Automatic Code Generation for Executing Tiled Nested Loops onto Parallel Architectures

Georgios Goumas
National Technical University of Athens
Computing Systems Laboratory
Dept. of Electrical and Computer Engineering
goumas@cslab.ntua.gr

Maria Athanasaki
National Technical University of Athens
Computing Systems Laboratory
Dept. of Electrical and Computer Engineering
maria@cslab.ntua.gr

Nectarios Koziris
National Technical University of Athens
Computing Systems Laboratory
Dept. of Electrical and Computer Engineering
nkoziris@cslab.ntua.gr

ABSTRACT
This paper presents a novel approach for the problem of generating tiled code for nested for-loops using a tiling transformation. Tiling or supernode transformation has been widely used to improve locality in multi-level memory hierarchies as well as to efficiently execute loops onto non-uniform memory access architectures. However, automatic code generation for tiled loops can be a very complex compiler work due to non-rectangular tile shapes and iteration space bounds. Our method considerably enhances previous work on rewriting tiled loops by considering parallelepiped tiles and arbitrary iteration space shapes. The complexity of code generation for tiling transformation is now reduced to the complexity of code generation for any linear transformation. Experimental results which compare all so far presented approaches, show that the proposed approach for generating tiled code is significantly accelerated.

Keywords
Loop tiling, non-unimodular transformations, Fourier-Motzkin elimination, code generation, Hermite Normal Forms.

1. INTRODUCTION
Tiling or supernode partitioning is a 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]. In their paper Ramanujam and Sadayappan [9] 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. In order to find optimal tile shapes, Boulet et al. in [3] and Xue in [12] and [14] used a communication function that has to be minimized by linear programming approaches.

SAC 2002, Madrid, Spain

Although tiling as a loop transformation is extensively discussed, only rectangular tiling is used in real applications. However, communication criteria may suggest the use of non-rectangular tiles. On the other hand, non-rectangular loop nests very commonly appear in numerical applications. This means that, in the general case, we have to deal with non-rectangular shapes in tiles and iteration spaces. A method for manipulating non-rectangular tiles and general convex spaces has been introduced by Ancourt and Irigoin in [1], but the problem of calculating the exact transformed loop bounds is formulated as a large system of linear inequalities. The authors use the Fourier-Motzkin elimination method in order to transform these inequalities into a form appropriate for the calculation of the exact final loop bounds. However, due to the vast complexity of the Fourier-Motzkin elimination, this method cannot be practically applied for iteration spaces of more than 3 dimensions.

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 [10] and [11]. 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 face this fact, we first transform the parallelepiped (non-rectangular) tile (Tile Iteration Space-TIS) into a rectangular one ($TTIS$), sweep the derived Transformed Tile Iteration Space $TTIS$ 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 into consideration the original iteration space bounds as well. In both cases, the resulting systems of inequalities are eliminated using Fourier-Motzkin (FM) elimination, but as it will be shown, the second application can easily be avoided. The complexity of our method is equal to the methods proposed for code generation of linear loop transformations. Compared to the method presented by Irigoin and Ancourt in [1], 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 is introduced in Section 2. We present tiling or supernode transformation in Section 3. The problem of code generation for tiling transformation is defined in Section 4. Previous work on generating tiled code is presented in Section 5. Our method for generating tiled code is presented in detail in section 6. In section 7 we compare our method with the one presented in [1] and present some experimental results. Finally, in Section 8 we summarize our results and propose future work.

2. ALGORITHMIC MODEL-NOTATION

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
      $AS_1(j)$
      ...
      $AS_k(j)$
    ENDFOR
    ...
  ENDFOR
ENDFOR

where: (1) $j = (j_1, ..., j_n)$, (2) l_1 and u_1 are rational-valued constants, (3) l_k and u_k ($k = 2...n$) are of the form:
$l_k = max(\lceil f_{k1}(j1, \ldots, j_{k-1})\rceil, \ldots, \lceil f_{kr}(j1, \ldots, j_{k-1})\rceil)$,
$u_k = min(\lfloor g_{k1}(j1, \ldots, j_{k-1})\rfloor, \ldots, \lfloor g_{kr}(j1, \ldots, j_{k-1})\rfloor)$,
where f_{ki} and g_{ki} are affine functions and (4) $AS_1, ..., AS_k$ are assignment statements of the form $V_0 = E(V_1, ..., V_l)$, where V_0 is an output variable indexed by j and produced by expression E operating on input variables $V_1, ..., V_l$, also indexed by j. Thus we are not only dealing with rectangular iteration spaces, but also with more general 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.

