Code Generation Methods for Tiling Transformations

GEORGIOS GOUMAS, MARIA ATHANASAKI AND NECTARIOS KOZIRIS

National Technical University of Athens

Dept. of Electrical and Computer Engineering

Computing Systems Laboratory

E-mail: {goumas, mathan, nkoziris}@cslab.ece.ntua.gr

Tiling or supernode transformation has been widely used to improve locality in multi-level memory hierarchies, as well as to efficiently execute loops onto parallel architectures. However, automatic code generation for tiled loops can be a very complex compiler work due to non-rectangular tile shapes and arbitrary iteration space bounds. In this paper, we first survey code generation methods for nested loops which are transformed using non-unimodular transformations. All methods are based on Fourier-Motzkin (FM) elimination. Next, we consider and enhance previous work on rewriting tiled loops by considering parallelepiped tiles and arbitrary iteration space shapes. In order to generate tiled code, all methods first enumerate the tiles containing points within the iteration space, and second, sweep the points within each tile. For the first, we extend previous work in order to access all tile origins correctly, while for the latter, we propose the transformation of the initial parallelepiped tile iteration space into a rectangular one, so as to generate code efficiently with the aid of a non-unimodular transformation matrix and its Hermite Normal Form (HNF). The resulting systems of inequalities are much simpler than those appeared in bibliography; thus their solutions are more efficiently determined using the FM elimination. Experimental results which compare all presented approaches, show that the proposed method for generating tiled code is significantly accelerated, thus rewriting any n-D tiled loop in a much more efficient and direct way.

Keywords: loop tiling, supernodes, non-unimodular transformations, Fourier- Motzkin elimination, code generation.

1. INTRODUCTION

Linear loop transformations such as reversal, interchanging, skewing, wavefront transformation etc. are extensively used to efficiently execute nested loops by restructuring the loop execution order. Such transformations can be represented by non-singular matrices, either unimodular, or non-unimodular ones. When unimodular transformations are concerned, code generation is simply restricted to the calculation of the lower and upper bounds of the transformed loop. When dealing with non-unimodular transformations, further attention needs to be paid, in order to skip the holes that are created in the transformed iteration space. In this case, the overall complexity is aggravated by the calculation of the exact integer lower and upper bounds, as well as by the calculation of the steps of the new loop variables. Researchers have solved the problem of code generation for linear loop transformations using the Fourier-Motzkin (FM) elimination method and the properties of the Hermite Normal Form (HNF) of the transformation matrix. Ramanujam in [1] and [2] used the HNF of the transformation matrix T and Hessenberg matrices in order to correct the bounds that result from the application of Fourier-Motzkin to the transformed iteration space. The use of the HNF of T is also proposed by Xue in [3] and Fernandez et al. in [4]. Li and Pingali in [5] follow a slightly different approach which is based on the decomposition of transformation matrix T to the product of its HNF $\widetilde{T}$ and a unimodular matrix U. U transforms the original iteration space to an auxiliary iteration space, whose bounds are calculated using Fourier-Motzkin. In the sequel, the bounds of the target iteration space are easily obtained from the bounds of the auxiliary iteration space. The strides of the loop variables in all previous works are obtained from the elements in the diagonal of the HNF of the transformation matrix. Note that all code generation methods use the Fourier-Motzkin elimination method which is extremely complex, as it depends doubly exponentially on the number of nested loops. Fortunately, in practical problems the loop depth is kept small, making code generation for linear loop transformations feasible.

Tiling or supernode partitioning is another loop transformation that has been widely used to improve locality in multi-level memory hierarchies, as well as to efficiently execute loops onto distributed memory architectures. Supernode transformation groups neighboring iterations together, in order to form a large and independent computational unit, called tile or supernode. Supernode partitioning of the iteration space was first proposed by Irigoin and Triolet in [6]. They introduced the initial model of loop tiling and gave conditions for a tiling transformation to be valid. Tiles are required to be atomic, identical, bounded and their union spans the initial iteration space. In their paper ([7]), Ramanujam and Sadayappan showed the equivalence between the problem of finding a set of extreme vectors for a given set of dependence vectors and the problem of finding a tiling transformation H that produces valid, deadlock-free tiles. Since the tiling planes h_i are defined by a set of extreme vectors, they gave a linear programming formulation for the problem of finding optimal shape tiles (thus determining optimal H) that reduces communication. Boulet et al. in [8] and Xue in [9] and [10] used a communication function that has to be minimized by linear programming approaches.

When executing nested loops on parallel architectures, the key issue in loop partitioning to different processors, is to mitigate communication overhead by efficiently controlling the computation to communication ratio. In distributed memory machines, explicit message passing incurs extra time overhead due to message startup latencies and data transfer delays. In order to eliminate the communication overhead, Shang [11], Hollander [12] and others, have presented methods for dividing the index space into independent sets of iterations, which are assigned to different processors. If the rank of the dependence vector matrix is $n_d < n$, the n-dimensional loop can always be transformed to have, at maximum, $n - n_d$ outermost FOR-ALL loops [13]. However, in many cases, independent partitioning of the index space is not feasible, thus data exchanges between processors impose additional communication delays. When fine grain parallelism is concerned, several methods were proposed to group together neighboring chains of iterations [14], [15], while preserving the optimal hyperplane schedule [16], [17], [18].

When tiling for exploiting parallelism, neighboring iteration points are grouped together to build a larger computation node, which is executed by a processor. Tiles have much larger granularity than single iterations, thus reducing synchronization points and alleviating overall communication overhead. Data exchanges are grouped and performed within a single message for each neighboring processor, at the end of each atomic supernode execution. Hodzic and Shang [19] proposed a method to correlate optimal tile size and shape, based on overall completion time reduction. They applied the hyperplane transformation to loop tiles and generated a schedule where the objective is to reduce the overall time by adjusting the tile size and shape appropriately. Their approach considers a schedule where each processor executes all tiles along a specific dimension, by interleaving computation and communication phases. All processors first receive data, then compute and finally send result data to neighbors in explicitly distinct phases, according to the hyperplane scheduling vector. In [20] the overall execution time for tiled nested loops was further minimized through communication and computation overlapping. The overall schedule resembles like a pipelined datapath where computations are not any more interleaved with send and receives to non-local processors, but partly overlapped using a modified hyperplane time schedule. Tile size and shape are thus determined to reduce overall completion time under this new time scheduling scheme.

Although tiling as a loop transformation is extensively discussed, only rectangular tiling is used in real applications. However, rectangular tiling is not always legal or, as we mentioned above, communication criteria may suggest the use of non-rectangular tiles. Hodzic and Shang in [21] have shown that non-rectangular tile shapes can also lead to smaller total execution times due to more efficient scheduling schemes. On the other hand, non-rectangular loop nests very commonly appear in numerical applications. In addition, a non-rectangular iteration space may arise when a loop transformation such as skewing is applied to an original rectangular iteration space. This means that, in the general case, we have to deal with non-rectangular shapes in tiles and iteration spaces, where code generation is far from being straightforward and efficient. This is because tiling is not a linear loop transformation, which means that previous work on code generation for linear transformations cannot be directly applied on tiling. Loop rewriting in the case of tiling involves the application of the tiling transformation H, being found by the above optimal loop tiling methods, to produce an equivalent nested loop with $2n$ depth. The innermost n-D nests are representing iterations within a tile and are sequentially executed, whereas the outermost n-D nests are for-across loops executed in parallel by different processors under a time synchronization and space scheduling scheme. The objective here is to generate the outermost n-D loop nests that scan all tiles and the innermost n-D loop nests that scan all points within a tile by calculating the correct index strides and loop bounds in the transformed tile space. Thus, due to the special characteristics of tiling described above, Fourier-Motzkin elimination has to be applied in larger systems of inequalities compared to those formed by usual linear transformations, making methods so far presented impractical for problems that are more complex, as in the case of non-rectangular tiles and non-rectangular spaces.

Tang and Xue in [22] have presented a method for generating SPMD code for tiled iteration spaces. They addressed issues such as both computation and data distribution and message passing code generation. Although this is a complete approach resulting in a very straightforward loop code, it is limited only to rectangular tiles. Another approach for generating code for tiled iteration spaces was introduced by Ancourt and Irigoin in [23]. In this paper, non-rectangular tiles and iteration spaces are considered as well. However, due to the fact that tiles have parallelepiped boundary surfaces, the problem of calculating the exact transformed loop bounds is formulated as a large system of linear inequalities. These inequalities are formed first to calculate the exact bounds for every tile execution and second, to find all iterations inside each tile. The authors use Fourier-Motzkin elimination to solve the problem, but the size of the formed systems of inequalities makes Fourier-Motzkin elimination very time-consuming for loop nests with depth greater than two or three. Note that although sophisticated implementations of Fourier-Motkzin have been presented (e.g. see [24]), the method remains an extremely complex one leading to impractical compilation times in Ancourt and Irigoin's approach.

