P' = \{j_1 \mid 23 \geq j_1 \geq 8\}$. All the 16 points in $pr_1 P'$ are projections of at least one point in P' . It is sufficient to replace the 16 values of i in P' and verify that, each time, the resulting (integer) polytope in j is not empty. Consequently, the size of the index space is 16. The mapping \mathbf{f} is not injective as, for instance, there are two points $[(9, 0)$ and $(9, 1)]$ in P' having the same projection (9) in $pr_1 P'$.

The algorithm described above is implemented in a routine *SizeIndexSpace (Lbl)*. The method will be illustrated more thoroughly in Section VI.

B. Index Space Determination for Array References

The basic idea of the previous approach is to find an appropriate transformation of the affine mapping, in order to compute the size of the index space by investigating the transformed iterator space. A more direct way of tackling the problem is to model the index space by means of a coordinate transformation from \mathbf{i} to \mathbf{x} and afterwards, to carry out the size computation inside the index space. But the index space modeling is not easy to accomplish.

Let $\{\mathbf{x} = \mathbf{T} \cdot \mathbf{i} + \mathbf{u} \mid \mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}\}$ be the linearly bounded lattice of a given array reference. If matrix \mathbf{T} is square and nonsingular, the index space is included into an *image polytope*, which can be easily determined noticing that the iterator vector $\mathbf{i} = \mathbf{T}^{-1} \cdot (\mathbf{x} - \mathbf{u})$ must represent a point inside the iterator polytope $\mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}$. Therefore, the resulting image polytope is: $\mathbf{A} \cdot \mathbf{T}^{-1} \cdot (\mathbf{x} - \mathbf{u}) \geq \mathbf{b}$.

Even in this case when matrix \mathbf{T} is invertible, not all the points in the image polytope belong to the index space. For instance, if $(i : 0 .. 9) :: \dots A[5 * i] \dots$, the image polytope of the array reference is $\{x \in \mathbf{Z} \mid 0 \leq x \leq 45\}$. But the indexes of A take only ten from these 46 values.

This section will show how to model the index space of an array reference, even when matrix \mathbf{T} is singular or nonsquare¹² (see Fig. 4). Afterwards, the size of the index space will be directly derived from this model.

For any matrix $\mathbf{T} \in \mathbf{Z}^{m \times n}$ having $\text{rank} \mathbf{T} = r$ and assuming the first r rows of \mathbf{T} are linearly independent,¹³ there exists a unimodular matrix $\mathbf{S} \in \mathbf{Z}^{n \times n}$ such that $\mathbf{T} \cdot \mathbf{S} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{0} \\ \mathbf{H}_{21} & \mathbf{0} \end{bmatrix}$, where $\mathbf{H}_{11} \in \mathbf{Z}^{r \times r}$ is a lower triangular matrix with positive diagonal elements, and $\mathbf{H}_{21} \in \mathbf{Z}^{(m-r) \times r}$ [14]. The block matrix is called the reduced Hermite form of matrix \mathbf{T} .

¹¹ The mapping may not be injective if the iterator space is modified: e.g., $(i : 0 .. 9) :: (j : 0 .. 10) ::$

¹² That is, the dimension of the index space differs from the dimension of the iterator space.

¹³ This assumption does not decrease the generality. It is done only to simplify the formulas, affected otherwise by a permutation matrix.

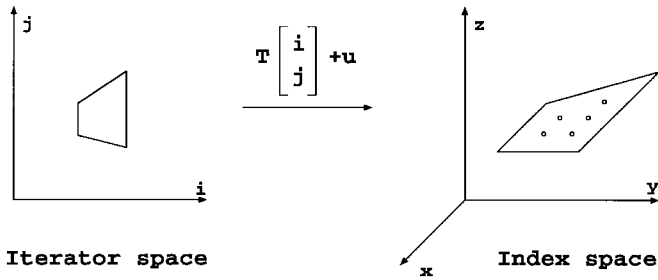


Fig. 4. Affine transformation of a polytope from the iterator space to the index space.

Let $\mathbf{S}^{-1}\mathbf{i} \stackrel{\text{def}}{=} \mathbf{j} \equiv \begin{bmatrix} \mathbf{j}_1 \\ \mathbf{j}_2 \end{bmatrix}$, where $\mathbf{j}_1, \mathbf{j}_2$ are r -, respectively, $(n - r)$ -, dimensional vectors. Then

$$\mathbf{x} = \mathbf{T}\mathbf{i} + \mathbf{u} = \mathbf{T}\mathbf{S}\mathbf{j} + \mathbf{u} = \begin{bmatrix} \mathbf{H}_{11} \\ \mathbf{H}_{21} \end{bmatrix} \mathbf{j}_1 + \mathbf{u}. \quad (5)$$