Throughout this paper the following notation is used: N is the set of naturals, Z is the set of integers 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. In addition we define the symbols a^+ and a^- as follows: $a^+ = max(a, 0)$ and $a^- = max(-a, 0)$.

3. SUPERNODE TRANSFORMATION

In a supernode 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. Supernode transformation is defined by the n-dimensional square matrix H. Each row vector of H is perpendicular to one family of hyperplanes forming the tiles. Dually, supernode transformation can be defined by n linearly independent vectors, which are the sides of the supernodes. Similar to matrix H, matrix P contains the side-vectors of a supernode as column vectors. It holds $P = H^{-1}$. Formally supernode transformation is defined as follows:

$$r : Z^n \longrightarrow Z^{2n}, r(j) = \begin{bmatrix} \lfloor Hj \rfloor \\ j - H^{-1}\lfloor Hj \rfloor \end{bmatrix},$$

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, supernode transformation is not a linear transformation. The following spaces are derived from a supernode transformation H, when 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 supernode 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 $J^S \xrightarrow{P} TOS$. 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

Consider a supernode 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 1, 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 $TOS = \{(-2,2), (-1,4), (-1,1), \ldots, (5,4), (6,0), (6,3)\}$. The points of TOS are shown in bold dots. ⊣

The iteration space J^n of an algorithm, as defined in section 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 the affine functions f_{kr} and g_{kr} in section 2 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 smallest integer such as gH is an integer matrix. Thus we can rewrite the above system of

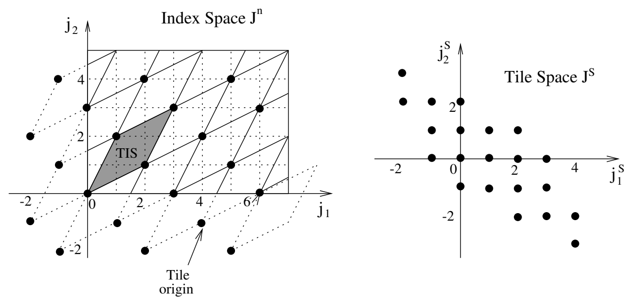

Figure 1: J^S, TIS and TOS from Example 1

inequalities as follows: $0 \leq gH(j - j_0) \leq (g-1)$. 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: $S(j - j_0) \leq \vec{s}$.

4. CODE GENERATION

In the next two sections we elaborate on methods for generating tiled code that will traverse an iteration space J^n transformed according to 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 supernode (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.

5. PREVIOUS WORK

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 $j'_1, j'_2, \ldots, j'_n$. Ancourt and Irigoin in [1] dealt with these subproblems constructing appropriate sets of inequalities for each case. In order to traverse the Tile Space J^S the first system is constructed, which consists of the inequalities representing the original index space and the inequalities 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 $j_0 = P j_0^S$. 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:

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

In order to traverse the internal points of every tile, the above set of inequalities is rewritten equivalently:

$$\begin{pmatrix} B \\ gH \\ -gH \end{pmatrix} j \leq \begin{pmatrix} \vec{b} \\ (g-1)\vec{1} + g j_0^S \\ \vec{0} - g j_0^S \end{pmatrix}, \qquad (2)$$

Ancourt and Irigoin propose the application of Fourier-Motzkin elimination to the above systems in order to obtain proper formulas for the lower and upper bounds of the $2n$-dimensional loop that will traverse the tiled space.

6. OUR METHOD

We shall now introduce an alternative method to generate tiled code. The concept of dividing the main problem into the subproblems of traversing the Tile Space J^S and sweeping the internal points of every tile, is preserved here as well. However, the point of view is different, since in both subproblems certain transformations are applied before constructing the final sets of inequalities and applying Fourier-Motzkin elimination to them. In this way, we shall be able to reduce the inequalities involved in the derived systems and thus significantly decrease the steps of Fourier-Motzkin elimination.