In this paper, we present an efficient way to generate code for tiled iteration spaces, considering both non-rectangular tiles and non-rectangular iteration spaces. Our goal is to reduce the size of the resulting systems of inequalities and to restrict the application of Fourier-Motzkin elimination to as fewer times as possible. We divide the main problem into the subproblems of enumerating the tiles of the iteration space and sweeping the internal points of every tile. For the first problem, we continue previous work concerning code generation for non-unimodular linear transformations in [1] and [2]. Tiling was used as an example to compute loop bounds, but the method proposed fails to enumerate all tile origins exactly. We adjust this method in order to access all tiles. As far as sweeping the internal points of every tile is concerned, we propose a novel method which uses the properties of non-unimodular transformations. The method is based on the observation that tiles are identical and that large computational complexity arises when non-rectangular tiles are concerned. To handle this fact, we first transform the parallelepiped (non-rectangular) tile (Tile Iteration Space-TIS) into a rectangular one (Transformed Tile Iteration Space-$TTIS$), sweep the derived $TTIS$ and use the inverse transformation in order to access the original points. The results are adjusted in order to sweep the internal points of all tiles, taking also into consideration the original iteration space bounds. In both cases, the resulting systems of inequalities are eliminated using Fourier-Motzkin elimination but, as it will be shown, the second application can be easily avoided. The complexity of our method is equal to that of the methods proposed for code generation of linear loop transformations. Compared to the method presented by Irigoin and Ancourt in [23], our method outperforms in terms of efficiency. Experimental results show that the procedure of generating tiled code is vastly accelerated, since the derived systems of inequalities, in our case, are smaller and can be simultaneously eliminated. In addition, the generated code is much simpler, containing less expressions and avoiding unnecessary calculations imposed by boundary tiles.

The rest of the paper is organized as follows: Basic terminology used throughout the paper and definitions from linear algebra are introduced in Section 2. We present tiling or supernode transformation in Section 3. The problem of generating code for tiled iteration spaces is defined in Section 4. Section 5 reviews previous work and proposes a new method for the problem of tile origins enumerating. Section 6 reviews previous work and proposes a new method for the problem of scanning the internal points of every tile. In Section 7 we compare our method with the one presented in [23] and present some experimental results. Finally, in Section 8 we summarize our results.

2. PRELIMINARIES

2.1 The Model of the Algorithms

In this paper we consider algorithms with perfectly nested FOR-loops that is, our algorithms are of the form:

```
FOR $j_1 = l_1$ TO $u_1$ DO
    FOR $j_2 = l_2$ TO $u_2$ DO
        ...
        FOR $j_n = l_n$ TO $u_n$ DO
            Loop Body
        ENDFOR
        ...
    ENDFOR
ENDFOR
```

where: (1) $j = (j_1, ..., j_n)$, (2) l_1 and u_1 are rational-valued parameters and (3) l_k and u_k ($k = 2...n$) are of the form: $l_k = max(\lceil f_{k1}(j_1, \ldots, j_{k-1})\rceil, \ldots, \lceil f_{kr}(j_1, \ldots, j_{k-1})\rceil)$ and $u_k = min(\lfloor g_{k1}(j_1, \ldots, j_{k-1})\rfloor, \ldots, \lfloor g_{kr}(j_1, \ldots, j_{k-1})\rfloor)$, where f_{ki} and g_{ki} are affine functions. Therefore, we are not only dealing with rectangular iteration spaces, but also with more general parameterized convex spaces, with the only assumption that the iteration space is defined as the bisection of a finite number of semi-spaces of the n-dimensional space Z^n. Each of these semi-spaces is corresponding to one of the bounds $l_k(j_1, \ldots, j_{k-1})$ or $u_k(j_1, \ldots, j_{k-1})$ of the nested for-loops.

2.2 Notation

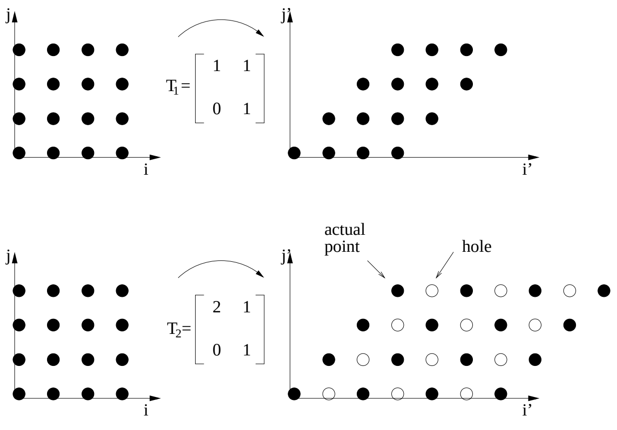
Throughout this paper the following notation is used: N is the set of naturals, Z is the set of integers, R is the set of rationals and n is the number of nested FOR-loops of the algorithm. $J^n \subset Z^n$ is the set of indices, or the iteration space of an algorithm: $J^n = \{j(j_1, ..., j_n) | j_i \in Z \wedge l_i \leq j_i \leq u_i, 1 \leq i \leq n\}$. Each point in this n-dimensional integer space is a distinct instantiation of the loop body. If A is a matrix we denote a_{ij} the matrix element in the i-th row and j-th column. We denote a vector as a or $\vec{a}$ according to the context. The k-th element of the vector is denoted a_k. In addition we define the symbols x^+ and x^- as follows: $x^+ = max(x, 0)$ and $x^- = max(-x, 0)$.

2.3 Linear Algebra Concepts

We present some basic linear algebra concepts which are used in the following sections:

Definition 1: A square matrix A is *unimodular*, if it is integral and its determinant equals to ± 1.

Unimodular transformations have a very useful property: their inverse transformation is integral as well. On the other hand the inverse of a non-unimodular matrix is not integral, which causes the transformed space to have “holes”. We call *holes* the integer points of the transformed space that have no integer anti-image in the original space.



j
i
$T_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$
j'
i'
j
i
$T_2 = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}$
actual point
hole
j'
i'

Figure 1. Unimodular and Non-Unimodular Transformations

Definition 2: Let A be a $m \times n$ integer matrix. We call the set $\mathcal{L}(A) = \{y|y = Ax \wedge x \in Z^n\}$ the *lattice* that is generated by the columns of A.

Consequently, we can define the holes of a non-unimodular transformation as follows: if T is a non-unimodular transformation, we call *holes* the points $j' \in Z^n$ such that $T^{-1}j' \notin Z^n$. On the contrary, we shall call *actual* points of a non-unimodular transformation T the points $j' \in \mathcal{L}(T) \Leftrightarrow T^{-1}j' \in Z^n$. Figure 1 shows the results of a unimodular and a non-unimodular transformation to the same index space. Holes are depicted by white dots and actual points by black ones.

Theorem 1: If T is a $m \times n$ integer matrix, and C is a $n \times n$ unimodular matrix, then $\mathcal{L}(T) = \mathcal{L}(TC)$.
Proof: Given in [1].

Definition 3: We say that a square, non-singular matrix $H = [\vec{h_1}, \ldots, \vec{h_n}] \in R^{n \times n}$ is in *Column Hermite Normal Form* (HNF) iff H is lower triangular ($h_{ij} \neq 0$ implies $i \geq j$) and for all $i > j$, $0 \leq h_{ij} < h_{ii}$ (the diagonal is the greatest element in the row and all entries are positive.)

Theorem 2: If T is a $m \times n$ integer matrix of full row rank, then there exists a $n \times n$ unimodular matrix C such that $TC = [\widetilde{T}0]$ and $\widetilde{T}$ is in Hermite Normal Form.
Proof: Given in [1].

Every integer matrix with full row rank has a unique Hermite Normal Form. By Theorem 1 we conclude that $\mathcal{L}(T) = \mathcal{L}(\widetilde{T})$ which means that an integer matrix of full row rank and its HNF produce the same lattice. This property will be proven to be very useful for the code generation of tiled spaces.

2.4 Fourier-Motzkin Elimination Method

The Fourier-Motzkin elimination method can be used to test the consistency of a system of linear inequalities $A\vec{x} \leq \vec{a}$, or to convert this system into a form in which the lower and upper bounds of each element x_i of the vector $\vec{x}$ are expressed only in terms of the variables $x_1, \ldots, x_{i-1}$. This fact is very important when using a nested loop in order to traverse an iteration space J^n defined by a system of inequalities. It is evident that the bounds of index j_k of the nested loop should be expressed in terms only of the $k-1$ outer indices. This means that Fourier-Motzkin elimination can convert a system describing a general iteration space into a form suitable for use in nested loops.

The method is based on the observation that a variable x_i can be eliminated from such a system by replacing each pair-wise combination of two inequalities which define a lower and an upper bound on x_i as follows: $\left.\begin{array}{c} L \leq c_1 x_i \\ c_2 x_i \leq U \end{array}\right\} \Rightarrow c_2 L \leq c_1 U$ with $c_1 > 0$ and $c_2 > 0$. After this elimination, another system of inequalities is derived, without x_i. This resulting system has a large number of inequalities involving variable x_i, but not all of them are necessary for the precision of the corresponding bounds. The unnecessary inequalities should be eliminated in order to simplify the resulting system. In order to specify the redundant inequalities two methods are proposed: the “Ad-Hoc simplification method” and the “Exact simplification method”. A full description of the Fourier-Motzkin elimination method, the Ad-Hoc simplification and the Exact simplification method is presented in [25].

If the initial system of inequalities consists of k inequalities with n variables, then the complexity of Fourier-Motzkin is $O(\frac{k^{2^n}}{2^{2^{(n+1)}-2}}) \approx O((\frac{k}{2})^{2^n})$. Therefore, it depends *doubly exponentially* on the number of variables involved.

