2nd International Conference on Parallel and Distributed Computing, Applications and Technologies, 9-11 July, 2001, Taiwan.

On nonsingular loop transformations using SUIF's dependence abstraction

Jingling Xue, School of Computer Science and Engineering, University of New South Wales, Sydney NSW 2052, Australia, jxue@cse.unsw.edu.au

Abstract

In SUIF, the data dependences of a program are represented by dependence vectors consisting of direction values. Based on SUIF's component arithmetic, the legality of a loop transformation can be tested sufficiently. By recasting SUIF's dependence vectors in terms of dependence polyhedra, this paper provides a geometric interpretation for the non-necessity of SUIF's legality test and gives a necessary and sufficient condition for checking the legality of any nonsingular transformation.

Keywords: Parallelising compilers, loop transformation, dependence vector, legality test.

1 Introduction

SUIF is a parallelising compiler developed at Stanford University and utilised by many universities and industrial laboratories around the world [3]. In SUIF, the data dependences of perfectly nested loops are captured by a dependence abstraction that encompasses both distance and direction vectors. A component arithmetic operating on dependence vectors is defined, allowing the legality of a loop transformation to be tested sufficiently. By recasting SUIF's dependence vectors in terms of dependence polyhedra, this paper provides a geometric interpretation of SUIF's component arithmetic, and consequently, a geometric understanding of the non-necessity of SUIF's legality test. A necessary and sufficient condition is presented that allows the legality of any nonsingular loop transformation to be tested exactly. Section 2 introduces SUIF's dependence vectors, component arithmetic and legality test. Section 3 describes how a dependence vector can be represented by a special form of convex polyhedron, which is a cone once the origin is appropriately translated. Section 4 gives a geometric interpretation of SUIF's component arithmetic and explains why SUIF's legality test is sufficient but not necessary. Section 5 presents a necessary and sufficient condition for checking the legality of a nonsingular transformation. Section 6 concludes the paper.

2 SUIF's dependence vectors, component arithmetic and legality test

In SUIF [4], a dependence vector for an n-dimensional perfect loop nest is a vector $\vec{d} = (d_1, \cdots .d_n)$, where the k-th entry d_k corresponds to the k-th loop. Each entry d_k is a possibly infinite range of integers, represented by $[d_k^{\min}, d_k^{\max}]$, where

$$d_k^{\min} \in \mathbb{Z} \cup \{-\infty\}, d_k^{\max} \in \mathbb{Z} \cup \{\infty\}, \text{ and } d_k^{\min} \leqslant d_k^{\max}$$

In general, d_k takes two kinds of values:

Distance Value: If $d_k^{\min}$ and $d_k^{\max}$ are integers, then $[d_k^{\min}, d_k^{\max}]$ represents a finite number of integer values. In the special case when $d_k^{\min} = d_k^{\max} = a$, $[d_k^{\min}, d_k^{\max}] = [a, a]$ degenerates into a singleton distance value, and

Direction Value: If $d_k^{\min} = -\infty$ or $d_k^{\max} = \infty$ or both, then $[d_k^{\min}, d_k^{\max}]$ represents infinitely many integer values.

Figure 1 gives the shorthands used in this paper and by others for representing direction values.

A dependence vector is a *distance vector* if all its entries are distance values and a *direction vector* otherwise. The set of distance vectors, $\mathcal{P}(\vec{d})$, represented by a dependence vector $\vec{d}$ is:

$$\mathcal{P}(\vec{d}) \quad = \quad \{(z_1, \ldots, z_n) \in \mathbb{Z}^n \mid d_k^{\min} \leqslant z_k \leqslant d_k^{\max}\}$$

which contains infinitely many distance vectors if $\vec{d}$ contains a direction value.

Integer Value Range	*This Paper*[1]	*SUIF*[4]	*Tiny*[5]
$[a,\infty]$	$a+$	$[a,\infty]$	unavailable when $a \neq 1$
$[-\infty,a]$	$a-$	$[-\infty,a]$	unavailable when $a \neq -1$
$[1,\infty]$	$+$	$+$	$<$
$[-\infty,-1]$	$-$	$-$	$>$
$[0,\infty]$	$0+$	0^+	$\leqslant$
$[-\infty,0]$	$0-$	0^-	$\geqslant$
$[-\infty,\infty]$	$*$	$\pm$	$*$

Figure 1: Shorthand notations for direction values.

