A Modified Vertex Method for Parallelization of Arbitrary Nested Loops

W. Bielecki, R. Kocisz
Faculty of Computer Science, Technical University of Szczecin, Zolnierska 49 st., 71-210 Szczecin, Poland, fax. (+4891) 487-64-39
wbielecki@wi.ps.pl, rkocisz@wi.ps.pl

Abstract

A technique, permitting us to linearize constraints formed to find affine schedules for arbitrary nested loops, is presented. The main advantage of this technique is that it does not require finding the polytope vertices and results in the fewer number of inequalities and equalities than that yielded with the vertex technique. Affine schedules found are valid for the arbitrary positive lower and upper loop bounds. Experiments with the Livermore loops are discussed. The restriction of the technique and tasks for future research are discussed.

1. Introduction

Affine transformations of loops are currently the most powerful approaches to extract parallelism in loops, minimize synchronization, and improve memory locality [2,4,5,6,8,9,10,13,14,18]. An affine partitioning scheme consists of affine mappings, one for each statement in the loop, from the original index variables to the values of index variables in the transformed loop.

An m-dimensional affine partitioning for statement s in a loop is an m-dimensional affine expression \( \Phi_s(i) = C_s i + c_s \), which maps an instance of statement s, indexed by its iteration vector i, to an m-dimensional vector.

There are different purposes of the use of affine partitionings: space partitioning (statements belonging to the same space partition are mapped to the same processor), time partitioning (statements belonging to time partition i are executed before those in partition i+1), partitioning to enhance locality and reduce communications. Given affine mappings, well-known code generation techniques can be applied to generate transformed loops[1,2,3,19].

In this paper, we consider only affine time partitioning mappings called schedules[6].

The well-known techniques[6,8,9,13,14], permitting us to find affine schedules, result in building non-linear constraints, which should next be linearized by means of Farkas’ Lemma[17] or the polyhedron vertex method[18,20].

This paper presents a technique permitting us to linearize dependence constraints for finding affine schedules. It does not require finding the polytope vertices and results in the fewer number of inequalities and equalities than that yielded with the vertex technique.

2. Background

In this paper, we deal with affine loop nests where lower and upper bounds as well as array subscripts and conditionals are affine functions of surrounding loop indices and possibly of structure parameters, and the loop steps are known constants.

Our technique requires an exact dependence analysis[7,16]. In general, any known technique, extracting exact dependences, can be applied. However, describing the technique depends on the format of the presentation of exact dependences and carrying out experiences depends on the availability of tools permitting us to extract exact dependences.

In this paper, describing our technique and fulfilling experiments are based on the dependence analysis proposed by Pugh and Wonnacott[16] and we assume that the reader is familiar with this analysis.

The dependence analysis by Pugh and Wonnacott is implemented in Petit, a research tool for doing dependence analysis and program transformations.

Definition 2.1 A set of vectors in \( \mathcal{Q}^n \) is called a polyhedron if there exists an integer matrix A and an integer vector b such that \( P = \{ x | Ax \leq b \} \). A polytope is a bounded polyhedron.

Definition 2.2 A vertex of a set \( K \) is any point in \( K \) which cannot be expressed as a convex combination of any other distinct points in \( K \).

Definition 2.3 A ray of set \( K \) is a vector \( r \), such that \( x \in K \) implies \( (x + \mu r) \in K \) for all \( \mu \geq 0 \).

Definition 2.4 A line (or bidirectional ray) of set \( K \) is a vector \( l \), such that \( x \in K \) implies \( (x + \mu l) \in K \) for all \( \mu \).

The vertex method is based on the following theorem[18].

Theorem 2.1 Let \( D \) be a nonempty polyhedron. \( D \) can be written \( D = P + C \), where \( P \) is a polytope and \( C \) is a cone. Then any affine function \( h \) defined over \( D \) is nonnegative.
on $D$ iff 1) $h$ is nonnegative on each of the vertices of $P$ and 2) the linear part of $h$ is nonnegative (resp. null) on the rays (resp. lines) of $C$.