3. TILING TRANSFORMATION

In a tiling transformation the index space J^n is partitioned into identical n-dimensional parallelepiped areas (tiles or supernodes) formed by n independent families of parallel hyperplanes. Tiling transformation is uniquely defined by the n-dimensional square matrix H. Each row vector of H is perpendicular to one family of hyperplanes forming the tiles. Dually, tiling transformation can be defined by n linearly independent vectors, which are the sides of the tiles. Similar to matrix H, matrix P contains the side-vectors of a tile as column vectors. It holds $P = H^{-1}$. Formally, tiling transformation is defined as follows:

$$r : Z^n \longrightarrow Z^{2n}, r(j) = \left[\begin{array}{c} \lfloor Hj \rfloor \\ j - H^{-1} \lfloor Hj \rfloor \end{array} \right]$$

where $\lfloor Hj \rfloor$ identifies the coordinates of the tile that iteration point $j(j_1, j_2, \ldots, j_n)$ is mapped to and $j - H^{-1}\lfloor Hj \rfloor$ gives the coordinates of j within that tile relative to the tile origin. Thus the initial n-dimensional iteration space J^n is transformed to a $2n$-dimensional one, the space of tiles and the space of indices within tiles. Apparently, tiling transformation is not a linear transformation. The following spaces are derived when tiling transformation H is applied to an iteration space J^n.

1. The Tile Iteration Space $TIS(H) = \{j \in Z^n | 0 \leq \lfloor Hj \rfloor < 1\}$, which contains all points that belong to the tile starting at the axes origins.

2. The Tile Space $J^S(J^n, H) = \{j^S | j^S = \lfloor Hj \rfloor, j \in J^n\}$, which contains the images of all points $j \in J^n$ according to tiling transformation.

3. The Tile Origin Space $TOS(J^S, H^{-1}) = \{j \in Z^n | j = H^{-1}j^S, j^S \in J^S\}$, which contains the origins of tiles in the original space.

According to the above it holds: $J^n \xrightarrow{H} J^S$ and $J^S \xrightarrow{P} TOS$. For simplicity reasons we shall refer to $TIS(H)$ as TIS, $J^S(J^n, H)$ as J^S and $TOS(J^S, H^{-1})$ as TOS. Note that all points of J^n that belong to the same tile are mapped to the same point of J^S. This means that a point j^S in this n-dimensional integer space J^S is a distinct tile with coordinates $(j_1^S, j_2^S, \ldots, j_n^S)$. Note also that TOS is not necessarily a subset of J^n, since there may exist tile origins which do not belong to the original index space J^n, but some iterations within these tiles do belong to J^n. The following example analyzes the properties of each of the spaces defined above.

Example 1: Consider a tiling transformation defined by $H = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{bmatrix}$ or equivalently by $P = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ applied to index space $J^2 = \{0 \leq j_1 \leq 6, 0 \leq j_2 \leq 4\}$. Then, as shown in Figure 2, TIS contains points $\{(0,0), (1,1), (2,2)\}$ and J^n is transformed by matrix H to Tile Space $J^S = \{(-2,3), (-2,2), (-1,2), \ldots, (3,-1), (3,-2), (4,-2), (4,-3)\}$. In the sequel, the Tile Space J^S is transformed by matrix P to $TOS = \{(-1,4), (-2,2), (0,3), \ldots, (5,1), (4,-1), (6,0), (5,-2)\}$. The points of TOS are shown in bold dots. $\square$

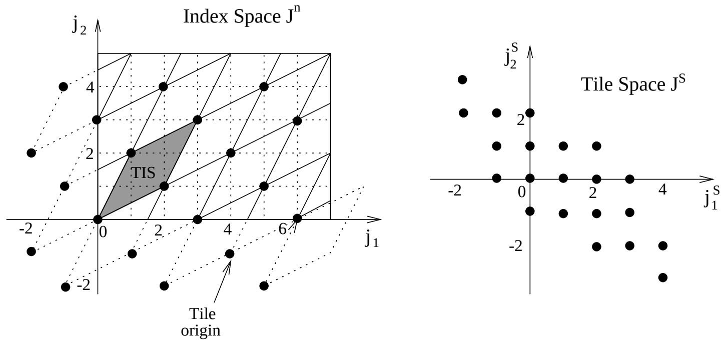


Figure 2. J^S, TIS and TOS from Example 1

The iteration space J^n of an algorithm, as defined in Section 2.2, can also be represented by a system of linear inequalities. An inequality of this system expresses a boundary surface of our iteration space. Thus J^n can be equivalently defined as: $J^n = \{j \in Z^n | Bj \leq \vec{b}\}$. Matrix B and vector $\vec{b}$ can be easily derived from l_k and u_k in Section 2.1. and vice versa. Similarly, points belonging to the same tile with tile origin

$j_0 \in TOS$ satisfy the system of inequalities $0 \leq H(j - j_0) < 1$. In order to deal with integer inequalities, we define g to be the minimum positive integer such as gH is an integer matrix. Thus, we can rewrite the above system of inequalities as follows: $0 \leq gH(j - j_0) < g \Leftrightarrow 0 \leq gH(j - j_0) \leq (g - 1)$ We denote $S = \begin{pmatrix} gH \\ -gH \end{pmatrix}$ and $\vec{s} = \begin{pmatrix} (g-1)\vec{1} \\ \vec{0} \end{pmatrix}$. Equivalently the above system becomes: $S(j - j_0) \leq \vec{s}$. Note that if $j_0 = 0$, $S(j - j_0) \leq \vec{s}$ is satisfied only by points in TIS.

Example 2: Consider the following nested loop:

```
FOR $j_1 = 0$ TO 39 DO
    FOR $j_2 = 0$ TO 29 DO
        $A[j_1, j_2] = A[j_1 - 1, j_2 - 2] + A[j_1 - 3, j_2 - 1]$
    ENDFOR
ENDFOR
```

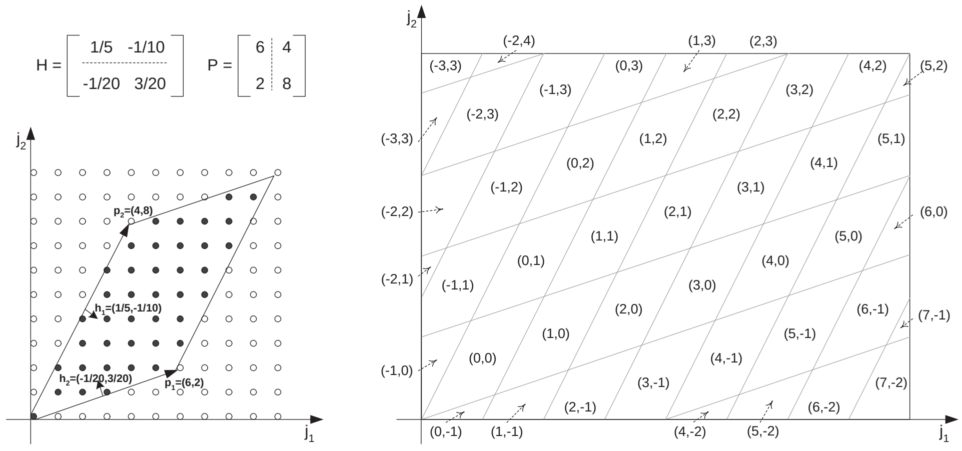


Figure 3. Tiling with matrices H and P

The index space J^2 is: $J^2 = \{(j_1, j_2) | 0 \leq j_1 \leq 39, 0 \leq j_2 \leq 29\}$. The set of inequalities describing the index space J^2 is: $\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} j_1 \\ j_2 \end{pmatrix} \leq \begin{pmatrix} 39 \\ 29 \\ 0 \\ 0 \end{pmatrix}$. Let us consider, without lack of generality, for tiling transformation matrix $H = \begin{bmatrix} \frac{1}{5} & -\frac{1}{10} \\ -\frac{1}{20} & \frac{3}{20} \end{bmatrix}$. Consequently, we have $H^{-1} = P = \begin{bmatrix} 6 & 4 \\ 2 & 8 \end{bmatrix}$. The system of inequalities $S(j - j_0) \leq \vec{s}$ describing a tile is (since $g = 20$):

$\begin{pmatrix} 4 & -2 \\ -1 & 3 \\ -4 & 2 \\ 1 & -3 \end{pmatrix} \begin{pmatrix} j_1 - j_{0_1} \\ j_2 - j_{0_2} \end{pmatrix} \leq \begin{pmatrix} 19 \\ 19 \\ 0 \\ 0 \end{pmatrix}$. Figure 3 shows the form of a tile, the partition of the index space J^n into tiles and the coordinates of every tile according to the tiling transformation, or equivalently the coordinates of every tile point in the Tile Space J^S □

4. CODE GENERATION

Let us formally define the problem of generating tiled code that will traverse an iteration space J^n transformed by a tiling transformation H. Applying this transformation to J^n, we obtain the Tile Space J^S, TIS and TOS. In Section 3 it is shown that tiling transformation is a $Z^n \longrightarrow Z^{2n}$ transformation, which means that a point $j \in J^n$ is transformed into a tuple of n-dimensional points (j_a, j_b), where j_a identifies the tile that the original point belongs to ($j_a \in J^S$) and j_b the coordinates of the point relevant to the tile origin ($j_b \in TIS$). The tiled code reorders the execution of indices imposed by the original nested loop, resulting in a rearranged, transformed order described by the following scheme: for every tile in the Tile Space J^S, traverse its internal points. According to the above, the tiled code should consist of a $2n$-dimensional nested loop. The n outermost loops traverse the Tile Space J^S, using indices $j_1^S, j_2^S, \ldots, j_n^S$, and the n innermost loops traverse the points within the tile defined by $j_1^S, j_2^S, \ldots, j_n^S$, using indices $j_1', j_2', \ldots, j_n'$. We denote l_k^S, u_k^S the lower and upper bounds of index j_k^S respectively. Similarly, we denote l_k', u_k' the lower and upper bounds of index j_k'. In all cases, lower bounds L_k are of the form: $L_k = max(l_{k,0}, l_{k,1}, \ldots)$ and upper bounds U_k of the form: $U_k = min(u_{k,0}, u_{k,1}, \ldots)$, where $l_{k,j}$, $u_{k,j}$ are affine functions of the outermost indices. Code generation involves the calculation of steps (loop strides) and exact lower and upper bounds for indices j_k^S and j_k'. The problem of generating tiled code for an iteration space can be separated into two subproblems: traversing the Tile Space J^S and sweeping the internal points of every tile or, in our context, finding lower and upper bounds for the n outermost indices $j_1^S, j_2^S, \ldots, j_n^S$ and finding lower and upper bounds for the n innermost indices $j_1', j_2', \ldots, j_n'$.