The lexicographic ordering relations, $\succ$, $\succcurlyeq$, $\prec$ and $\preccurlyeq$ on integer vectors[5] can be extended to operate on dependence vectors. Let $\vec{d}$ be a dependence vector. We define:

$$\vec{d} \succ 0 \qquad \text{if } \forall \vec{z} \in \mathcal{P}(\vec{d}) : \vec{z} \succ \vec{0}$$

That is, $\vec{d}$ is lexicographically positive if all its distance vectors are. Likewise for the other three operators. The notation D stands for the set of all dependence vectors in a loop nest, called the *dependence set*. In a sequential program, its dependence vectors are lexicographically positive:

$$\forall\ \vec{d} \in D : \vec{d} \succ \vec{0}$$

This paper considers the class of loop transformations representable by square integer matrices. This class includes well-known transformations such as loop interchange, permutation, reversal and skewing. Since T is linear, the following well-known result is immediate.

Lemma 1 *A nonsingular transformation T is legal iff $\forall \vec{d} \in D : (\forall \vec{z} \in \mathcal{P}(\vec{d}) : T\vec{z} \succ 0)$.*

In[4], Wolf and Lam introduce component arithmetic on dependence vectors so that they can calculate $T\vec{d}$ directly, where $T \in \mathbb{Z}^{n\times n}$ for a dependence vector $\vec{d}$:

$$T\vec{d} = \begin{pmatrix} t_{1,1} & \dots & t_{1,n} \\ \vdots & \vdots & \vdots \\ t_{n,1} & \dots & t_{n,n} \end{pmatrix} \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{n} t_{1,j} d_j \\ \vdots \\ \sum_{j=1}^{n} t_{n,j} d_j \end{pmatrix}$$

and check the legality of a transformation T by examining $T\vec{d}$ only.

In SUIF's component arithmetic, the component addition is defined to be:

$$[a,b] + [c,d] \quad = \quad [a+c, b+d]$$

where for all $x \in \mathbb{Z} \cup \{\infty\}$, $x + \infty$ is ∞ and for all $x \in \mathbb{Z} \cup \{-\infty\}$, $x + (-\infty)$ is $-\infty$. Similarly, multiplication by an integer scalar $s \in \mathbb{Z}$ is defined as:

$$s[a,b] \quad = \quad \begin{cases} [sa, sb] & \text{if } s \geqslant 0 \\ [sb, sa] & \text{if } s < 0 \end{cases}$$

where $s\infty$ is ∞ for positive s, 0 if $s = 0$, and $-\infty$ for negative s, and likewise for $s(-\infty)$. Finally, the component subtraction $a - b$ is defined to be $a + (-1)b$.

Based on component arithmetic, Wolf and Lam[4] developed a sufficient condition for checking the legality of a unimodular transformation. Their result, which applies to non-unimodular transformations, is stated below.

Theorem 1 *([5, Lemma 2.6 and Theorem 2.8]) Let $T \in \mathbb{Z}^{n\times n}$ be a nonsingular transformation. We have (a) $\forall \vec{d} \in D : \vec{z} \in \mathcal{P}(\vec{d}) \Longrightarrow T\vec{z} \in \mathcal{P}(T\vec{d})$. and (b) T is legal if $\forall\ \vec{d} \in D : T\vec{d} \succ \vec{0}$.*

As illustrated in Figure 2 (and pointed out in[4]), SUIF's component arithmetic guarantees that $\mathcal{P}(T\vec{d}) \supseteq \{T\vec{z} \mid \vec{z} \in \mathcal{P}(\vec{d})\}$, where the equality almost never holds. While classifying T_1 and T_2 as being legal, SUIF's legality test is inconclusive when applied to Examples 3 and 4 because $T_3\vec{d}_3 = T_4\vec{d}_4 = (0+, 0+, *) \not\succ \vec{0}$.