All the polyhedra produced by the dependence analysis of programs are in fact polytopes. Hence, an affine function is nonnegative on a polyhedron iff it is nonnegative on the vertices of this polyhedron.

To apply the vertex method for finding schedules, we firstly need to find the vertices of all polytopes defined by correspondent dependence relations. This problem is known to be non-polynomial[15]. Next, for each dependence relation, we should construct a linearized constraint not including the iteration vector.

Our technique requires that firstly, when necessary, dependence relations should be transformed to equivalent ones which satisfy the following conditions: 1) the dependence relation constraints do not include equalities; 2) all inequalities of the dependence relation constraints are resolved for the relation variables and solutions are presented in such a form that the operations $<, \leq, >, \geq$ between each pair of the relation variables can be determined.

Satisfying these conditions permits us to form proper constraints. We refer to the format of dependence relations, meeting the conditions above, as the normalized format.

To normalize relation dependences, we should carry out the following steps:
1) If the dependence relation constraint includes equalities, resolve them; let a solution to these equalities be of the form $x = a$; substitute all occurrences of $x$ in the dependence relations for $a$.
2) If the dependence relation constraint includes unresolved inequalities, resolve them for each relation variable.
3) Discover which operations from $<, \leq, >, \geq$ are hold.
4) If a dependence relation is presented as the union of relations, split it into a set of relations not including the union operations.

Let us consider the following relation: $R := \{[i,j] \rightarrow [i',j'] : 2i' = 1 \& \& j' = 1 \& \& 2i + 1 = 0 \& \& 2j + 1 = 0 \& \& 2 \leq i \leq 25\} \cup \{[i,j] \rightarrow [2i-1,2j-1] : 2i' = 1 \& \& 2j' = 1 \& \& 2 \leq i \leq 25 \& \& 2 \leq j \leq 25\}$.

Firstly the equalities in this relation are resolved: $i' = 2i - 1; j' = 2j - 1; i = 1; j = 1$.

The second step of the normalization procedure is skipped since all the inequalities in the relation are already resolved. Now we can present the relation in the following form: $R := \{[i,1] \rightarrow [2i-1,1] : 2 \leq i \leq 25\} \cup \{[i,j] \rightarrow [2i-1,2j-1] : 2 \leq i \leq 25 \& \& 2 \leq j \leq 25\}$.

We discover that between the pair of relation variables $i,j$ the operation “$<$” ($i < j$) is hold.

Finally, the union of the relations is split into the two separated relations as follows: $R_{1} := \{[i,1] \rightarrow [2i-1,1] : 2 \leq i \leq 25\}; R_{2} := \{[i,j] \rightarrow [2i-1,2j-1] : 2 \leq i \leq 25 \& \& 2 \leq j \leq 25\}$.

3. Linearizing schedule constraints

In this section, we consider the following task. Given a loop with parameterized bounds and a set of normalized dependence relations, $D$, originated with this loop and presented as follows

$D = \{\bar{I}_{p} \rightarrow \bar{J}_{p}, p = 1,2,\ldots, q\}$, (1)

find an $m$-dimensional affine schedule $\bar{F}_{sk}(i) = C_{sk}i + \bar{c}_{sk}$ for each statement $sk$ of the loop. For $k = 1, 2, \ldots, r$, such that $\bar{F}_{sj}(\bar{I}_{p}) > \bar{F}_{si}(\bar{I}_{p})$, where $r$ is the number of assignment statements in the loop body, $si, sj$ are the statements which instances originate the source and destination of the dependence $\bar{I}_{p} \rightarrow \bar{J}_{p}$. $C_{sk}$ is a matrix of dimensions $m \times n_{k}$. $n_{k}$ is the number of the loop nests surrounding statement $sk$, $\bar{c}_{sk}$ is an $m$-dimensional vector representing a constant term, "$>$" denotes the relation "lexicographically more than", such that $\bar{F}_{sj}(\bar{I}_{p}) > \bar{F}_{si}(\bar{I}_{p})$ iff operation $\bar{F}_{sj}(\bar{I}_{p})$ of statement $sj$ is executed before operation $\bar{F}_{si}(\bar{I}_{p})$ of statement $si$.