Denoting $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$, and $\mathbf{u} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix}$ (where $\mathbf{x}_1, \mathbf{u}_1$ are r -dimensional vectors), it follows that $\mathbf{x}_1 = \mathbf{H}_{11}\mathbf{j}_1 + \mathbf{u}_1$. As \mathbf{H}_{11} is nonsingular, being lower triangular of rank r , \mathbf{j}_1 can be obtained explicitly

$$\mathbf{j}_1 = \mathbf{H}_{11}^{-1}(\mathbf{x}_1 - \mathbf{u}_1). \quad (6)$$

The iterator vector \mathbf{i} results with a simple substitution

$$\begin{aligned} \mathbf{i} &= \mathbf{S} \begin{bmatrix} \mathbf{j}_1 \\ \mathbf{j}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{S}_2 \end{bmatrix} \begin{bmatrix} \mathbf{H}_{11}^{-1}(\mathbf{x}_1 - \mathbf{u}_1) \\ \mathbf{j}_2 \end{bmatrix} \\ &= \mathbf{S}_1\mathbf{H}_{11}^{-1}(\mathbf{x}_1 - \mathbf{u}_1) + \mathbf{S}_2\mathbf{j}_2 \end{aligned}$$

where \mathbf{S}_1 and \mathbf{S}_2 are the submatrices of \mathbf{S} containing the first r and the last $n - r$, columns of \mathbf{S} , respectively. As the iterator vector must represent a point inside the iterator polytope $\mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}$, it follows that

$$\mathbf{A}\mathbf{S}_1\mathbf{H}_{11}^{-1}\mathbf{x}_1 + \mathbf{A}\mathbf{S}_2\mathbf{j}_2 \geq \mathbf{b} + \mathbf{A}\mathbf{S}_1\mathbf{H}_{11}^{-1}\mathbf{u}_1. \quad (7)$$

As the rows of matrix \mathbf{H}_{11} are r linearly independent r -dimensional vectors, each row of \mathbf{H}_{21} is a linear combination of the rows of \mathbf{H}_{11} . Then from (5), it results that there exists a matrix $\mathbf{C} \in \mathbb{R}^{(m-r) \times r}$ such that¹⁴

$$\mathbf{x}_2 - \mathbf{u}_2 = \mathbf{C} \cdot (\mathbf{x}_1 - \mathbf{u}_1). \quad (8)$$

If $r < n$, the *image polytope* of the index space can be obtained eliminating with the Fourier–Motzkin technique the $n - r$ variables of \mathbf{j}_2 . Usually, this image polytope contains “holes,”¹⁵ as not all its points represent valid projections of the n -dimensional polytope (7) as explained in Section IV-A. Even if $r = n$ (therefore, no projection is needed), the image polytope is not dense when $\det \mathbf{H}_{11} \neq 1$, as it can be seen in

¹⁴The coefficients of matrix \mathbf{C} are determined by backward substitutions from the equations: $\mathbf{H}_{21}.\text{row}(i) = \sum_{j=1}^r c_{ij} \cdot \mathbf{H}_{11}.\text{row}(j)$ for any $i = 1, \dots, m - r$.

¹⁵That is, points in the image polytope but not in the index space.

Fig. 1(b), where $\det \mathbf{H}_{11} = 2$ for both operands $A[j][i + N]$ after “normalizing” the j loop.

Indeed, taking into account that the elements of \mathbf{j}_1 must be integers, it follows by multiplying and dividing the right member of (6) with $\det \mathbf{H}_{11}$ that the points \mathbf{x} inside the index space must supplementarily satisfy the divisibility constraints

$$\det \mathbf{H}_{11} \mid \mathbf{h}_i^T(\mathbf{x}_1 - \mathbf{u}_1) \quad \forall i = 1, \dots, r \quad (9)$$

where \mathbf{h}_i^T are the rows of the matrix with integer coefficients $\det \mathbf{H}_{11} \cdot \mathbf{H}_{11}^{-1}$, and $a \mid b$ means “ a divides b .” According to (9), when $r = n$, the points \mathbf{x} are uniformly spaced along the r linear independent coordinates, the size of gaps in these dimensions being equal to the diagonal elements¹⁶ of \mathbf{H}_{11} . This latter result was known to be valid for matrices \mathbf{T} square and nonsingular [11].

Example: $(i : 0..9) :: \dots A[5 * i] \dots$. As $r = m = n = 1$, $\mathbf{H}_{11} = \mathbf{T} = [5]$, and $\mathbf{S} = [1]$, the inequalities (7) yield the image polytope $0 \leq x \leq 45$. As $r = m$, there is no equality (8). In addition, the points x of the index space are subject to the divisibility constraint (9): $5 \mid x$. The gap between these points is equal to $h_{11} = 5$.

The system of inequality (7), (8), and the divisibility conditions (9) (not necessary if \mathbf{H}_{11} is unimodular) characterize the index space of the given array reference.

In conclusion, the size of the index space is equal to the size of the projection polytope $pr_r([\mathbf{A}\mathbf{S}_1\mathbf{H}_{11}^{-1} \ \mathbf{A}\mathbf{S}_2], \mathbf{b} + \mathbf{A}\mathbf{S}_1\mathbf{H}_{11}^{-1}\mathbf{u}_1)$ less the points which do not represent valid projections of the polytope (7) and taking, in addition, the divisibility constraints (9) into account. The computation can be easily performed by the routine *CountPolytopePoints*, described in Section III-A, slightly modified to accumulate only the points which are valid projections of (7) and, supplementarily, satisfy the constraints (9) when matrix \mathbf{H}_{11} is not unimodular.