Dependence Set	Transformation	Transformed Dependences Using Component Arithmetic	Result of SUIF's Legality Test
$D_1 = \{\vec{d_1}\} = \{(+,+)\}$	$T_1 = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$	$T_1\vec{d_1} = (2+,+)$	Legal
$D_2 = \{\vec{d_2}\} = \{(1,+)\}$	$T_2 = \left(\begin{smallmatrix} 0 & 2 \\ -1 & 2 \end{smallmatrix}\right)$	$T_2\vec{d_2} = (2+,+)$	Legal
$D_3 = \{\vec{d_3}\} = \{(+,1,+)\}$	$T_3 = \left(\begin{smallmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \\ -1 & 1 & 1 \end{smallmatrix}\right)$	$T_3\vec{d_3} = (+0,+0,*)$	Inconclusive
$D_4 = \{\vec{d_4}\} = \{(+,1,+)\}$	$T_4 = \left(\begin{smallmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \\ -1 & -1 & 1 \end{smallmatrix}\right)$	$T_4\vec{d_4} = (+0,+0,*)$	Inconclusive

Figure 2: SUIF's legality test for four examples.

3 Dependence polyhedron abstraction

For practical programs, the entries of a dependence vector are either direction values (those in Figure 1) or singleton distance values (values in $\mathbb{Z}$). The non-singleton distance values $[d_k^{\min}, d_k^{\max}]$, where $d_k^{\min}, d_k^{\max} \in \mathbb{Z}$, are uncommon. Without loss of generality, we assume that a dependence vector component draws its values from:

$$\mathbb{Z} \cup \{*\} \cup (\mathbb{Z} \times \{+,-\}).$$

Note that '+' is a shorthand for '1+' and '−' is a shorthand for '−1−'.

By convention, $\vec{e}_k$ denotes the k-th elementary vector, i.e., the vector whose k-th entry is 1 and the others are 0. Based on Irigoin's work [2] and Darte and Vivien's work [1], $\mathcal{P}(\vec{d})$ is a special form of convex polyhedron, which is a cone once the origin is appropriately translated:

$$\mathcal{P}(\vec{d}) \quad = \quad V(\vec{d}) + \mathsf{cone}\ R(\vec{d})$$

where $V(\vec{d}) \in \mathbb{Z}^n$ and $R(\vec{d}) \subset \mathbb{Z}^n$ are obtained using the algorithm in Figure 3.

DepPoly($\vec{d}$)
$R(\vec{d}) := \emptyset$;
for $k = 1$ *to* n *do*
  if $d_k =$ '$*$' *then*
    $V(\vec{d})_k := 0$; /* Any value in $\mathbb{Z}$ is ok */
    $R(\vec{d}) := R(\vec{d}) + \{\pm\vec{e}_k\}$;
  else if $d_k =$ '$a+$' *then*
    $V(\vec{d})_k := a$;
    $R(\vec{d}) := R(\vec{d}) + \{\vec{e}_k\}$;
  else if $d_k =$ '$a-$' *then*
    $V(\vec{d})_k := a$;
    $R(\vec{d}) := R(\vec{d}) + \{-\vec{e}_k\}$;
  else /* *if* $d_k =$ 'a' *then* */
    $V(\vec{d})_k := a$;

Figure 3: An algorithm for constructing the dependence polyhedron for a dependence vector.

Figure 4 depicts the dependence polyhedra for the eight dependence vectors in Figure 2.

4 Non-necessity of SUIF's legality test

A nonsingular transformation T maps the set of dependences in $\mathcal{P}(\vec{d})$ to the following set:

$$\mathcal{P}_T(\vec{d}) \quad = \quad \{T\vec{z} \mid \vec{z} \in \mathcal{P}(\vec{d})\}$$