5. TRAVERSING THE TILE SPACE

In this section we elaborate on the subproblem of traversing the Tile Space J^S. We first review previous work in the field and, in the sequel, we extend one of the approaches in order to efficiently and correctly traverse the Tile Space.

5.1 Previous Work

The method presented by Ancourt and Irigoin in [23] traverses the Tile Space J^S by constructing a system of inequalities which consists of one system representing the original index space and one system representing a tile. Recall from Section 3 that a point $j \in J^n$ that belongs to a tile with tile origin $j_0 \in TOS$, satisfies the set of inequalities: $S(j - j_0) \leq \vec{s}$. Let us denote $j_0^S \in J^S$ the coordinates of j_0 in the Tile Space J^S. Clearly it holds $j_0 = Pj_0^S$. Consequently, the preceding system of inequalities becomes:

$\begin{pmatrix} -gI & gH \\ gI & -gH \end{pmatrix} \begin{pmatrix} j_0^S \\ j \end{pmatrix} \leq \vec{s}$. Recall also that a point $j \in J^n$ satisfies the system of inequalities $Bj \leq \vec{b}$. Combining these systems we obtain the final system of inequalities:

$$\begin{pmatrix} 0 & B \\ -gI & gH \\ gI & -gH \end{pmatrix} \begin{pmatrix} j_0^S \\ j \end{pmatrix} \leq \begin{pmatrix} \vec{b} \\ \vec{s} \end{pmatrix} \tag{1}$$

Ancourt and Irigoin propose the application of Fourier-Motzkin elimination to the above system in order to obtain proper formulas for the lower and upper bounds of the n-dimensional loop that will traverse the Tile Space. Although this method scans the tile origins correctly, the elimination of the above system is impractical even for loops with small nestings.

Ramanujam in [1] and [2] applied tiling transformation to the set of inequalities $Bj \leq \vec{b}$ representing the iteration space as follows: $Bj \leq \vec{b} \Rightarrow BH^{-1}Hj \leq \vec{b} \Rightarrow$

$$BPj_0^S \leq \vec{b} \tag{2}$$

Here, again, the application of Fourier-Motzkin elimination to the derived system of inequalities is proposed, in order to obtain closed form formulas for tile bounds $l_1^S, \ldots, l_n^S$ and $u_1^S, \ldots, u_n^S$.

This approach is more elegant since the system constructed has the same size as the one that arises when any linear transformation is concerned. Unfortunately, it fails to enumerate tile origins exactly. Problems arise because the method is applicable to integral matrices, while H is not integral. Note that the system of inequalities in (2) is satisfied by points in the Tile Space J^S whose inverse belong to J^n. However, as stated in Section 3 there exist some points in TOS that do not belong to J^n. In Figure 3, tiles in the lower boundaries such as (-3,3), (-2,1), (4,-2) and others are not scanned by this method because their origins do not belong to the original index space J^n. Nevertheless, these tiles also contain points of the index space J^2 and should be traversed, although they do not satisfy the preceding systems of inequalities.

5.2 Our Approach

We shall now extend the work presented in [1] and [2] in order to be also applicable to the enumeration of tiles. We modify the system in (2) in order to include all tile origins. What is needed is a proper reduction of the lower bounds and/or a proper increase of the upper bounds of our space. Lemma 1 determines how much we should expand space bounds, in order to include all points of TOS.

Lemma 1: If we apply tiling transformation P to an index space J^n, whose bounds are expressed by the system of inequalities $Bj \leq \vec{b}$, then for all tile origins $j_0 \in TOS$ it holds:

$$Bj_0 \leq \vec{b'} \tag{3}$$

where $\vec{b'}$ is a n-dimensional vector formed by the vector $\vec{b}$ so that its i-th element is given by the equation:

$$b'_i = b_i + \frac{g-1}{g} \sum_{r=1}^{n} (\sum_{j=1}^{n} \beta_{ij} p_{jr})^- \tag{4}$$

where g is the minimum integer number by which the tiling matrix H should be multiplied in order to become integral.

Proof: Suppose that the point $j \in J^n$ belongs to the tile with origin j_0. Then j can be expressed as the sum of j_0 and a linear combination of the column-vectors of the tiling matrix P: $j = j_0 + \sum_{j=1}^{n} \lambda_j \vec{p_j}$. In addition, as aforementioned, the following equality holds: $\vec{0} \leq gH(j - j_0) \leq (g \vec{-} 1)$. The i-th row of this inequality can be rewritten as follows:$0 \leq \vec{h_i}(j - j_0) \leq \frac{g-1}{g}$, where $\vec{h_i}$ is the i-th row-vector of matrix $H = P^{-1}$. Therefore: $0 \leq \vec{h_i} \sum_{j=1}^{n} \lambda_j \vec{p_j} \leq \frac{g-1}{g}$. As $P = H^{-1}$ it holds that $\vec{h_i}\vec{p_i} = 1$ and $\vec{h_i}\vec{p_j} = 0$ if $i \neq j$. Consequently the last formula can be rewritten as follows: $0 \leq \lambda_i \leq \frac{g-1}{g}$ for all $i = 1, \ldots, n$.

For each $j \in J^n$, it holds $Bj \leq \vec{b}$. The k-th row of this system can be written as follows: $\sum_{j=1}^{n} \beta_{kj} j_j \leq b_k$ We can rewrite the last inequality in terms of the corresponding tile origin as follows: $\sum_{j=1}^{n} \beta_{kj} (j_{0j} + \sum_{i=1}^{n} \lambda_i p_{ji}) \leq b_k \Rightarrow \sum_{j=1}^{n} \beta_{kj} j_{0j} \leq b_k - \sum_{j=1}^{n} \beta_{kj} \sum_{i=1}^{n} \lambda_i p_{ji}$

$$\Rightarrow \sum_{j=1}^{n} \beta_{kj} j_{0j} \leq b_k - \sum_{i=1}^{n} \lambda_i \sum_{j=1}^{n} \beta_{kj} p_{ji} \tag{5}$$

In addition, as proved above, it holds $0 \leq \lambda_i \leq \frac{g-1}{g}$ for all $i = 1, \ldots, n$. If multiplied by $\sum_{j=1}^{n} \beta_{kj} p_{ji}$ this inequality gives:

a) If $\sum_{j=1}^{n} \beta_{kj} p_{ji} > 0$: $0 \leq \lambda_i \sum_{j=1}^{n} \beta_{kj} p_{ji} \leq \frac{g-1}{g} \sum_{j=1}^{n} \beta_{kj} p_{ji}$

b) If $\sum_{j=1}^{n} \beta_{kj} p_{ji} < 0$: $\frac{g-1}{g} \sum_{j=1}^{n} \beta_{kj} p_{ji} \leq \lambda_i \sum_{j=1}^{n} \beta_{kj} p_{ji} \leq 0$

Using the symbols x^+ and x^- given in Section 2.2, the previous inequalities can be rewritten as follows: $-\frac{g-1}{g} (\sum_{j=1}^{n} \beta_{kj} p_{ji})^- \leq \lambda_i \sum_{j=1}^{n} \beta_{kj} p_{ji} \leq \frac{g-1}{g} (\sum_{j=1}^{n} \beta_{kj} p_{ji})^+ \Rightarrow -\lambda_i \sum_{j=1}^{n} \beta_{kj} p_{ji} \leq \frac{g-1}{g} (\sum_{j=1}^{n} \beta_{kj} p_{ji})^-$. If added for $i = 1, \ldots, n$ this inequality gives: $-\sum_{i=1}^{n} \lambda_i \sum_{j=1}^{n} \beta_{kj} p_{ji} \leq \frac{g-1}{g} \sum_{i=1}^{n} (\sum_{j=1}^{n} \beta_{kj} p_{ji})^-$.

Therefore, from the last formula and the inequality (5), we conclude that $\sum_{j=1}^{n} \beta_{kj} j_{0j} \leq b_k + \frac{g-1}{g} \sum_{i=1}^{n} (\sum_{j=1}^{n} \beta_{kj} p_{ji})^-$. Thus, for each tile with origin j_0 which has at least one point in the initial iteration space, it holds that $Bj_0 \leq \vec{b'}$, where the vector $\vec{b'}$ is constructed so as its kth element is given by the form: $b'_k = b_k + \frac{g-1}{g} \sum_{i=1}^{n} (\sum_{j=1}^{n} \beta_{kj} p_{ji})^-$. $\dashv$

If we consider Tile Space J^S then, since $j_0 = Pj_0^S$, we equivalently get the system of inequalities

$$BPj_0^S \leq \vec{b'} \tag{6}$$

Thus we can adjust the system of inequalities in (2) making use of Lemma 1 and replacing the vector $\vec{b}$ with $\vec{b'}$.