Two benchmark tests in Section VI will exemplify this method.

V. OTHER APPLICATIONS OF THE COUNTING ALGORITHMS

A. The Equivalence of Array References

Two array references of an indexed signal are equivalent if they have the same index spaces. The equivalence test reduces to the equality verification of two linearly bounded lattices [3]

$$\begin{aligned} \{ \mathbf{x} = \mathbf{T}_1\mathbf{i}_1 + \mathbf{u}_1 \in \mathbf{Z}^m \mid \mathbf{A}_1\mathbf{i}_1 \geq \mathbf{b}_1, \mathbf{i}_1 \in \mathbf{Z}^{n_1} \} \\ \stackrel{?}{=} \{ \mathbf{x} = \mathbf{T}_2\mathbf{i}_2 + \mathbf{u}_2 \in \mathbf{Z}^m \mid \mathbf{A}_2\mathbf{i}_2 \geq \mathbf{b}_2, \mathbf{i}_2 \in \mathbf{Z}^{n_2} \}. \end{aligned}$$

Example: $(i : 0 .. N) :: \dots = A[i] + A[N - i]$. The two array references are equivalent, as they cover the same set of scalars $A[0], \dots, A[N]$. Their corresponding LBL’s modeling the set of indexes $\{x = i \mid N \geq i \geq 0, i \in \mathbf{Z}\}$ and $\{x = -i + N \mid N \geq i \geq 0, i \in \mathbf{Z}\}$ are equal.

¹⁶If h_{ii} are the diagonal elements of matrix \mathbf{H}_{11} , it can be verified that the divisibility constraints (9) are not affected when \mathbf{x}_1 is subject to translations of vectors $\mathbf{v}_i = [0 \ \dots \ h_{ii} \ \dots \ 0]^T, \forall i = 1, \dots, r$. Indeed, $\mathbf{h}_i^T(\mathbf{x}_1 + \mathbf{v}_i - \mathbf{u}_1) = \mathbf{h}_i^T(\mathbf{x}_1 - \mathbf{u}_1) + \mathbf{h}_i^T\mathbf{v}_i = \mathbf{h}_i^T(\mathbf{x}_1 - \mathbf{u}_1) + \det \mathbf{H}_{11}$.

```

bool EquivalenceTest (Lbl1, Lbl2) { // Lbl1  $\stackrel{?}{=}$  Lbl2
  if ((n1 = SizeIndexSpace(Lbl1)) != SizeIndexSpace(Lbl2)) return NO;
  else return (n1 == SizeIndexSpace(Lbl1 ∩ Lbl2));
}

```

Testing the equivalence of array references is necessary in the context of processing behavioral specifications for checking the correctness of code transformations [20]. It is also employed in the context of memory estimation/allocation for multidimensional signal processing [1], [3], to construct correct (polyhedral) dependence graphs for behavioral specifications.¹⁷

Testing the equality of linearly bounded lattices cannot be done comparing their iterator polytopes $\mathbf{A}_1 \mathbf{i}_1 \geq \mathbf{b}_1$ and $\mathbf{A}_2 \mathbf{i}_2 \geq \mathbf{b}_2$. These can have different sizes or even belong to spaces of different dimensions, and still they can be mapped to the same image. For instance, the array references

$$(i : 0 .. 2) :: (j : 0 .. 2) :: \dots A[i + j] \dots$$

$$(i : 0 .. 4) :: \dots A[i] \dots$$

have the same index space $\{x \in \mathbf{Z} \mid 0 \leq x \leq 4\}$, although the iterator space dimensions are different (two and one, respectively), and the iterator polytopes have different sizes (nine and five, respectively).

The index spaces of the two array references are also difficult to compare because of the “holes” (see Section IV-B). If the index spaces have the same dimension and the image polytopes are dense, then an equivalence test is possible. It is sufficient to compute the vertices of the two image polytopes employing Chernikova’s algorithm [5], [13] and to check the equality of the two sets of vertices. As the image polytopes can contain “holes,” the equality of the image polytopes is not a proof that the corresponding array references are equivalent. For instance

$$(i : 0 .. 15) :: \dots A[2 * i] \dots$$

$$(i : 0 .. 10) :: \dots A[3 * i] \dots$$

have the same image polytope: $\{x \in \mathbf{Z} \mid 0 \leq x \leq 30\}$, but the two array references have obviously different index spaces.

A simple and universal¹⁸ equivalence test is shown at the top of the page. A necessary and sufficient equality condition is the following. The two LBL’s must have the same size, and at the same time, their intersection must also have the same size as two equal LBL’s are obviously equal also to their intersection. The problem of “holes” is, therefore, circumvented, and furthermore, the only necessary operations

¹⁷If the equivalence of arrays (or parts of arrays having the index space modeled as LBL’s) is not detected, false nodes and dependences can appear in the dependence graphs, affecting the necessary storage requirement of the given specification.

¹⁸Disregarding the enumerative techniques, which are too computationally expensive whenever dealing with large iterator spaces.

are LBL intersection, e.g., solved as in [1], and index space size computation (Section IV).