We require satisfying the following additional condition: $m$ should be as few as possible.

Since all dependence relations are presented in the normalized format, we can write $\bar{I}$ and $\bar{J}$, for some $j \in [1,q]$, in the following form

$\bar{I}_{j} = (A_{j} + D_{1j})\mathbf{i}_{i} + \bar{B}_{1j} + \bar{P}_{1j}$

$\bar{J}_{j} = (A_{2j} + D_{2j})\mathbf{i}_{j} + \bar{B}_{2j} + \bar{P}_{2j}$,

where $\bar{I}_{j}, \bar{J}_{j}$ are $m_{1j}$ and $m_{2j}$-dimensional vectors, respectively, $m_{1j} \leq n, m_{2j} \leq n$ are the numbers of the loops, surrounding two statements originating the dependence which source and destination are $\bar{I}_{j}, \bar{J}_{j}$, respectively; $n$ is the number of the loop nests; $\bar{B}_{1j}, \bar{B}_{2j}$ are $m_{1j}$ and $m_{2j}$-dimensional vectors, respectively, which represent constant terms; $\mathbf{i}_{i}$ and $\mathbf{i}_{j}$ are $n$-dimensional vectors defining vectors $\bar{I}_{j}$ and $\bar{J}_{j}$, respectively. $A_{1j}$ and $A_{2j}$ are matrices of dimension $m_{1j} \times n$ and $m_{2j} \times n$, respectively, which elements are constants; $\bar{P}_{1j}, \bar{P}_{2j}$ are $m_{1j}$ and $m_{2j}$-dimensional vectors, respectively, which represent symbolic parameters; $D_{1j}$ and $D_{2j}$ are matrices of dimensions
A one-dimensional affine schedule $\Phi$ for the set of dependence relations defined by (1) is legal if and only if for each dependence relation $\mathbf{1}_p \rightarrow \mathbf{1}_p$, $p = 1, 2, \ldots, q$, the following condition is satisfied

$$\Phi_s(\mathbf{1}_p) \geq \Phi_s(\mathbf{1}_p), p = 1, 2, \ldots, q.$$ 

A legal affine schedule permits executing the time partitions sequentially without the violation of any data dependence, that is, if an instance $u$ of statement $s1$ depends on an instance $v$ of statement $s2$, $u$ must either be executed in the same iteration as $v$ after $v$, or $u$ is executed in an iteration that comes after the iteration that executes $v$.

**Definition 3.2** A legal one dimensional affine schedule $\Phi$ satisfies strictly the dependence relation $\mathbf{1}_p \rightarrow \mathbf{1}_p$, $p = 1, 2, \ldots, q$, for a particular $p \in [1, q]$, originated with statements $s1$ and $s2$, if the following condition holds

$$\Phi_s(\mathbf{1}_p) > \Phi_s(\mathbf{1}_p), p = 1, 2, \ldots, q.$$ 

The condition above guarantees that two dependent instances $v$ and $u$ belong to different time partitions such that instance $u$ belongs to the partition that comes after the partition containing $v$.

The well-known technique [9] tries firstly to find the first dimension of mapping $\Phi_s$ which satisfies strictly as many as possible dependence relations comprised in set (1). If not all dependence relations from set (1) are satisfied strictly, it tries next to find the second dimension of $\Phi_s$ which satisfies strictly as many as possible non-satisfied strictly dependence relations contained in set (1) with the first dimension of $\Phi_s$. This procedure is continued until all dependences are satisfied strictly.