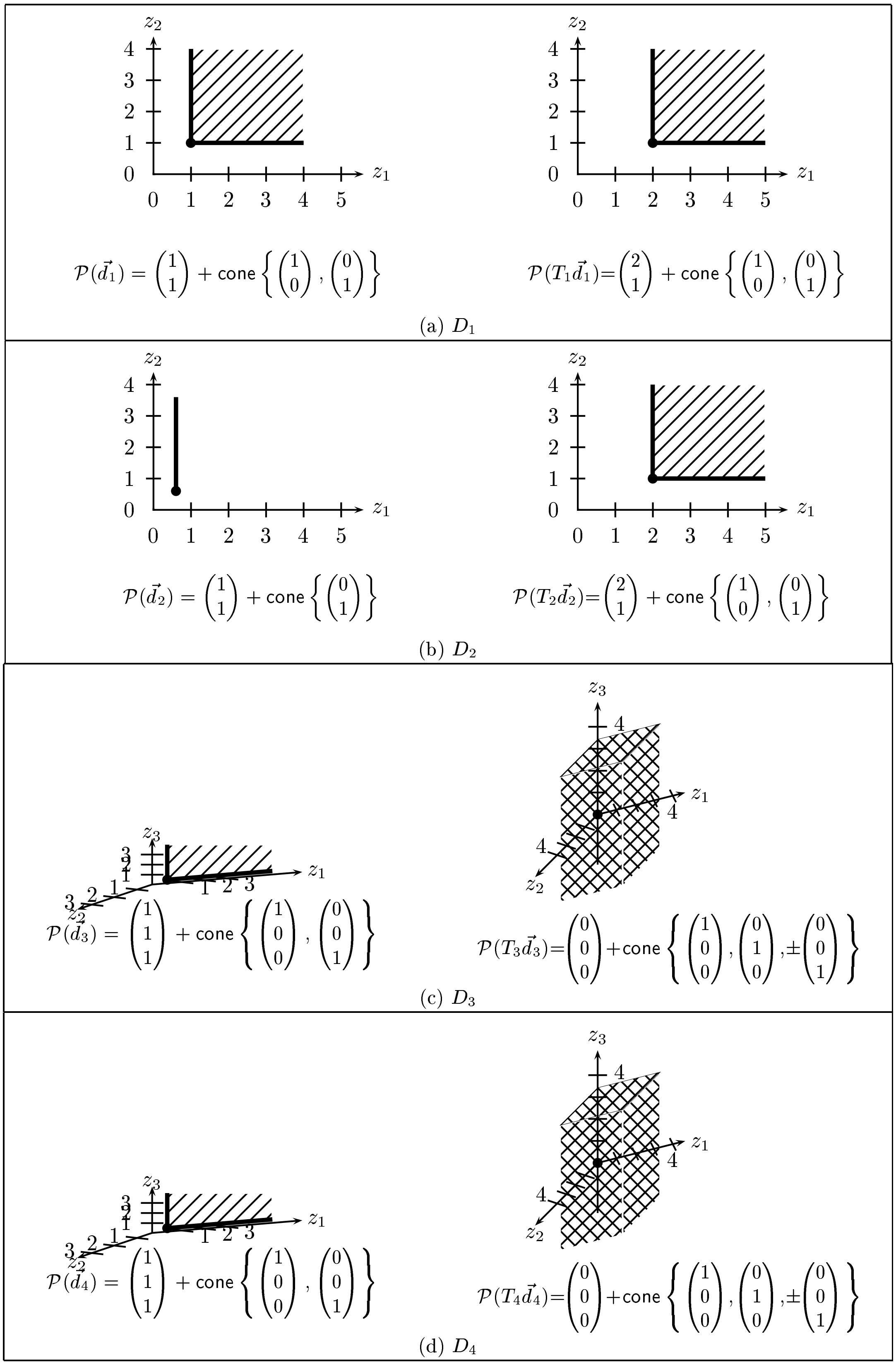

Figure 4: Dependence polyhedra for four examples in Figure 2.