B. Counting the Dependences Between Array References

The number of dependences between array references has been previously employed in data locality exploitation [9] but only when the surrounding loops have constant boundaries. More recently, the number of dependences between (parts of) arrays has been employed to compute tight upper bounds for memory size in nonprocedural algorithmic specifications [3, ch. 4].

Let $A[x_1(i_1, \dots, i_n)] \dots [x_m(i_1, \dots, i_n)]$ and $A[y_1(j_1, \dots, j_s)] \dots [y_m(j_1, \dots, j_s)]$ be two array references of an m -dimensional signal A , in the scope of two nested loops having the iterators (i_1, \dots, i_n) and (j_1, \dots, j_s) , respectively. The number of dependences between the two array references is the number of vectors $(i_1, \dots, i_n, j_1, \dots, j_s)$ for which the indexes are respectively equal: $x_1 = y_1, \dots, x_m = y_m$. The number of dependence relations is the number of distinct index vectors (x_1, \dots, x_m) for which there is at least a dependence.

Example:

$$(i : 0 .. 2) :: (k : 0 .. 9) ::$$

$$(j : 0 .. 2) :: \dots A[i + j] \dots \dots A[2 * k] \dots$$

There are five dependences between the two array references as there are five vectors $(i, j, k) = \{(0, 0, 0), (2, 0, 1), (1, 1, 1), (0, 2, 1), (2, 2, 2)\}$ —leading to the equality of the indexes. There are only three dependence relations, as a dependence can happen only for the indexes 0, 2, and 4.

Let $\{\mathbf{x} = \mathbf{T}_1 \mathbf{i}_1 + \mathbf{u}_1 \mid \mathbf{A}_1 \mathbf{i}_1 \geq \mathbf{b}_1\}$, $\{\mathbf{x} = \mathbf{T}_2 \mathbf{i}_2 + \mathbf{u}_2 \mid \mathbf{A}_2 \mathbf{i}_2 \geq \mathbf{b}_2\}$ be the two linearly bounded lattices modeling the index space of the two given array references. The scalars common to both index spaces belong to the intersection of the two LBL’s, which is also a linearly bounded lattice $Lbl = \{\mathbf{x} = \mathbf{T} \cdot \mathbf{i} + \mathbf{u} \mid \mathbf{A} \cdot \mathbf{i} \geq \mathbf{b}\}$ computed, e.g., as in [1].

The number of dependences is provided by the routine $CountPolytopePoints(\mathbf{A}, \mathbf{b})$, described in Section III. The number of dependence relations can be computed with the routine $SizeIndexSpace(Lbl)$ (Section IV-A).

VI. ILLUSTRATIVE EXAMPLES AND RESULTS

The presented techniques have been implemented in C++ and tested on an HP 9000/735 workstation. They are embedded in ATOMIUM, a memory management system for multidimensional signal processing [17]. They are actually used by the memory estimation [1] and allocation tools [2]. They are also useful in the LOVER verification environment for validating loop transformations on algorithmic specifications [20].

Two examples illustrate the computation of the index space size with the routine $SizeIndexSpace$ (Section IV-A).

Example 1:

$$(i : 0..511) :: (j : 0..511) :: \dots M[2*i + 3*j + 1] \\ [5*i + j + 2][4*i + 6*j + 3] \dots$$

$$\{\mathbf{x} = \mathbf{T}\mathbf{i} + \mathbf{u} \mid \mathbf{A}\mathbf{i} \geq \mathbf{b}\} = \left\{ \begin{array}{l} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 5 & 1 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \geq \begin{bmatrix} 0 \\ -511 \\ 0 \\ -511 \end{bmatrix} \end{array} \right\}.$$

As there is a unimodular matrix $\mathbf{S} = \begin{bmatrix} -1 & 3 \\ 1 & -2 \end{bmatrix}$ such that

$$\mathbf{T} \cdot \mathbf{S} = \begin{bmatrix} 1 & 0 \\ -4 & 13 \\ \dots & \dots \\ 2 & 0 \end{bmatrix}$$

as in Fig. 3(d), it results that $\text{rank } \mathbf{T} = 2 = n$. Hence, the mapping is injective. Therefore, the index and iterator spaces are equal in size. The size of the iterator polytope is computed with the routine *CountPolytopePoints*(\mathbf{A}, \mathbf{b}) which takes into account that both iterators have constant bounds[see Section III-A, item 1)] with range 512 each. The size of the index space is, therefore, $512 \times 512 = 262144$.

Example 2:

$$(i : 0 .. 511) :: (j : 0 \dots 511) :: (k : 0 .. 511) :: \\ \dots M[i + k][j + k] \dots$$

With the unimodular matrix

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

As $r = m < n$, as in Fig. 3(b), ($2 < 3$), the transformed iterator polytope is $\{\mathbf{j} \in \mathbf{Z}^3 \mid \mathbf{A}' \cdot \mathbf{j} \geq \mathbf{b}\} = \{511 \geq j_1 - j_3 \geq 0, 511 \geq j_2 - j_3 \geq 0, 511 \geq j_3 \geq 0\}$. Its projection is $\text{pr}_2(\mathbf{A}' \cdot \mathbf{j} \geq \mathbf{b}) = \{(j_1, j_2) \in \mathbf{Z}^2 \mid 511 \geq j_1 - j_2 \geq -511, 1022 \geq j_1 \geq 0, 1022 \geq j_2 \geq 0\}$.