Consider how we can form schedule constraints using dependence relations. If a dependence relation $\mathbf{1}_j \rightarrow \mathbf{1}_j$, $j \in [1, q]$ is the self dependence, that is, it is originated with the same statement $s$, we seek a legal one-dimensional affine schedule for statement $s$, $\Phi_s(\mathbf{i}) = C_s \mathbf{i} + C_s$, such that the following condition is satisfied

$$C_s \mathbf{j} + C_s \geq C_s \mathbf{j} + C_s$$

or

$$C_s \mathbf{j} \geq C_s \mathbf{j}.$$ (2)

If a dependence relation $\mathbf{1}_j \rightarrow \mathbf{1}_j$, $j \in [1, q]$ is originated with two different statements $s1$ and $s2$, we seek two legal affine schedules, $\Phi_{s1}(\mathbf{i}) = C_{s1} \mathbf{i} + C_{s1}$ and $\Phi_{s2}(\mathbf{i}) = C_{s2} \mathbf{i} + C_{s2}$, such that the following condition is satisfied

$$C_{s1} \mathbf{j} + C_{s2} \geq C_{s1} \mathbf{j} + C_{s1}.$$ (3)

Let us introduce a $q_j$-dimensional vector $\mathbf{i}_j$ composed of all unique coordinates (having different names), $\mathbf{i}_j^1, \mathbf{i}_j^2, \ldots, \mathbf{i}_j^q_j$, of vectors $\mathbf{1}_j$ and $\mathbf{1}_j$, $q_j \leq m1_j + m2_j$ and rewrite equalities (2) and (3) in the form

$$\sum_{k=1}^{q_j} F_k^j \mathbf{i}_j^k + f_j \geq 0, j = 1, 2, \ldots, q$$

where $F_k^j$ and $f_j$ are expressions composed of integers and symbolic parameters.

We suppose that the lower bounds of loops to be parallelized are equal to or greater than 1.

The idea of a technique proposed is the following. We approximate each polyhedron defined by a dependence relation with an unbounded polyhedron with a vertex defined by the lower loop bounds equals to 1 and $n$ rays, each of which goes in the same direction as a correspondent coordinate of the iteration vector, where $n$ is the number coordinates of the iteration vectors (such a polyhedron is a kind of a rectangle). Then we linearize each constraint presented by (4) so that each one is legal for each point within the unbounded polyhedron formed as above stated. To guarantee this, we add to each linearized dependence constraint one or more additional inequalities or equalities which form additional constraints. Then we have to combined all constraints in one system and resolve it for unknowns representing schedules.

The modification of the vertex method is the following. Instead of linearizing constraints for each vertex of a polyhedron, we linearize constraints for one “artificial” vertex and add additional constraints on schedules to guarantee their legality.

### 3.1. First dimension

In this subsection, we present an algorithm for linearizing schedule constraints for the first dimension of schedules. It takes into account the operations $<, >, \leq, \geq$, $=$ between the relation variables in dependence relations.

**Algorithm 1** Linearize affine schedule constraints.

1. When necessary, normalize the dependence relations contained in set (1).
2. From each dependence relation $\mathbf{1}_j \rightarrow \mathbf{1}_j$, $j = 1, 2, \ldots, q$, build a constraint in the form of (4).
3. Construct a system of linear inequalities and equalities as follows. For each constraint (4) do:
3.1) Verify whether there exist such members $F'_j$ and $F''_j$ that $F'_j = -F''_j$, if so, then for each such a pair, construct the inequality
\[
F'_j \geq 0 \text{ if } i'_j \geq i''_j, \\
-F''_j \geq 0 \text{ if } i'_j < i''_j, \\
F'_j = 0 \text{ if } a \leq i'_j - i''_j b, a < 0, b > 0, \\
F''_j = 0 \text{ if } c \leq i'_j - i''_j \leq d,
\]
where $c, d$ are symbolic parameters or affine functions of those.

3.2) For the remaining members, form the inequalities as follows:
\[
F'_j \geq 0, k \neq s, r \text{ if } i'_j \geq 0, \\
-F''_j \geq 0, k \neq s, r \text{ if } i''_j < 0, \\
F'_j = 0, k \neq s, r \text{ if } a \leq i'_j \leq b, a < 0, b > 0, \\
F''_j = 0 \text{ if } c \leq i''_j \leq d,
\]
where $c, d$ are symbolic parameters or affine functions of those.