6.1 Enumerating the Tiles

The subproblem of traversing the Tile Space J^S has been faced by many authors as an example of applying the non-unimodular tiling transformation to the original iteration space. More specifically, Ramanujam in [10] [11] applied the non-unimodular tiling transformation to the set of inequalities $Bj \leq \vec{b}$ representing the iteration space as follows: $Bj \leq \vec{b} \Rightarrow BH^{-1}Hj \leq \vec{b} \Rightarrow$

$$BPj_0^S \leq \vec{b} \qquad (3)$$

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 $u_1^S, \ldots, u_n^S$.

Unfortunately the previous approach fails to enumerate tile origins exactly. Note that the system of inequalities in (3) is satisfied by points in the Tile Space J^S whose inverse belong to J^n. However, as stated in section 3 and indicated in Figure 1 there exist some points in TOS that do not belong to J^n. Although these points do not satisfy the preceding systems of inequalities, they must be also traversed. Consequently, a modification is needed in order for Fourier-Motzkin elimination to scan all tiles correctly. As we can see in Figure 2, what is needed is a proper reduction of the lower bounds and/or a proper increase of the upper bounds of our space in order to include all tile origins. 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:

$$Bj_0 \leq \vec{b}', \qquad (4)$$

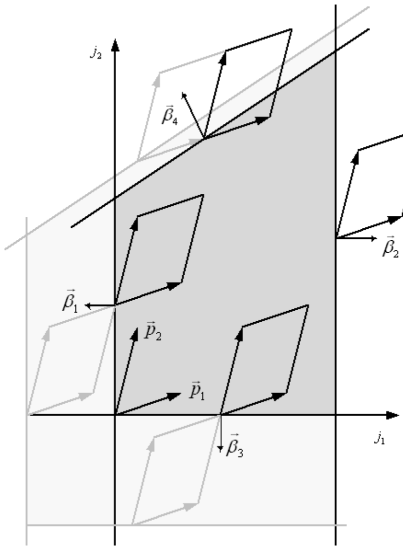

Figure 2: Expanding bounds to include all tile origins

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

$$b'_i = b_i + \frac{g-1}{g}\sum_{r=1}^{n}(\sum_{j=1}^{n}\beta_{ij}p_{jr})^- \tag{5}$$

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 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: $0 \le gH(j - j_0) \le (g-1)$. The i-th row of this inequality can be rewritten: $0 \le \vec{h_i}(j - j_0) \le \frac{g-1}{g}$, where $\vec{h_i}$ is the i-th row-vector of matrix $H = P^{-1}$. Therefore: $0 \le \vec{h_i}\sum_{j=1}^{n}\lambda_j\vec{p_j} \le \frac{g-1}{g}$. As $P = H^{-1}$ it holds that $\vec{h_i}\vec{p_i} = 1$ and $\vec{h_i}\vec{p_j} = 0$ if $i \ne j$. Consequently the last form can be rewritten: $0 \le \lambda_i \le \frac{g-1}{g}$ for all $i = 1, \ldots, n$.

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

$$\Rightarrow \sum_{j=1}^{n}\beta_{kj}j_{0j} \le b_k - \sum_{i=1}^{n}\lambda_i\sum_{j=1}^{n}\beta_{kj}p_{ji} \tag{6}$$

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

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

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

Using the symbols a^+ and a^- given in the Notation section of this paper, the previous inequalities can in every case be rewritten as follows: $-\frac{g-1}{g}(\sum_{j=1}^{n}\beta_{kj}p_{ji})^- \le \lambda_i\sum_{j=1}^{n}\beta_{kj}p_{ji} \le \frac{g-1}{g}(\sum_{j=1}^{n}\beta_{kj}p_{ji})^+$
If added for $i = 1, \ldots, n$ this inequality gives: $-\sum_{i=1}^{n}\lambda_i\sum_{j=1}^{n}\beta_{kj}p_{ji} \le \frac{g-1}{g}\sum_{i=1}^{n}(\sum_{j=1}^{n}\beta_{kj}p_{ji})^-$.