There are 784 897 points (j_1, j_2) in $\text{pr}_2(\mathbf{A}' \cdot \mathbf{j} \geq \mathbf{b})$, and all of them prove to be valid projections. For instance, $(1 \ 1)$ is the projection of two points, $(1 \ 1 \ 0)$ and $(1 \ 1 \ 1)$, satisfying the system of inequalities $\mathbf{A}' \cdot \mathbf{j} \geq \mathbf{b}$. It follows that the size of the index space of $M[i + k][j + k]$ is 784 897. As this value is inferior to the size of the iterator space, which is $512^3 = 134\,217\,728$, it follows that the affine mapping $\mathbf{t}(i, j, k) = \mathbf{T} \cdot [i \ j \ k]^T$ is not injective. Indeed, this can be easily seen noticing that the triplets (i, j, k) equal to $(0, 1, 1)$ and $(1, 2, 0)$ are mapped to the same point $(1, 2)$ in the index space.

Table I shows the central processing unit (CPU) times (column 6) for the computation of the index space size when the upper boundaries of the three loops (column 1) are increased gradually. It can be noticed that the ratio (column 4) between the LBL size (column 3) and the iterator space size (column 2) is decreasing dramatically if the affine mapping is

TABLE I
THE COMPUTATION OF THE INDEX SPACE SIZE FOR AN ARRAY
REFERENCE. EXPERIMENTAL RESULTS FOR EXAMPLE 2

Loop bound	Size polytope (iterator sp.)	Size LBL (index sp.)	LBL/Polytope ratio [%]	CPU	
				Enum.	SizeIndexSpace
3	64	37	57	0.0004s	0.0005s
7	512	169	33	0.012s	0.015s
15	4096	721	17	0.087s	0.050s
31	32768	2977	9.0	1.05s	0.18s
63	262144	12097	4.6	17.48s	0.70s
127	2,097,152	48769	2.3	5m23s	2.75s
255	16,777,216	195841	1.1	1h55m36s	10.86s
511	134,217,728	784897	0.5	?	43.25s

TABLE II
THE COMPUTATION OF THE INDEX SPACE SIZE WHEN ALL THE
REDUNDANT INEQUALITIES ARE DETECTED DURING THE
FOURIER-MOTZKIN ELIMINATION. EXPERIMENTAL RESULTS FOR EXAMPLE 2

Loop bound	3	7	15	31	63	127	255	511
CPU [sec]	0.013	0.028	0.076	0.24	0.85	3.00	11.27	44.10

TABLE III
THE COMPUTATION OF THE LBL SIZE BY INDEX SPACE
DETERMINATION. EXPERIMENTAL RESULTS FOR EXAMPLE 2

Loop bound	3	7	15	31	63	127	255	511
CPU [sec]	0.0005	0.020	0.060	0.20	0.78	2.88	11.02	43.40

not injective. Therefore, the approximation of the index space size by the iterator space size must be avoided!

The CPU times obtained employing an enumerative technique are displayed in column 5. They may be lower than those corresponding to the routine *SizeIndexSpace* for small iterator spaces, but they grow dramatically with the LBL size as the enumerative technique has to keep track of the index values. Although the run times for our routine are also increasing significantly (linear with the index space size), the method is still effective even for huge sizes of the iterator space.

The computation time is usually affected by the size of the iterator space,¹⁹ as shown in Section IV-A. For the current example, the CPU time increases approximately quadratically with the upper bound of the loops, while the iterator space size increases cubically. However, the global behavior of the algorithm is usually better, as the results in Table V will show.

Table II displays the run times for the routine *SizeIndexSpace* when the double description of polyhedra (see Section III-B) is employed to detect all the redundant inequalities during the Fourier-Motzkin elimination. It can be seen from this example that the detection of all redundant inequalities can create more overhead than speed benefit. The simple technique described in Section III-A which eliminates only part of the redundant inequalities yields in general better CPU times.

The same examples will illustrate also the computation of the index space size by the effective construction of the index space of an array reference (Section IV-B).

¹⁹Unless the case when the mapping is injective, and the iterator space is a hypercube.

TABLE IV
COMPARISON BETWEEN THE TWO METHODS FOR COMPUTING THE SIZE OF AN INDEX SPACE. EXPERIMENTAL RESULTS FOR EXAMPLE 1

Loop bound	Size image polytope	Size LBL (index sp.)	Ratio [%] LBL/ImagePolytope	CPU [sec]	
				<i>SizeIndexSpace</i>	Index sp. determ.
255	845,836	65,536	7.74	0.0080	0.13
511	3,395,596	262,144	7.72	0.0157	0.57
767	7,649,292	589,824	7.71	0.0235	1.32
1023	13,606,924	1,048,576	7.70	0.0312	2.44

```

(i: -M+1..M)::          /* The parameters M,N,m,n are defined */
(j: -N+1..N)::
(k: -m+1..m-1)::
(l: -n+1..n-1)::
  Delta[2*M+i][2*N+j][k+m][l+n+1] = Delta[2*M+i][2*N+j][k+m][l+n]
  + A[2*M+i][2*N+j] - Ad[2*M+i][2*N+j][2*M+i+k][2*N+j+1] ;
  -----

```

Fig. 5. Excerpt from the motion detection algorithm.