3.3) Form the following inequality
\[
\sum_{k=1}^{q} F^k_j + f_j \geq 0, \quad (5)
\]
where $q_j$ is the same as in constraint (4).

4) If there exist two or more the same equalities or inequalities in the formed constraint, eliminate such ones except for one for each of them.

**Definition 3.3** An inequality, defined by (5), is called as follows
\[
\Phi_{j1} = [C_{11}^j + C_{12}^j]_t, 
\]
we firstly build the constraint
\[
2C_{11}^t s + C_{12}^n s - C_{11}^n s - C_{12}^n s (2j'' - i) \geq 0, 
\]
which transform to the form
\[
(C_{11}^t + C_{12}^n s) i - C_{12}^n s j'' \geq 0. 
\]
Taking into account that $j'' \geq 1$, the resulting constraint is as follows
\[
C_{11}^t + C_{12}^n s \geq 0, \\
C_{12}^n s \geq 0, \\
C_{11}^t \geq 0. 
\]
where "\(\sqrt{\cdot}\)" marks the basic inequality.

### 3.2. Remaining dimensions

The remaining dimensions of schedules are formed successively in a manner similar to that for the first dimension. The only difference is that those dependences, that are already satisfied by preceding dimensions, should not be considered.

Let us assume that the first $j$-th dimensions of schedules have been determined, which we denote as $C_{s}^{j-1}$, and that we are currently processing the $j$-th dimension of the schedules.

We say that a dependence relation $I_p \rightarrow J_p$, originated with two different statements $s_1$ and $s_2$, is satisfied strictly by the first $j$-th dimensions of the schedules if the following condition is satisfied
\[
C_{s_1}^{j-1} I_p + C_{s_2}^{j-1} I_p < C_{s_1}^{j-1} I_p + C_{s_1}^{j-1} C_{s_2}^{j-1} I_p. 
\]
This means that this dependence relation will be respected regardless of how the remaining dimensions of the schedules are determined. Therefore, in order to determine the $i$-th dimension of the schedules, we should build constraints taking into account only unsatisfied dependences. The constraint is of the form
\[
C_{s}^{j-1} J_p + C_{s}^{j-1} I_p + C_{s}^{j-1} I_p. 
\]
We termine determining the dimensions of the schedules when all the dependences from set $D$ are satisfied strictly.

### 3.3. Solving linearized constraints

...
Let the basic inequalities of schedule constraints, constructed by our algorithm, be presented in the form $A_i x \geq 0$. For solving these constraints, we use the following approach. $k$ new variables, $n_i$, $i = 1..k$, are introduced, where $k$ is the number of basic inequalities in the constraint system. $k$ new constraints of the form $0 \leq n_i \leq 1$ are added to the system and all the basic inequalities in this system are transformed to the form $A_i x \geq n_i$. Then a goal function of the form $F(n_1, n_2, ..., n_k) = n_1 + n_2 + ... + n_k$ is constructed. Linear programming can be applied for finding unknowns of so modified constraints to maximize the value of function $F$. This permits us to maximize the number of basic inequalities satisfied strictly for each dimension of schedules. When the unknowns of constraints are found, we can check which basic inequalities are not satisfied strictly ($n_i = 0$), and then apply the same algorithm to find the next dimension of schedules constructing a new system which includes only non-satisfied strictly basic inequalities and correspondent additional constraints formed on the basis of Algorithm 1.

All the mathematical operations that are necessary for applying the approach described above are implemented in the PIP Library[22] which we have used for carrying out our experiments.

4. Advantages and restrictions of the technique

The advantages of the presented technique are i) it does not require finding vertices (non-polynomial problem), (ii) in general, it results in the much fewer number of inequalities and equalities in linearized schedule constraints in comparison with that yielded with the vertex technique, iii) linearized constraints do not comprise symbolic parameters, this reduces considerably the problem of solving schedule constraints, iv) schedules received are valid for the arbitrary positive bounds of loops to be parallelized.