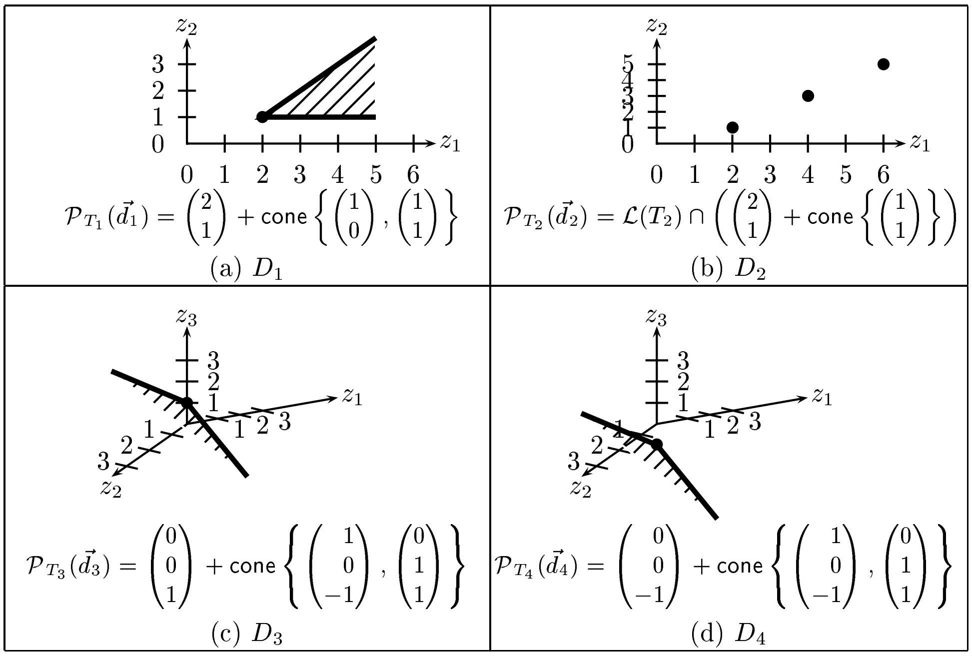

Figure 5: Transformed dependences for the four examples in Figure 2.

In terms of dependence polyhedron abstraction, this set can be specified precisely as:

$$\mathcal{P}_T(\vec{d}) \quad = \quad \mathcal{L}(T) \;\cap\; (TV(\vec{d}) + \mathsf{cone}\; TR(\vec{d}))$$

where $\mathcal{L}(T)$ is the lattice generated by the columns of T [6] and $TR(\vec{d}) = \{T\vec{r} \mid \vec{r} \in R(\vec{d})\}$, i.e., $TR(\vec{d})$ is the image of $R(\vec{d})$ under T.

Figure 5 depicts the transformed dependences for the four examples in Figure 2. $\mathcal{P}_{T_2}(\vec{d}_2)$ is not dense because T_2 is nonunimodular. $\mathcal{P}_{T_1}(\vec{d}_1)$, $\mathcal{P}_{T_3}(\vec{d}_3)$ and $\mathcal{P}_{T_4}(\vec{d}_4)$ are all convex because T_1, T_3 and T_4 are all unimodular; they are cones once the corresponding origins are translated to the points highlighted with fat dots.

The set $\mathcal{P}_T(\vec{d})$ can be represented *exactly* by a single dependence vector iff

- T is unimodular, and
- All rays and lines in $\{T\vec{r} \mid \vec{r} \in R(\vec{d})\}$ are elementary basis vectors.

Therefore, neither of the four sets $\mathcal{P}_{T_1}(\vec{d}_1)$, $\mathcal{P}_{T_2}(\vec{d}_2)$, $\mathcal{P}_{T_3}(\vec{d}_3)$ and $\mathcal{P}_{T_4}(\vec{d}_4)$ displayed in Figure 5 is representable by one single dependence vector.

Based on dependence polyhedron abstraction, a geometric interpretation for SUIF's component arithmetic can be given as follows. The set $\mathcal{P}_T(\vec{d})$ for a dependence vector $\vec{d}$ is approximated by one dependence vector $T\vec{d}$ such that $\mathcal{P}(T\vec{d})$ is the smallest dependence polyhedron enclosing $\mathcal{P}_T(\vec{d})$. The four sets in Figure 5 are approximated as illustrated in Figure 4. Although $\mathcal{P}_{T_1}(\vec{d}_1) \neq \mathcal{P}_{T_2}(\vec{d}_2)$, $\mathcal{P}(T_1\vec{d}_1) = \mathcal{P}(T_2\vec{d}_2)$ holds. Similarly, $\mathcal{P}_{T_3}(\vec{d}_3) \neq \mathcal{P}_{T_4}(\vec{d}_4)$ but $\mathcal{P}(T_3\vec{d}_3) = \mathcal{P}(T_4\vec{d}_4)$ holds. We shall see shortly that T_3 is legal but T_4 is not. It is impossible to distinguish both cases by examining $T_3\vec{d}_3 = T_4\vec{d}_4$ alone.