Example 1 (Revisited): As $m = 3$ and $r = n = 2$, we have $\mathbf{x}_1 = [x \ y]^T$ and $\mathbf{x}_2 = [z]$. Taking also into account the results already obtained above

$$\mathbf{S}_1 \equiv \mathbf{S} = \begin{bmatrix} -1 & 3 \\ 1 & -2 \end{bmatrix}, \quad \mathbf{H}_{11} = \begin{bmatrix} 1 & 0 \\ -4 & 13 \end{bmatrix}$$

$$\mathbf{H}_{11}^{-1} = \frac{1}{13} \begin{bmatrix} 13 & 0 \\ 4 & 1 \end{bmatrix}, \quad \mathbf{H}_{21} = [2 \ 0].$$

Condition (7) yields

$$\begin{aligned} 6648 &\geq -x + 3y \geq 5 \\ 6644 &\geq 5x - 2y \geq 1. \end{aligned}$$

As the only row in \mathbf{H}_{21} is two times the first row in \mathbf{H}_{11} , the image polytope is completed with the equality (8): $z - 3 = 2(x - 1)$.

As $\det \mathbf{H}_{11} = 13$, the image polytope is not dense. Condition (9) yields $13 \mid 4(x - 1) + (y - 2)$, as $13 \mid 13(x - 1)$ is always satisfied.

The number of points in the image polytope, also satisfying the divisibility condition above, equals 262 144, the size of the index space which is also here the size of the iterator space.

Example 2 (Revisited): As $m = r = 2$ and $n = 3$, we denote the $(n - r)$ -dimensional vector $\mathbf{j}_2 = [\lambda]$. As $\mathbf{H}_{11} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (thus unimodular) and \mathbf{H}_{21} does not exist (as $m = r$), with

$$\mathbf{S}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{S}_2 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$$

only condition (7), yields

$$\begin{aligned} 511 &\geq x & -\lambda &\geq 0 \\ 511 &\geq y - \lambda & &\geq 0 \\ 511 &\geq \lambda & &\geq 0. \end{aligned}$$

The number of points $\mathbf{x} = [x \ y]^T$ in the image polytope is 784 897, the number of valid projections in $\{(x, y) \in \mathbf{Z}^2 \mid 511 \geq x - y \geq -511, 1022 \geq x \geq 0, 1022 \geq y \geq 0\}$.

The experimental run times, displayed in Table III for the same upper bounds of loops as in Table I, are similar (only slightly higher due to the matrix multiplications).

When matrix \mathbf{H}_{11} is not unimodular, the CPU times can be significantly higher than those yielded by the routine *SizeIndexSpace*. The reason is that the method of index space determination operates on the *image* polytope (which contains “holes”), while *SizeIndexSpace* operates on the *iterator* polytope, which can be significantly “smaller” when \mathbf{H}_{11} is not unimodular. As shown in Table IV, the “holes” represent more than 92% of the image polytope. Exploiting the fact that, in Example 1, gaps in the index space have uniform sizes (of $h_{11} = 1$ along x , and $h_{22} = 13$ along y), as explained in Section IV-B, the routine based on index space determination is still practical, although not as efficient as *SizeIndexSpace*.

In order to demonstrate that the algorithm having a worst-case exponential complexity is, however, globally effective, the CPU time contribution of the procedure *SizeIndexSpace* has been monitored, while dealing with two signal processing applications.

The first benchmark test is an algorithm for updating the singular value decomposition of matrices—algebraic kernel used, e.g., in beamforming and Kalman filtering. The code given in [3, pp. 182–184] contains nested loops up to three levels deep, the upper bounds depending on N —the order of matrices. The iterator space of most of the array references is growing proportionally to N^3 . The total CPU time, while processing all the array references in the code, as well as the average CPU time per array reference, is increasing roughly linearly with N (see Table V).

The second benchmark test is a motion detection algorithm used for compressing moving images. The full code is given in [3, pp. 185–187], and an excerpt from it is shown in Fig. 5. The code contains nested loops up to four levels deep. Most of the signal indexes are complex. The CPU times in Table V show a slightly less than linear increase with each of the four

TABLE V
EXPERIMENTAL RESULTS FOR THE PRACTICAL COMPUTATIONAL
COMPLEXITY OF THE PROCEDURE *SizeIndexSpace*

Application	Parameters	CPU [sec]	Average CPU [sec]
Updating SVD	N=128	0.70	0.0084
	N=256	1.11	0.0133
	N=512	2.17	0.0261
	N=1024	4.56	0.0549
	N=2048	9.23	0.1112
Motion detection G=H=512	M=N=8 m=n=2	0.36	0.0053
	m=n=4	0.45	0.0066
	M=N=16 m=n=2	1.12	0.0164
	m=n=4	1.28	0.0188
	m=n=8	1.57	0.0230
	M=N=32 m=n=2	3.81	0.0560
	m=n=4	4.20	0.0617
	m=n=8	4.83	0.0710
	m=n=16	5.72	0.0841
	M=N=64 m=n=2	14.47	0.2127
	m=n=4	15.22	0.2238
	m=n=8	16.17	0.2378
	m=n=16	18.61	0.2736
	m=n=32	20.85	0.3066

parameters M , N , m , and n , with each iterator range being proportional to one of the parameters.