Therefore, from the last formula and the inequality (6), we conclude that $\sum_{j=1}^{n}\beta_{kj}j_{0j} \le b_k + \frac{g-1}{g}\sum_{i=1}^{n}(\sum_{j=1}^{n}\beta_{kj}p_{ji})^-$. Thus, for each tile with origin j_0 which has at least one point in the initial iteration space, it holds that $Bj_0 \le \vec{b'}$, 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

$$BPj_0^S \le \vec{b'} \tag{7}$$

Thus we can adjust the system of inequalities in (3) making use of Lemma 1 and replacing the vector $\vec{b}$ with $\vec{b'}$. 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.

6.2 Sweeping the points within a tile

As far as the scanning of the internal points of a tile is concerned, we present a new approach based on the use of a non-unimodular tranformation. Unlike the method presented in section 5, which traverses the tile moving along the directions of the orthocanonical space, this technique scans the points moving along the sides of the tiles. Our goal is to 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 a n-dimensional nested loop and then transform the indices of the loop so as to return to the proper points of the TIS. Our method consists of three steps: determination of the transformation matrix, code generation for traversing the $TTIS$ and adjustment in order to access all points of J^n.

6.2.1 Determining the Transformation Matrix

We are searching for a transformation pair (P', H'): $TTIS \xrightarrow{P'} TIS$ and $TIS \xrightarrow{H'} TTIS$ (Fig. 3). Intuitively, we demand P' to be parallel to 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 to this, we demand the lattice of H' (denoted as $\mathcal{L}(H')$) to be an integer space in order for loop indices to be able to traverse it. Formally, we are searching for a 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.

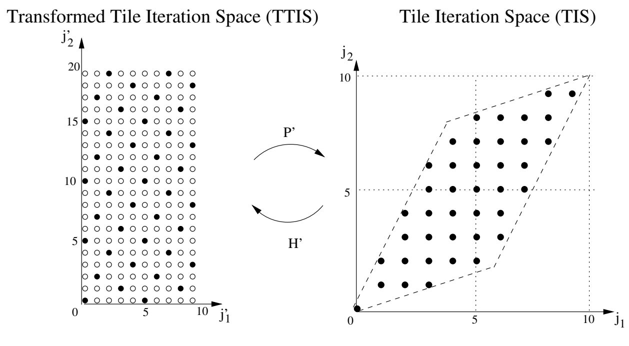

Figure 3: Traverse the TIS with a non-unimodular transformation

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, $\hat{u}_k = (u_{k1}, \dots, u_{kn})$, $u_{kk} = 1, u_{kj} = 0, j \neq k$. Then according to the above $A\hat{u}_k = [\sum_{i=1}^{n} a_{1i}u_{ki}, \sum_{i=1}^{n} a_{2i}u_{ki}, \dots, \sum_{i=1}^{n} a_{ni}u_{ki}]^T = [a_{1k}, a_{2k}, \dots, a_{nk}]^T \in Z^n$. This holds for all $\hat{u}_k, k = 1 \dots n$, 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 3, 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'.

6.2.2 *Traversing the TTIS*

For the code generation we shall use the same notion as in [10], [13], [8] 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. We use a n-dimensional nested loop with iterations indexed by $j'(j'_1, j'_2, \dots, j'_n)$ 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 $j'_{k+1}, \dots, j'_n$ should also be augmented by certain offset values $a_{(k+1)k}, \dots, a_{nk}$. 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 $P' \begin{bmatrix} j'_1 & j'_2 & \dots & j'_n + c_n \end{bmatrix}^T \in Z^n$. 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 incremental 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 $c_k, a_{(k+1)k}, \dots, a_{nk}$ such that:
$P' \begin{bmatrix} j'_1 & \dots & j'_k + c_k & j'_{k+1} + a_{(k+1)k} & \dots & j'_n + a_{nk} \end{bmatrix}^T \in Z^n$. The following Lemma 3 proves that incremental steps c_k and incremental offsets a_{kl}, ($k = 1 \dots n$ and $l = 1 \dots k-1$), are directly obtained from the Column Hermite Normal Form of matrix H', denoted $\widetilde{H'}$.