Based on dependence polyhedron abstraction, $T\vec{d}$ can also be constructed as follows:

1. Construct the smallest dependence polyhedron enclosing $\mathcal{P}_T(\vec{d})$ of the form:

$$\mathcal{P}(T\vec{d}) \quad = \quad V(T\vec{d}) + \mathsf{cone}\; R(T\vec{d})$$

where

$$\begin{aligned} V(T\vec{d}) &= TV(\vec{d}) \\ R(T\vec{d}) &= \{\mathsf{sign}(\vec{r}_k) \times \vec{e}_k \mid \exists\, \vec{r} \in TR(\vec{d}) : \vec{r}_k \neq 0 \text{ for some } k\} \end{aligned}$$

2. Construct $T\vec{d}$ from $\mathcal{P}(T\vec{d})$ by applying the reverse of DepPoly in Figure 3.

This procedure also serves as a proof for our geometric interpretation of SUIF's component arithmetic.

Let us apply this procedure to calculate $T_3\vec{d_3}$ in Figure 4. Based on $\mathcal{P}_{T_3}(\vec{d_3}) = T_3V(\vec{d_3}) + \mathsf{cone}\ T_3R(\vec{d_3})$ in Figure 5(c), we find in the first step $V(T_3\vec{d_3}) = (0,0,1)$ and $R(T_3\vec{d_3}) = \{(1,0,0), (0,1,0), \pm(0,0,1)\}$. In the second step, applying the reverse of DepPoly yields $T_3\vec{d_3} = (0+, 0+, *)$.

5 A necessary and sufficient legality test

The legality of a nonsingular transformation T can be tested exactly. The notation $\vec{x}[i:j]$ denotes the subvector containing the entries $x_i, \ldots, x_j$ in that order.

Theorem 2 *A nonsingular transformation T is legal iff for every dependence vector $\vec{d}$ in the dependence set D of the program, the following two conditions are both true:*

(a) $TV(\vec{d}) \succ \vec{0}$

(b) $\forall\, \vec{r} \in TR(\vec{d}) : \vec{r}[1:k] \not\prec \vec{0}$, *where $(TV(\vec{d}))_k$ is the first nonzero entry of $TV(\vec{d})$.*

In Figure 2, SUIF's legality test was found to be inconclusive for Examples 3 and 4. From Figure 5(c), we obtain $T_3V(\vec{d_3}) = (0,0,1)$ and $T_3R(\vec{d_3}) = \{(1,0,-1),(0,1,1)\}$. From Figure 5(d), we obtain $T_4V(\vec{d_4}) = (0,0,-1)$ and $T_4R(\vec{d_4}) = \{(1,0,-1),(0,1,1)\}$. By applying Theorem 2, we find that T_3 is legal but T_4 is not.

6 Conclusion

In this paper, we recast SUIF's dependence vectors using convex polyhedra, which are cones once the origins are appropriately translated. Based on the dependence polyhedron abstraction, we give a geometric interpretation of SUIF's component arithmetic and explain why SUIF's legality test is sufficient but not necessary. We present a necessary and sufficient condition for testing the legality of any nonsingular loop transformation. The proposed legality test will allow a larger class of loop transformations to be applied.

7 Acknowledgements

This work is supported by an Australian Research Grant A10007149.

References

1. A. Darte and F. Vivien. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. In *1996 International Conference on Parallel Architectures and Compilation Techniques*, pages 281–291, Boston, MA,, 1996.
2. F. Irigoin. Loop reordering with dependence direction vectors. Technical Report EMP-CAI-I A/184, Ecole des Mines de Paris, Nov. 1988.
3. M. Lam et al. *The SUIF Compiler Project*. http://suif.stanford.edu.
4. M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):452–471, Oct. 1991.
5. M. J. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley, 1996.
6. J. Xue. Automating non-unimodular loop transformations for massive parallelism. *Parallel Computing*, 20(5):711–728, 1994.