VII. CONCLUSION

In the context of memory management for multidimensional signal processing systems, we have addressed a central problem encountered when handling array variables in behavioral specifications: counting the scalars covered by array references. The importance of the problem results from the fact that in memory allocation tools the size of array references influence the storage requirements. The novel algorithms described in this paper can solve also a related problem of data-flow analysis: counting the dependences between array references. These algorithms are embedded in ATOMIUM, a memory management system targeting mainly image and video processing applications.

REFERENCES

- [1] F. Balasa, F. Catthoor, and H. De Man, "Background memory area estimation for multi-dimensional signal processing systems," *IEEE Trans. VLSI Syst.*, vol. 3, pp. 157–172, June 1995.
- [2] ———, "Dataflow-driven memory allocation for multi-dimensional signal processing systems," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 31–34.
- [3] F. Balasa, "Background memory allocation for multi-dimensional signal processing," Ph.D. dissertation, IMEC, Katholieke Univ. Leuven, Leuven, Belgium, 1995.
- [4] U. Banerjee, R. Eigenmann, A. Nicolau, and D. A. Padua, "Automatic program parallelization," *Proc. IEEE*, vol. 81, pp. 211–243, Feb. 1993.
- [5] N. V. Chernikova, "Algorithm for discovering the set of all the solutions of a linear programming problem," *USSR Comput. Math. Math. Phys.*, vol. 8, pp. 282–293, 1968.
- [6] G. B. Dantzig and B. C. Eaves, "Fourier–Motzkin elimination and its dual," *J. Combinatorial Theory (A)*, vol. 14, pp. 288–297, 1973.
- [7] R. J. Duffin, "On Fourier's analysis of linear inequality systems," *Math. Program. Stud.*, vol. 1, pp. 71–95, 1974.
- [8] M. Dyer, "On counting lattice points in polyhedra," *SIAM J. Comput.*, vol. 20, pp. 695–707, Aug. 1991.
- [9] C. Eisenbeis, O. Temam, and H. Wijshoff, "Fast enumeration of solutions for data dependence analysis and data locality optimization," in *Proc. Int. Conf. Parallel Processing*, Aug. 1993.

- [10] P. N. Hilfinger, J. Rabaey, D. Genin, C. Scheers, and H. De Man, "DSP specification using the Silage language," in *Proc. Int. Conf. Acoustics, Speech Signal Processing*, Apr. 1990, pp. 1057–1060.
- [11] W. Li and K. Pingali, "A singular loop transformation framework based on nonsingular matrices," in *Proc. 5th Annu. Workshop Languages Compilers Parallelism*, Aug. 1992.
- [12] P. Lippens, J. van Meerbergen, W. Verhaegh, and A. van der Werf, "Allocation of multiport memories for hierarchical data streams," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 728–735.
- [13] T. H. Matheiss and D. S. Rubin, "A survey and comparison of methods for finding all vertices of convex polyhedral sets," *Math. Oper. Res.*, vol. 5, pp. 167–185, May 1980.
- [14] M. Minoux, *Mathematical Programming—Theory and Algorithms*. New York: Wiley, 1986.
- [15] L. J. Mordell, "Lattice points in a tetrahedron and generalized Dedekind sums," *J. Indian Math.*, vol. 15, pp. 41–46, 1951.
- [16] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall, "The double description method," in *Contributions to the Theory of Games—Annals of Mathematics Studies*, no. 28, H. W. Kuhn and A. W. Tucker, Eds. Princeton, NJ: Princeton Univ. Press, 1953, pp. 51–73.
- [17] L. Nachtergaele, F. Catthoor, F. Balasa, F. Franssen, E. De Greef, H. Samsom, and H. De Man, "Optimization of memory organization and partitioning for decreased size and power in video and image processing systems," in *Proc. Int. Workshop Memory Technology, Design, Testing*, Aug. 1995, pp. 82–87.
- [18] W. Pugh, "A practical algorithm for exact array dependence analysis," *Commun. ACM*, vol. 35, pp. 102–114, Aug. 1992.
- [19] L. Ramachandran, D. D. Gajski, and V. Chaiyakul, "An algorithm for array variable clustering," in *Proc. European Design Test Conf.*, Feb. 1994, pp. 262–266.
- [20] H. Samsom, F. Franssen, F. Catthoor, and H. De Man, "Verification of loop transformations for real time signal processing applications," in *VLSI Signal Processing VII*, J. Rabaey, P. Chau, and J. Eldon, Eds. New York: IEEE Press, 1994, pp. 208–217.
- [21] A. Schrijver, *Theory of Linear and Integer Programming*. New York: Wiley, 1986.
- [22] N. Tawbi, "Parallelization automatique: estimation des durées d'exécution et allocation statique de processeurs," Ph.D. dissertation, Inst. Blaise Pascal, Université Paris, France, Sept. 1991.
- [23] L. Thiele, "Compiler techniques for massive parallel architectures," in *State-of-the-Art in Computer Science*, P. Dewilde, Ed. Norwell, MA: Kluwer, 1992.
- [24] M. van Swaaij, F. Franssen, F. Catthoor, and H. De Man, "High-level modeling of data and control flow for signal processing systems," in *Design Methodologies for VLSI DSP Architectures and Applications*, M. Bayoumi, Ed. Norwell, MA: Kluwer, 1993.
- [25] I. Verbauwhede, C. Scheers, and J. M. Rabaey, "Memory estimation for high level synthesis," in *Proc. 31st Design Automation Conf.*, June 1994, pp. 143–148.
- [26] D. Wilde, "A library for doing polyhedral operations," M.Sc. thesis, Oregon Univ., Eugene, Dec. 1993.