Lemma 3: If $\widetilde{H'}$ is the column HNF of H' and $j'(j'_1, j'_2, \dots, j'_n)$ 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 additional offsets are $a_{kl} = \widetilde{h'}_{kl}$, ($k = 1 \dots n$ and $l = 1 \dots k-1$).

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 incremental offsets are added to the current indices.

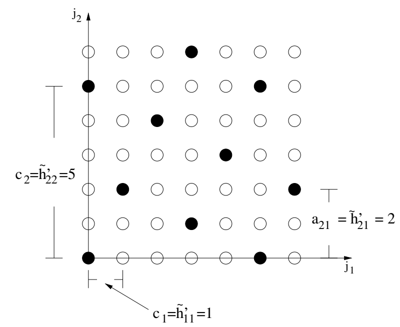

Figure 4: Steps and initial offsets in $TTIS$ derived from matrix $\widetilde{H'}$

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

$j'_2 = -\widetilde{H'}[2][1]; \dots, \widetilde{H'}[n][1]; PHASE = 1;$
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. $\dashv$

6.2.3 *Accessing the points of J^n*

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 that is derived from the indices of the former loop ($j' = (j'_1, \dots, j'_n)$) 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 before, a point $j \in J^n$ satisfies the following set of inequalities: $Bj \leq \vec{b}$. Replacing j by the above equation we have: $B(j_0 + P'j') \leq \vec{b} \Rightarrow$

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

By 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 arised in the previous section is also faced, since no computation is performed in these tiles.

7. COMPARISON

We shall now compare our method for generating tiled code with the one presented by Irigoin and Ancourt in [1]. Let us primarily consider the problem of enumerating the tile space. Comparing the systems (1) and (7) we conclude that in our case, the Fourier-Motzkin elimination algorithm [2], whose complexity is doubly exponential [7], 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 (2). 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 (7) 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}. Thus we can apply FM elimination to both systems 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.

The cost of this acceleration is that system (7) is not as precise as (1), it may result to some redundant tiles. However these are restricted only at the edges of the initial 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. In addition our method results to fewer inequalities for the bounds of the Tile Space and thus the computational cost avoided during run time may compensate the overhead of enumerating redundant tiles.

In order to evaluate the proposed method, we ran several representative examples for 2-dimentional and 3-dimentional iteration spaces and counted the number of row-operations needed for Fourier - Motzkin algorithm to be completed in both cases for the calculation of the bounds of the Tile Space. More specifically, we examined 15 2-dimensional examples with both rectangular and non-rectangular iteration spaces, rectangular and non-rectangular tiling matrices and 10 3-dimensional examples. The table in Figure 5 indicates the average values of the experimental results.

	Ancourt - Irigoin	Our Method
2-D	136,737	58
3-D	1,104,262,263	303

Figure 5: Average Row-operations performed by Experiments

8. CONCLUSIONS - FUTURE WORK

In this paper, we 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 every tile. In the first case, we extended previous work on non-unimodular tranformations 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. 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. Future work, involves the development of a framework combining this work with the one presented in [5] for pipelined scheduling of tiles to clusters.

9. REFERENCES

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

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

[3] P. Boulet, A. Darte, T. Risset and Y. Robert, "(Pen)-ultimate tiling?," *INTEGRATION, The VLSI Jounal,* volume 17, pp. 33–51, 1994. 2000.

[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] 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, April 2001.

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

[7] M. Jimenez, "Multilevel Tiling for Non-Rectangular Iteration Spaces," *PhD thesis, Universitat Politecnica de Catalunia,* Spain, 1999.

[8] W. Li and K. Pingali, "A Singular Loop Transformation Framework based on Non-singular Matrices," *Proceedings of the Fifth Workshop on Languages and Compilers for Parallel Computing,* August 1992.

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

[10] J. Ramanujam, "Non-Unimodular Transformations of Nested Loops," *Proceedings of Supercomputing 92,* (November 92), pp. 214–223, 1992.

[11] J. Ramanujam, "Beyond Unimodular Transformations," *Journal of Supercomputing,* 9(4), pages 365–389, October 1995.

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

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

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