Geometrically the term added to each element of $\vec{b}$ expresses a parallel shift of the corresponding bound of the initial space. In Figure 4 we present an example of our method. Each row $\vec{\beta_i}$ of the matrix B expresses a vector vertical to the corresponding bound of

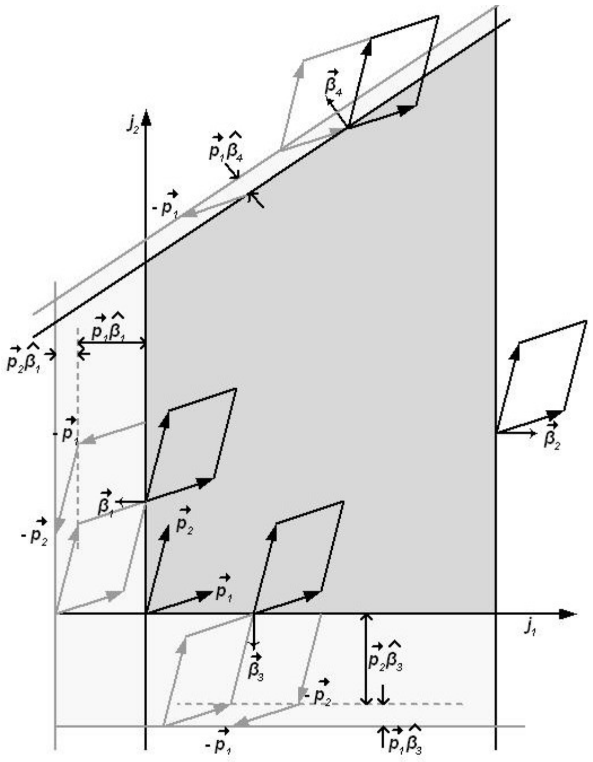


Figure 4. Expanding bounds to include all tile origins

the iteration space and its direction is outwards. The equation of this boundary surface is $\vec{\beta_i}\vec{x} = b_i$. A parallel shift of this surface by a vector $\vec{x_0}$ is expressed by the equation $\vec{\beta_i}(\vec{x} - \vec{x_0}) = b_i \Leftrightarrow \vec{\beta_i}\vec{x} = b_i + \vec{\beta_i}\vec{x_0}$. As shown in Figure 4, we shift a boundary surface by a tile edge-vector $\vec{p_r}$, if this vector forms an angle greater than 90^o with vector $\vec{\beta_i}$ (as the angles between the vectors $\vec{\beta_1}$ and $\vec{p_1}$, $\vec{\beta_1}$ and $\vec{p_2}$, $\vec{\beta_3}$ and $\vec{p_1}$, $\vec{\beta_3}$ and $\vec{p_2}$, $\vec{\beta_4}$ and $\vec{p_1}$ in Figure 4), or equivalently if and only if $\vec{p_r}\vec{\beta_i} < 0$. This fact can be expressed as follows: if the dot product of one of the columns of the matrix P, $\vec{p_r}$ and a row of B, $\vec{\beta_i}$ is negative, then we should subtract this dot product from the constant b_i. In formula (4) we add the term $(\sum_{j=1}^{n} \beta_{ij}p_{jr})^- = (\vec{\beta_i}\vec{p_r})^-$ to the constant b_i for all vectors $\vec{p_r}$. The factor $\frac{g-1}{g}$ by which the shifting constant is multiplied, expresses the fact that a tile is a semiopen hyperparallelepiped and thus we need not contain in the tile space the tiles which just touch the initial iteration space. Note, however, that this expansion of bounds may include some redundant tiles, whose origins belong to the extended space but their internal points remain outside the original iteration space. These tiles will be accessed but their internal points will not be swept, as it will be shown next, thus imposing little computation overhead in the execution of the tiled code. Consequently, we succeeded to correctly access all tile origins with a small system of inequalities that can be efficiently eliminated for practical problems.

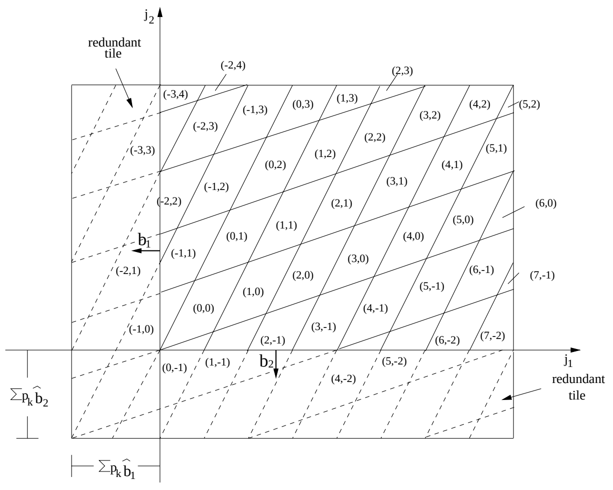


Figure 5. Example 3: Expansion of bounds

Example 3: We shall enumerate the tiles of the algorithm shown in Example 2. We consider the same tiling transformation. Following our approach, we construct the system of inequalities in (6) making use of the expression in (4). Expression (4) in our case gives $\vec{b'} = \begin{bmatrix} 39 & 29 & 9.5 & 9.5 \end{bmatrix}^T$ and thus the system in (6) becomes: $\begin{pmatrix} 6 & 4 \\ 2 & 8 \\ -6 & -4 \\ -2 & -8 \end{pmatrix} \begin{pmatrix} j_1^S \\ j_2^S \end{pmatrix} \leq \begin{pmatrix} 39 \\ 29 \\ 9.5 \\ 9.5 \end{pmatrix}$. The expansion of bounds for this example is shown in Figure 5. If we multiply row 1 by 2 and add it to row 4 we get $10j_1^S \leq 87.5 \Rightarrow j_1^S \leq 8$. Similarly we get $j_1^S \geq -4$ (this corresponds to a rough application of FM). Consequently a loop that enumerates the origins of tiles in our case has the form:

```
FOR $j_1^S = -4$ TO 8 DO
    FOR $j_2^S = MAX(\lceil \frac{-9.5-6j_1^S}{4} \rceil, \lceil \frac{-9.5-2j_1^S}{8} \rceil)$ TO $MIN(\lfloor \frac{39-6j_1^S}{4} \rfloor, \lfloor \frac{29-2j_1^S}{8} \rfloor)$ DO
        ...
    ENDFOR
ENDFOR
```

Note that tiles $(8, -3)$ and $(-4, 4)$ are redundant. □

6. SCANNING THE INTERIOR OF A TILE

6.1 Previous Work

Traversing the internal points of a tile is also discussed by Ancourt and Irigoin in [23]. Similarly to the approach used there to enumerate tile origins, Ancourt and Irigoin construct a proper system of inequalities, which they eliminate using Fourier-Motzkin elimination.

The inequalities describing the original index space and the tile can be equivalently rewritten as:

$$\begin{pmatrix} B \\ gH \\ -gH \end{pmatrix} j \leq \begin{pmatrix} \vec{b} \\ (g-1)\vec{1} + gj_0^S \\ \vec{0} - gj_0^S \end{pmatrix} \tag{7}$$

where vector j_0^S gives the coordinates of the tile to be traversed. Again here, the size of the system is too big, making Fourier-Motzkin elimination inefficient for practical problems.

6.2 Our Approach

In the following, we present a new method to sweep the internal points of a tile. Our goal is to construct a smaller system of inequalities that can be efficiently eliminated. Our approach is based on the use of a non-unimodular transformation. We shall traverse the TIS and then slide the points of TIS properly, so as to scan all points of J^n. In order to achieve this, we transform the TIS to a rectangular space, called the Transformed Tile Iteration Space ($TTIS$). We traverse the $TTIS$ with an n-dimensional nested loop and then transform the indices of the loop, so as to return to the proper points of the TIS. In other words, we are searching for a transformation pair (P', H'): $TTIS \xrightarrow{P'} TIS$ and $TIS \xrightarrow{H'} TTIS$ (Fig. 6). Intuitively, we demand P' to be parallel to the tile sides, that is the column vectors of P' should be parallel to the column vectors of P. This is equivalent to the row vectors of H' being parallel to the row vectors of H. In addition, we demand the lattice of H' to be an integer space in order for loop indices to be able to traverse it. Formally, we are searching for an n-dimensional transformation $H' : H' = VH$, where V is a $n \times n$ diagonal matrix and $\mathcal{L}(H') \subseteq Z^n$. The following Lemma proves that the second requirement is satisfied if and only if H' is integral.

Lemma 2: If $j' = Aj, j \in Z^n$ then $j' \in Z^n$ iff A is integral.
Proof: If A is integral it is clear that $j' \in Z^n \forall j \in Z^n$. Suppose that $j' \in Z^n \forall j \in Z^n$. We shall prove that A is integral. Without lack of generality we select $j = \hat{u}_k$, where $\hat{u}_k$ is the k-th unitary vector, $\hat{u}_k = (u_{k1}, \ldots, u_{kn})$, $u_{kk} = 1, u_{kj} = 0, j \neq k$. Then according to the above $A\hat{u}_k = [\sum_{i=1}^n a_{1i}u_{k_i}, \sum_{i=1}^n a_{2i}u_{k_i}, \ldots, \sum_{i=1}^n a_{ni}u_{k_i}]^T = [a_{1k}, a_{2k}, \ldots, a_{nk}]^T \in Z^n$. This holds for all $\hat{u}_k, k = 1 \ldots n$, thus A is integral. $\dashv$

Let us construct V in the following way: every diagonal element v_{kk} is the smallest integer such that $v_{kk}h_k$ is integral, where h_k is the k-th row of matrix H. Thus both requirements for H' are satisfied. It is obvious that H' is a non-unimodular transformation. This means that the Transformed Tile Iteration Space contains holes. In Figure 6, the holes in the $TTIS$ are depicted by white dots, while the actual points are depicted by black dots. So, in order to traverse the TIS, we have to scan all actual points of the $TTIS$ and then transform them back using matrix P'. For the code generation we shall use the same notion as in [1], [3], [5] and [4]. However, we shall avoid the application of the Fourier-Motzkin elimination method to calculate the bounds of the $TTIS$ by taking advantage of the tile shape regularity.