Florin Balasa (S'93–M'95) received the M.Sc. and Ph.D. degrees in computer science from the Polytechnical Institute of Bucharest, Romania, in 1981 and 1994, respectively. He received the M.Sc. degree in mathematics from the University of Bucharest, Romania, in 1990 and the Ph.D. degree in electrical engineering from the Katholieke Universiteit Leuven, Leuven, Belgium, in 1995.

He worked over six years at the Research Institute for Electronic Components, Bucharest, Romania. From 1990 to 1995, he worked at the VLSI System Design Methodology Division, Interuniversity Microelectronics Center (IMEC), Belgium. In the fall of 1995, he joined the CAD/CAE Department, Rockwell Semiconductor Systems, Newport Beach, CA. Since 1996, he is also a Lecturer at the University of California, Irvine. His research interests include high-level synthesis, physical design, operations research, and data dependence analysis.



Francky Catthoor (S'86–M'87) was born in Temse, Belgium, on October 1959. He received the engineering and the Ph.D. degrees in electrical engineering from the Katholieke Universiteit Leuven, Leuven, Belgium, in 1982 and 1987, respectively.

From 1983 to 1987, he was a Researcher in the area of VLSI design methodologies for digital signal processing at the Katholieke Universiteit Leuven, Leuven, Belgium. Since 1987, he has headed several research domains in the area of high-level and system synthesis techniques and architectural methodologies, all within the VLSI System Design Methodology (VSDM) Division, the Interuniversity Micro-Electronics Center (IMEC), Heverlee, Belgium. He has been an Assistant Professor of Electrical Engineering at the Katholieke Universiteit Leuven, Leuven, Belgium, since 1989. His current research activities are architecture design methods and system-level exploration for power and area, mainly oriented toward memory management and global data transfer optimization. The major target application domains are real-time signal and data processing algorithms in image, video, and end-user telecom applications, and data structure dominated modules in telecom networks. Both customized architectures and programmable multimedia processors are targeted. In these fields, he has authored or coauthored about 170 papers.

Dr. Catthoor received the Young Scientist Award from the Marconi International Fellowship in 1986. He received two Best Paper Awards. In 1995, he became an Associate Editor for the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS and since 1996, Associate Editor for Kluwer's *Journal of VLSI Signal Processing*.



Hugo J. De Man (M'81–SM'81–F'86) was born in Boom, Belgium, on September 19, 1940. He received the electrical engineering and Ph.D. degrees in applied sciences from the Katholieke Universiteit Leuven, Leuven, Belgium, in 1964 and 1968, respectively.

In 1968, he became a Member of the Staff of the Laboratory for Physics and Electronics of Semiconductors at the University of Leuven, Leuven, Belgium, working on device physics and integrated circuit technology. From 1969 until 1971 he was at the Electronic Research Laboratory, University of California, Berkeley, as an ESRO-NASA Postdoctoral Research Fellow, working on computer-aided device and circuit design. In 1971 he returned to the University of Leuven as a Research Associate of the NFWO (Belgian National Science Foundation) and in 1974 he became a Professor. During the winter-quarter of 1974-1975, he was a Visiting Associate Professor at University of California, Berkeley. Since 1984, he has been Vice-President of the VLSI System Design Group of the Interuniversity Micro-Electronics Center (IMEC), Leuven, Belgium, specializing in the research of design methodologies for integrated systems for telecommunication. Research of this group has been at the basis of EDC-Mentor Graphics DSP-station. In 1995 he became a Senior Fellow of the Interuniversity Micro-Electronics Center (IMEC), Leuven, Belgium, responsible for research in system design technologies.

Dr. De Man is a corresponding member of the Royal Academy of Sciences, Belgium, and a member of the Royal Flemish Engineering Society (KVIV). He received a Best Paper Award on bipolar device simulation at the ISSCC of 1973 and a Best Paper Award on an integrated CAD system at the 1981 ESSCIRC Conference. In 1986 he received, together with L. Claesen, the Best Paper Award in CAD from the ICCD-86 Conference and a Best Paper Award in 1987 for publication in the *International Journal of Circuit Theory and Applications*. In 1989, he received the Best Paper Award at the Design Automation Conference. He was an Associate Editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS from 1975 to 1980 and an European Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS from 1982 to 1985.