The drawback of the technique is that it does not extract loop parallelism in that case when a linearized constraint includes two inequalities of the following forms:

$$\sum_{k=1,k \neq s}^{q} F^k_j + f_j \geq 0$$

and

$$-(\sum_{k=1,k \neq s}^{q} F^k_j + f_j) \geq 0.$$

In such a case, these inequalities result in one equality of the form

$$\sum_{k=1,k \neq s}^{q} F^k_j + f_j = 0$$

that is, we cannot satisfy strictly such inequalities, hence we cannot satisfy strictly corresponding dependence relations. In such cases, Farkas’ Lemma or the vertex method has to be applied to extract loop parallelism.

5. Related work and experiments

To our knowledge, there are the two well-known most powerful techniques permitting us to linearize schedule constraints built by means of known techniques [6,8,9,10,13,14]: Farkas’ Lemma [8,9,13,14] and the vertex method [18,20].

Applying Farkas’ Lemma results in a set of $(n+q+1)$ equalities for each dependence constraint [21], where $n$ is the size of the iteration vector, and $q$ is the size of the structural parameter vector[8].

Using the vertex method leads in general(eliminating the iteration indices and structural parameters) to a set of $v(l+q)$ inequalities for each dependence relation[18,20], where $v$ is the number of vertices of the polyhedral domain of the iteration vector, and $q$ is the size of the structural parameter vector. To apply the vertex method, we firstly need to find the vertices of a polytope representing dependences. This problem is known to be non-polynomial[15].

<table>
<thead>
<tr>
<th>Loop index</th>
<th>Modified Vertex Method</th>
<th>Vertex Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>The number of constraint s</td>
<td>Schedule</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>3</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>5</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>6</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>7</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>21</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>22</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>23</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Our technique requires building \((I+f)\) inequalities and equalities for each dependence relation, where \(f\) is the total number of distinct members \(F^j_i\) in a correspondent constraint (5).

To examine the power of the presented technique and to compare it with the vertex method, we have implemented both these techniques in a tool using Petit and the Omega calculator\[11\] for finding dependence relations and their normalization, respectively, as well as PIP\[22\] for solving linearized constraints. For carrying out experiments, we have chosen the Livermore loops \[23\], a widely used tool to measure CPU performance. The Livermore loops are composed of a number of tests which include a wide range of computational structures many of which can be found in applications of electrical engineering, for example, inner product, first difference, tridiagonal elimination, etc.

Table 1 shows the results of experiments carried out on the non-parametrized Livermore loops. The first column includes the indices of those Livermore loops for which Petit is able to discover dependence relations without any transformations of the original Livermore loop bodies. Such constructions as \(goto\) or \(break\) statements as well as indexing tables with elements of other tables prevent discovering dependences in loops with Petit. In the second and fourth columns, “+” (“-”) means that schedules are found (do not found) with the presented method and vertex method, respectively. The third and fifth columns represent the number of constraints yielded with our method and the vertex method, respectively.

As we can see from Table 1, our technique yields the fewer number of constraints than that yielded with the vertex method. However, because of the reason explained in Section 4, the presented technique does not permit us to find schedules for several Livermore loops (8,10,11, 22,24). Both methods generate the same schedules except for loop 6, for which our technique generates a two-dimensional schedule while the vertex method yields a one-dimensional schedule. The reason is that our technique has to satisfy dependences on an unbounded polyhedron while the vertex method has to satisfy dependences on a polytope.

6. Conclusion

In our future work, we intend to take into consideration the upper loop bounds in linearized dependence constraints to strengthen our technique preserving the same number of inequalities and equalities in linearized constraints as that yielded with the technique presented in this paper.

References

[14] Lim, W., Lam, M.S., “Maximizing Parallelism and Minimizing Synchronization with Affine Transforms”. In: Conference Record of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1997