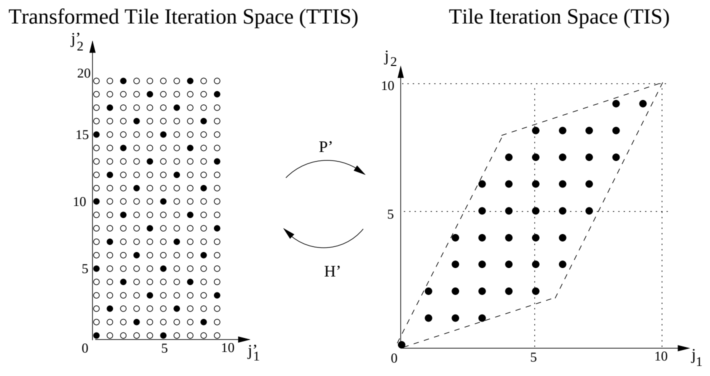

Transformed Tile Iteration Space (TTIS)
Tile Iteration Space (TIS)
j'2
j'1
P'
H'
j2
j1
0
5
10
15
20

Figure 6. Traverse the TIS with a non-unimodular transformation

We use a n-dimensional nested loop with iterations indexed by $j'(j'_1, j'_2, \ldots, j'_n)$ in order to traverse the actual points of the $TTIS$. The upper bounds of the indices j'_k are easily determined: it holds $j'_k \leq v_{kk} - 1$. However, the increment step c_k of an index j'_k is not necessarily 1. In addition to this, if index j'_k is incremented by c_k, then all indices $j'_{k+1}, \ldots, j'_n$ should be initialized at certain offset values $a_{(k+1)k}, \ldots, a_{nk}$. Suppose that for a certain index vector j', it holds $P'j' \in Z^n$. The first question is how much to increment the innermost index j'_n so that the next swept point is also integral. Formally, we search the minimum $c_n \in Z$ such that $P' \begin{bmatrix} j'_1 & j'_2 & \ldots & j'_n + c_n \end{bmatrix}^T \in Z^n$. After determining c_n, the next step is to calculate the increment step of index j'_{n-1} so that the next swept point is also integral. In this case, it is possible that index j'_n should also be augmented by an offset $a_{n(n-1)} : 0 \leq a_{n(n-1)} < c_n$. In the general case of index j'_k we need to determine $c_k, a_{(k+1)k}, \ldots, a_{nk}$ so that: $P' \begin{bmatrix} j'_1 & \ldots & j'_k + c_k & j'_{k+1} + a_{(k+1)k} & \ldots & j'_n + a_{nk} \end{bmatrix}^T \in Z^n$. Every index j'_k has $k-1$ different offsets a_{ki}, depending on each of the increment steps c_i of the $k-1$ outer indices $j'_i, i = 1 \ldots k-1$. These offsets are $a_{k1}, \ldots, a_{k(k-1)}$. The following Lemma 3 proves that increment steps c_k and offsets a_{kl}, ($k = 1 \ldots n$ and $l = 1 \ldots k-1$), are directly obtained from the Hermite Normal Form of matrix H', denoted $\widetilde{H'}$.

Lemma 3: If $\widetilde{H'}$ is the column HNF of H' and $j'(j'_1, j'_2, \ldots, j'_n)$ is the index vector used to traverse the actual points of $\mathcal{L}(H')$, then the increment step (stride) for index j'_k is $c_k = \widetilde{h'}_{kk}$ and the offsets are $a_{kl} = \widetilde{h'}_{kl}$, ($k = 1 \ldots n$ and $l = 1 \ldots k-1$).

Proof: It holds $\mathcal{L}(H') = \mathcal{L}(\widetilde{H'})$ and $\vec{0} \in \mathcal{L}(H')$. In addition to this, the columns of $\widetilde{H'}$ belong to $\mathcal{L}(H')$. Suppose $\vec{x} \in Z^n/\vec{0}$ with the following properties: $x_i = 0$ for $i < k$ and $0 \leq x_i \leq \widetilde{h'}_{ik}$ for $k \leq i \leq n$. It suffices to prove that $\vec{x} \notin \mathcal{L}(H')$. Suppose that $\vec{x} \in \mathcal{L}(H')$ which means that $\exists j \in Z^n : \widetilde{H'}j = \vec{x}$. $\widetilde{H'}$ is a lower triangular non-negative matrix and thus it holds: $x_1 = \widetilde{h'}_{11}j_1 = 0 \Rightarrow j_1 = 0$. Similarly $j_i = 0$ for $i < k$. In addition it holds: $x_k = \widetilde{h'}_{kk}j_k$. According to the above it should hold $0 \leq x_k = \widetilde{h'}_{kk}j_k \leq \widetilde{h'}_{kk} \Rightarrow 0 \leq j_k \leq 1$. In addition $0 \leq x_{k+1} = \widetilde{h'}_{(k+1)k}j_k + \widetilde{h'}_{(k+1)(k+1)}j_{k+1} \leq \widetilde{h'}_{(k+1)k}$. Since $\widetilde{h'}_{(k+1)(k+1)} \geq \widetilde{h'}_{(k+1)k} \Rightarrow j_{k+1} = 0$. Similarly $j_i = 0$ for $i > k+1$. Consequently either $\vec{x} = \vec{0}$ which is a contradiction, or $\vec{x}$ is the $k-th$ column of $\widetilde{H'}$. $\dashv$

According to the above analysis, the point that will be traversed using the next instantiation of indices is calculated from the current instantiation, since steps and offsets are added to the current indices. There are two remarks that have to be made on that: First, the indices of the loop must be initialized to the values: $j'_2 = -a_{21}, \ldots, j'_n = -a_{n1}$. Second, the loop must choose the appropriate offset for index j'_k among $a_{k1}, \ldots, a_{k(k-1)}$. Recall that a_{kl} expresses the offset of index j'_k when index j'_l ($l < k$) is incremented by its step c_l. The following form of loops suggests a solution to this problem using an extra variable *PHASE* showing the index that has just been incremented. The mod expression is used to traverse the minimum point in the particular dimension.

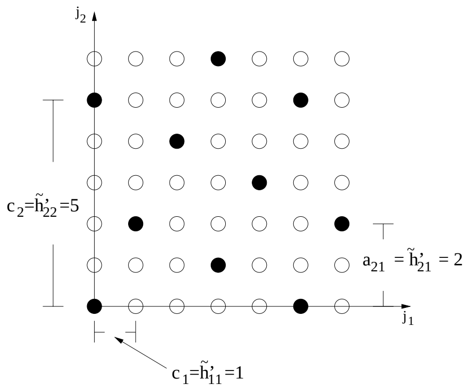


Figure 7. Steps and offsets in $TTIS$ derived from matrix $\widetilde{H'}$

Theorem 3: The following n-dimensional nested loop traverses all points $j' \in TTIS$

```
$j'_2 = -\widetilde{H'}[2][1]; \ldots, \widetilde{H'}[n][1]; PHASE = 1;$
FOR $j'_1 = 0$ TO $v_{11} - 1$ $STEP = c_1$ $PHASE = 1$ DO
    FOR $j'_2 = (j'_2 + \widetilde{H'}[2][PHASE])\%c_2$ TO $v_{22} - 1$ $STEP = c_2$ $PHASE = 2$ DO
        ...
        FOR $j'_n = (j'_n + \widetilde{H'}[n][PHASE])\%c_n$ TO $v_{nn} - 1$ $STEP = c_n$ DO
            ...
        ENDFOR
        ...
    ENDFOR
ENDFOR
```

Proof: It can be easily derived from Lemmas 2 and 3.

We now need to adjust the above loop, which sweeps all points in $TTIS$, in order to traverse the internal points of any tile in J^S. If $j' \in TTIS$ is the point derived from the indices of the former loop and $j^S \in J^S$ is the tile whose internal points $j \in J^n$ we want to traverse, it will hold: $j = Pj^S + P'j' = j_0 + P'j'$, $j_0 \in TOS$, where $j_0 = Pj^S$ is the tile origin, and $P'j' \in TIS$ the corresponding to j' point in TIS. Special attention needs to be given so that the points traversed do not outreach the original space boundaries. As we have mentioned, a point $j \in J^n$ satisfies the following set of inequalities: $Bj \leq \vec{b}$. Replacing j by the above equation we have: $B(j_0 + P'j') \leq \vec{b} \Rightarrow$

$$BP'j' \leq \vec{b} - Bj_0 \tag{8}$$

Applying Fourier-Motzkin elimination to the preceding set of inequalities, we obtain proper expressions for j', so as to not cross the original space boundaries. In this way, the problem of redundant tiles that arose in the previous section is also faced, since no computation is performed in these tiles.

Example 4: Continuing the previous examples, we shall now sweep the internal points of a tile. If we follow our method we have the following: $H' = \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix}$ and $V = \begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix}$. Accordingly $P' = \begin{bmatrix} \frac{3}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{2}{5} \end{bmatrix}$. The Hermite Normal Form of matrix H' is $\widetilde{H'} = \begin{bmatrix} 1 & 0 \\ 2 & 5 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$ and thus, as shown in Figure 7, $c_1 = 1$, $c_2 = 5$ $a_{21} = 2$. We construct matrix $[BP'|\vec{b}|B] = \begin{pmatrix} \frac{3}{5} & \frac{1}{5} & 39 & 1 & 0 \\ \frac{1}{5} & \frac{2}{5} & 29 & 0 & 1 \\ -\frac{3}{5} & -\frac{1}{5} & 0 & -1 & 0 \\ -\frac{1}{5} & -\frac{2}{5} & 0 & 0 & -1 \end{pmatrix}$. FM elimination on this matrix gives: $\begin{pmatrix} 1 & 0 & 78 & 2 & -1 \\ \frac{3}{5} & \frac{1}{5} & 39 & 1 & 0 \\ \frac{1}{5} & \frac{2}{5} & 29 & 0 & 1 \\ -1 & 0 & 29 & -2 & 1 \\ -\frac{3}{5} & -\frac{1}{5} & 0 & -1 & 0 \\ -\frac{1}{5} & -\frac{2}{5} & 0 & 0 & -1 \end{pmatrix}$. Consequently, the loop that traverses the indices inside every tile, in our case is:

```
$\begin{pmatrix} j_{01} \\ j_{02} \end{pmatrix} = \begin{bmatrix} 6 & 4 \\ 2 & 8 \end{bmatrix} \begin{pmatrix} j_1^S \\ j_2^S \end{pmatrix}$
$j'_2 = -2$
FOR $j'_1 = MAX(0, -29 - 2j_{0_1} + j_{0_2})$ TO $MIN(9, 78 - 2j_{0_1} + j_{0_2})$ $STEP = 1$ DO
  FOR $j'_2 = MAX((j'_2 + 2)\%5, -3j'_1 - 5j_{0_1}, \lceil \frac{-j'_1 - 5j_{0_2}}{2} \rceil)$
                              TO $MIN(19, -3j'_1 - 5j_{0_1} + 195, \lfloor \frac{-j'_1 - 5j_{0_1} + 145}{2} \rfloor)$ $STEP = 5$ DO
            $\begin{pmatrix} j_1 \\ j_2 \end{pmatrix} = \begin{pmatrix} j_{0_1} \\ j_{0_2} \end{pmatrix} + \begin{bmatrix} \frac{3}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{2}{5} \end{bmatrix} \begin{pmatrix} j'_1 \\ j'_2 \end{pmatrix}$
        $A[j_1, j_2] = A[j_1 - 1, j_2 - 2] + A[j_1 - 3, j_2 - 1]$
  ENDFOR
ENDFOR
```

□

7. COMPARISON

We shall now compare our method for generating tiled code with the one presented by Ancourt and Irigoinin [23]. Let us primarily consider the problem of enumerating the Tile Space. The system in (1) contains $4n$ inequalities with $2n$ variables, while the one in (6) contains $2n$ inequalities with n variables. This means that in our case, the extremely complex Fourier-Motzkin elimination algorithm is supplied with a much smaller input. In order to sweep the internal points of every tile, Ancourt and Irigoin propose the application of Fourier-Motzkin elimination to the system derived from expression (7). In our case, we can avoid any further application of FM elimination by making the following observation: the first part of the systems of inequalities in (6) and (8) are expressed by matrices BP and BP' respectively. The second one can be derived from the first one by dividing each column k by the constant v_{kk}. Since all steps of FM elimination depend only on the form of these matrices, we can apply it to both systems (6) and (8) by executing the method only once. That is, if we apply FM elimination to the matrix $[BP|\vec{b'}|BP'|\vec{b}|B]$, taking care to adapt the matrix BP to the desired form, the matrix BP' can be simultaneously adapted. Consequently, the procedure of generating tiled code becomes much more efficient, as in the case of linear loop transformations.

Moreover, our method results in fewer inequalities for the bounds of the Tile Space and thus needs fewer bound calculations during run time. As a drawback, it may enumerate some redundant tiles as well. However, these are restricted only at the edges of the Tile Space. If the Tile Space is large enough, then we can safely assert that their number is negligible in respect to the total number of tiles that actually have to be traversed. In any case, the method that sweeps iterations within tiles, finds no iterations to be executed within these tiles.

Table 1: Example Iteration Spaces

	i_1		i_2		i_3		i_4	
	lower bound	**upper bound**	**lower bound**	**upper bound**	**lower bound**	**upper bound**	**lower bound**	**upper bound**
Space$_1$	0	3999	0	2999	-	-	-	-
Space$_2$	0	1599	0	3199	-	-	-	-
Space$_3$	0	239	$-\frac{i_1}{3}$	$\frac{159+2i_1}{5}$	-	-	-	-
Space$_4$	0	2099	0	$2099 - i_1$	-	-	-	-
Space$_5$	0	4999	0	5999	0	3999	-	-
Space$_6$	0	1999	0	$3999 - i_1$	$-i_1$	$2999 - i_1$	-	-
Space$_7$	0	1999	$-i_1$	$3999 - i_1$	$-i_1$	$2999 - i_1$	-	-
Space$_8$	0	49	$3i_1$	$59 - 2i_1$	$-2i_2$	$39 + 2i_1 - 3i_2$	-	-
Space$_9$	0	19	0	29	0	39	0	19
Space$_{10}$	0	19	0	$29 - i_1$	$-i_2$	$39 - i_1$	$-i_1$	$19 - i_2$
Space$_{11}$	0	19	0	$29 - 2i_1$	$-2i_1$	$39 - 3i_2$	0	$19 - i_1$
Space$_{12}$	0	19	$-2i_1$	$29 - 2i_1$	$-3i_2$	$39 - 3i_2$	$-2i_1$	$19 - 2i_1$

In order to evaluate the proposed method, we ran a number of examples with different matrices P, B, b and counted the number of row-operations needed by Fourier-Motzkin for the calculation of the bounds in the Tile Space. The implementation of the Fourier-Motkzin algorithm was based on the techniques presented in [24]. Table 1 shows the iteration spaces used. Note that the first four spaces are 2-dimensional, the second ones are 3-dimensional and the last ones are 4-dimensional. The shapes in all cases vary from rectangular to more complex ones. Each iteration space was tiled using four tiling transformations. More specifically, the 2-dimensional algorithms were tiled using $P_1 = \begin{bmatrix} 5 & 0 \\ 0 & 4 \end{bmatrix}$, $P_2 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$, $P_3 = \begin{bmatrix} 10 & -5 \\ -8 & 10 \end{bmatrix}$ and $P_4 = \begin{bmatrix} 6 & 3 \\ 2 & 9 \end{bmatrix}$, the 3-dimensional using $P_5 = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 5 \end{bmatrix}$, $P_6 = \begin{bmatrix} 5 & 6 & 0 \\ 0 & 6 & 2 \\ 3 & 0 & 2 \end{bmatrix}$, $P_7 = \begin{bmatrix} 5 & 6 & 0 \\ 0 & 6 & 2 \\ 3 & -1 & 2 \end{bmatrix}$ and $P_8 = \begin{bmatrix} 15 & 6 & -1 \\ 0 & 6 & 2 \\ 4 & 0 & 2 \end{bmatrix}$ and the 4-dimensional using $P_9 = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 15 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$, $P_{10} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$, $P_{11} = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 5 & -5 & 0 \\ -2 & 0 & 5 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$ and $P_{12} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ -5 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 4 & 0 & 0 & 4 \end{bmatrix}$. The row operations required in each case are shown in Tables 2–4. We denote the method presented by Ancourt and Irigoin in [23] as AI and our method as RI (Reduced Inequalities). It is clear that, as expected, in all cases RI method greatly outperforms AI method.

Table 2: Number of Row Operations for 2-dimensional Algorithms

	AI method	**RI method**
Space$_1/P_1$	25	5
Space$_1/P_2$	756	30
Space$_1/P_3$	727	30
Space$_1/P_4$	772	30
Space$_2/P_1$	25	5
Space$_2/P_2$	802	30
Space$_2/P_3$	769	30
Space$_2/P_4$	730	30
Space$_3/P_1$	95	5
Space$_3/P_2$	536	24
Space$_3/P_3$	691	25
Space$_3/P_4$	816	24
Space$_4/P_1$	210	5
Space$_4/P_2$	859	32
Space$_4/P_3$	836	32
Space$_4/P_4$	823	32

Table 3: Number of Row Operations for 3-dimensional Algorithms

	AI method	**RI method**
Space$_5/P_5$	59	11
Space$_5/P_6$	8671	106
Space$_5/P_7$	9924	106
Space$_5/P_8$	30264	257
Space$_6/P_5$	757	11
Space$_6/P_6$	13393	84
Space$_6/P_7$	18578	111
Space$_6/P_8$	60536	259
Space$_7/P_5$	1157	11
Space$_7/P_6$	12987	106
Space$_7/P_7$	26226	106
Space$_7/P_8$	91118	257
Space$_8/P_5$	4083	11
Space$_8/P_6$	128344	147
Space$_8/P_7$	33157	147
Space$_8/P_8$	347018	187

We ran the above experiments on a Sun HPC450 with 4 UltraSparcII 400MHz CPUs and 1GB RAM, using Solaris 8. Some indicative execution times are, for example: Using $iteration\ space_8$ with tiling matrix P_6, AI method completed in $6.9min$, while RI method completed in $4.5msec$. Using $iteration\ space_8$ with tiling matrix P_8, AI method completed in $25min$, while RI method completed in $5.4msec$. This illustrates the improve-

Table 4: Number of Row Operations for 4-dimensional Algorithms

	AI method	**RI method**
Space$_9/P_9$	108	20
Space$_9/P_{10}$	1143	115
Space$_9/P_{11}$	6228	193
Space$_9/P_{12}$	13649	164
Space$_{10}/P_9$	5085	21
Space$_{10}/P_{10}$	16492	313
Space$_{10}/P_{11}$	151682	227
Space$_{10}/P_{12}$	291120	441
Space$_{11}/P_9$	702	20
Space$_{11}/P_{10}$	2625	126
Space$_{11}/P_{11}$	15642	228
Space$_{11}/P_{12}$	14695	209
Space$_{12}/P_9$	4032	20
Space$_{12}/P_{10}$	10045	293
Space$_{12}/P_{11}$	30173	266
Space$_{12}/P_{12}$	219219407	375

ment achieved when applying our method for the reduction of the compilation time using tiling transformations.

8. CONCLUSIONS

In this paper, we surveyed all previous work concerning code generation for linear loop transformations and tiling transformations and proposed a novel approach for the problem of generating code for tiled nested loops. Our method is applied to general parallelepiped tiles and non-rectangular space boundaries as well. In order to generate code efficiently, we divided the problem in the subproblems of enumerating the tiles and sweeping the points inside each tile. In the first case, we extended previous work on non-unimodular transformations in order to precisely traverse all tile origins. In the second case, we proposed the use of a non-unimodular transformation in order to transform the tile iteration space into a hyper-rectangle. We traversed that rectangular space and adjusted the results in order to access the internal points of any single tile in the tile space. Fourier-Motzkin elimination is applied only once to a system of inequalities as large as the systems that arise from any linear transformation. Experimental results show that our method outperforms previous work since it constructs smaller systems of inequalities that can be simultaneously eliminated.

References

[1] J. Ramanujam, “Non-Unimodular Transformations of Nested Loops,” in *Proceedings of Supercomputing 92*, pp. 214–223, Nov. 1992.

[2] J. Ramanujam, “Beyond Unimodular Transformations,” *Journal of Supercomputing,* 9(4), pp. 365–389, Oct. 1995.

[3] J. Xue, “Automatic Non-Unimodular Loop Transformations for Massive Parallelism,” *Parallel Computing,* 20(5) pp. 711-728, 1994.

[4] A. Fernandez, J. Llaberia and M. Valero, “Loop Transformation Using Nonunimodular Matrices,” *IEEE Trans. on Parallel and Distributed Systems,* vol.6, no.8, pp. 832–840, Aug. 1995.

[5] W. Li and K. Pingali, “A Singular Loop Transformation Framework based on Nonsingular Matrices,” in *Proceedings of the Fifth Workshop on Languages and Compilers for Parallel Computing,* Aug. 1992.

[6] F. Irigoin and R. Triolet, “Supernode Partitioning,” in *Proceedings of the 15th Ann. ACM SIGACT-SIGPLAN Symp. Principles of Programming Languages,* pp. 319–329, San Diego, California, Jan. 1988.

[7] J. Ramanujam and P. Sadayappan, “Tiling Multidimensional Iteration Spaces for Multicomputers,” *Journal of Parallel and Distributed Computing,* vol. 16, pp.108–120, 1992.

[8] P. Boulet, A. Darte, T. Risset and Y. Robert, “(Pen)-ultimate tiling?,” *INTEGRATION, The VLSI Jounal,* volume 17, pp. 33–51, 1994.

[9] J. Xue, “Communication-Minimal Tiling of Uniform Dependence Loops,” *Journal of Parallel and Distributed Computing,* vol. 42, no.1, pp. 42–59, 1997.

[10] J. Xue, “On Tiling as a Loop Transformation,” *Parallel Processing Letters,* vol.7, no.4, pp. 409–424, 1997.

[11] W. Shang and J.A.B. Fortes, “Independent Partitioning of Algorithms with Uniform Dependencies,” *IEEE Trans. Comput.,* vol. 41, no. 2, pp. 190–206, Feb. 1992.

[12] E. H. Hollander, “Partitioning and Labeling Loops by Unimodular Transformations,” *IEEE Trans. on Parallel and Distributed Systems,* vol. 3, no. 4, pp. 465–476, Jul. 1992.

[13] M. Wolf and M. Lam, “A Loop Transformation Theory and an Algorithm to Maximize Parallelism,” *IEEE Trans. on Parallel and Distributed Systems,* vol.2 no.4, pp. 452–471, Oct. 1991.

[14] J.-P. Sheu and T.-S. Chen, “Partitioning and Mapping Nested Loops for Linear Array Multicomputers,” *Journal of Supercomputing,* vol. 9, pp. 183–202, 1995.

[15] Chung-Ta King, W-H Chou and L. Ni, “Pipelined Data-Parallel Algorithms: Part II Design,” *IEEE Trans. on Par. and Dist. Systems,* vol. 2, no. 4, pp. 430–439, Oct. 1991.

[16] P. Tsanakas, N. Koziris and G. Papakonstantinou, “Chain Grouping: A Method for Partitioning Loops onto Mesh-Connected Processor Arrays,” *IEEE Trans. on Parallel and Distributed Systems,* vol. 57, no. 2, pp.941–955, Sep. 2000.

[17] J.-P. Sheu and T.-H. Tai, “Partitioning and Mapping Nested Loops on Multiprocessor Systems,” *IEEE Trans. on Parallel and Distributed Systems,* vol. 2, no. 4, pp. 430–439, Oct. 1991.

[18] I. Drossitis, G. Goumas and N. Koziris, G. Papakonstantinou, P. Tsanakas, “Evaluation of Loop Grouping Methods based on Orthogonal Projection Spaces,” in *Proceedings of the 2000 Int’l Conference on Parallel Processing (ICPP-2000),* pp. 469–476, Toronto, Canada, Aug. 2000.

[19] E. Hodzic and W. Shang, “On Supernode Transformation with Minimized Total Running Time,” *IEEE Trans. on Parallel and Distributed Systems,* vol. 9, no. 5, pp. 417–428, May 1998.

[20] G. Goumas, A. Sotiropoulos and N. Koziris, “Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping,” *Int’l Parallel and Distributed Processing Symposium 2001 (IPDPS-2001),* San Francisco, California, Apr. 2001.

[21] E. Hodzic and W. Shang, “On Time Optimal Supernode Shape,” in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA),* Las Vegas, CA, Jun. 1999.

[22] P. Tang and J. Xue, “Generating efficient tiled code for distributed memory machines,” *Parallel Computing,* 26(11) pp. 1369–1410, 2000.

[23] C. Ancourt and F. Irigoin, “Scanning Polyhedra with DO Loops,” in *Proceedings of the Third ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPoPP),* pp. 39–50, Apr. 1991.

[24] W. Pugh, “The Omega Test: a fast and practical integer programming algorithm for dependence analysis,” *Comuunications of the ACM,* 35(8): pp. 102–114, Aug. 1992.

[25] A.J.C. Bik and H.A.G. Wijshoff, “Implementation of Fourier-Motzkin Elimination,” in *Proceedings of the first annual Conference of the ASCI,* Heijen, The Netherlands, pp 377–386, May 1995.

Georgios Goumas, received his Diploma in Electrical and Computer Engineering from the National Technical University of Athens in 1999. He is currently a PhD candidate at the Department of Electrical and Computer Engineering, National Technical University of Athens. His research interests include Parallel Processing (Parallelizing Compilers, Automatic Loop Partitioning), Parallel Architectures, High Speed Networking and Operating Systems.

Maria Athanasaki, received her Diploma in Electrical and Computer Engineering from the National Technical University of Athens in 2001. She is currently a PhD candidate at the Department of Electrical and Computer Engineering, National Technical University of Athens. Her research interests include Parallel and Distributed Systems, Parallelizing Compilers, Dependence Analysis and High Performance Numerical Applications.

Nectarios Koziris, received his Diploma in Electrical Engineering from the National Technical University of Athens and his Ph.D. in Computer Engineering from NTUA (1997). He is currently a faculty member at the Department of Electrical and Computer Engineering, National Technical University of Athens. His research interests include Computer Architecture, Parallel Processing, Parallel Architectures (Loop Compilation Techniques, Automatic Algorithm Mapping and Partitioning) and Communication Architectures for Clusters. He has published more than 40 research papers in international refereed journals, and in the proceedings of international conferences and workshops. He has also published two Greek textbooks “Mapping Algorithms into Parallel Processing Architectures”, and “Computer Architecture and Operating Systems”. Nectarios Koziris is a recipient of the IEEE-IPDPS01 best paper award for the paper “Minimising Completion Time for Loop Tiling with Computation and Communication Overlapping”. He is reviewer in International Journals and Conferences. He served as PC member for HiPC 2002 and Program Chair for the SAC03 ACM Symposium on Applied Computing-Special Track on Parallel, Distributed Systems and Networking. He conducted research in several EU and national Research Programmes. He is a member of IEEE Computer Society, member of IEEE-CS TCPP and TCCA (Technical Committees on Parallel Processing and Computer Architecture), ACM and organized the Greek IEEE Chapter Computer Society.