An Efficient Data Dependence Analysis for Parallelizing Compilers

ZHUYIAN LI, PEN-CHUNG YEW, SENIOR MEMBER, IEEE, AND CHUAN-QI ZHU

Abstract—An efficient and precise data dependence analysis is the key to the success of a parallelizing compiler because it is required in almost all phases of the parallelism detection and enhancement in such compilers. However, existing test algorithms are quite weak in analyzing multidimensional array references, which are usually where the parallelism is in most programs.

There are basically two approaches in dependence testing. The first approach is based on numerical methods. This approach tests array references dimension by dimension, thus it may lose accuracy when references have coupled subscripts. The second approach is based on checking the consistency of a set of inequalities. It tests multiple dimensions simultaneously, and hence can greatly enhance the accuracy. However, this approach is very time consuming.

In this paper, a new algorithm, called the λ test, is presented for an efficient and accurate data dependence analysis of multidimensional array references. It extends the numerical methods to allow all dimensions of array references to be tested simultaneously. Hence, it combines the efficiency and the accuracy of the both approaches. This algorithm has been implemented in Parafuse [12], a Fortran program parallelization restructure developed at the University of Illinois at Urbana-Champaign.

Some experimental results are also presented to show its effectiveness.

Index Terms—Array subscripts, compilers, convex set, data dependence analysis, hyperplanes, linear (in)equalities, loop bounds, program parallelization.

I. INTRODUCTION

A parallelizing compiler relies on data dependence analysis to detect independent operations in a user’s program. For array operands, the analysis needs to check the existence of integer solutions to a linear system obtained from array subscript expressions. In simple cases, this can be done rather easily. But in many cases, this is not so. An analyzer can only resort to checking certain sufficient conditions to determine the absence of a data dependence. If the conditions do not hold, the existence of a data dependence becomes unclear. To err on the safe side, the analyzer must assume that a data dependence exists. Most existing analysis algorithms give only such “incomplete” tests.

There are several well-known data dependence analysis algorithms in practice. They are based on the theories of Diophantine equations and the bounds on real functions [3], [4], [22], [1], [2], [13] calls this class of algorithms numerical methods. Current numerical methods handle only one single dimension. For multidimensional array references, each dimension has to be tested separately. These methods are generally efficient and can detect data independence in many practical cases. Nevertheless, for complicated array subscripts, especially for multidimensional arrays, their test results are often too conservative. More precise results can be achieved by checking the consistency of a linear system of inequalities and equalities. In theory, it could be solved by integer programming (IP) or, with some loss of precision, by linear programming (LP). However, currently known LP and IP algorithms, such as the simplex method [7] and the Karmarkar’s method [8], are aimed at large systems with at least a few hundred constraints and variables. They are not suitable for data dependence analysis where a large number of small systems need to be analyzed. Program verification faces similar problems—methods other than IP or LP are needed to improve its efficiency. Recently, several authors proposed to use methods such as the Fourier–Motzkin elimination and loop residue methods (see, e.g., [6], [17]) for data dependence analysis [11], [20], [13]. In [21], a modified simplex method for IP without considering the cycling problem is considered.

It can yield a conservative solution when no conclusion could be reached after a certain number of iterations.

Most of these methods can determine whether there exists a real-valued solution to a system of equations and inequalities. Even though these methods [11], [20], [13] still do not answer whether integer solutions exist, they are a very good approximation in practical cases. Unfortunately, compared to earlier numerical methods, these methods are very time consuming. The worst-case computing time is exponential in the number of loop indexes even for single-dimensional array references. Even though empirical data on the efficiency of these methods are very limited, some experimental results have shown that average testing time in the Fourier–Motzkin elimination is about 22 to 28 times longer than existing numerical methods [19]. Unless far more efficient algorithms are found, difficulties in testing multidimensional array references will remain.

In this paper, we extend the existing numerical methods to overcome these difficulties. A geometrical analysis reveals that we can take advantage of the regular shape of the convex sets derived from multidimensional arrays in a data depen-

1 Reference [13] correctly pointed out that the general methods should only be used in cases where numerical methods are ineffective.
The general methods proposed before assume very general convex sets; this assumption causes their inefficiency. We have implemented a new algorithm called the \( \Lambda \) test and performed some measurements. Results were quite encouraging (see Section IV).

As in earlier numerical methods, the proposed scheme uses Diophantine equations and bounds of real functions. The major difference lies in the way multiple dimensions are treated. In earlier numerical methods, data areas accessed by two array references are examined dimension by dimension. If the examination of any dimension shows that the two areas representing the subscript expressions are disjoint, there is no data dependence between the two references. However, if each pair of areas appears to overlap in each individual dimension, it is unclear whether there is an overlapped area when all dimensions are considered simultaneously. In this case, a data dependence has to be assumed. Our algorithm treats all dimensions simultaneously. Based on the subscripts, it selects a few suitable "viewing angles" so that it gets an exact view of the data areas. Selection of the viewing angles is rather straightforward and only a few angles are needed in most cases.

We present the rest of our paper as follows. In Section II, we give some examples to illustrate the difficulties in data dependence analysis on multidimensional array references. Some measurement results on a large set of real programs are presented to show the actual frequency of such difficult cases. In Section III, we describe our new algorithm and provide its theoretical background. In Section IV, we present our experimental results and give a brief conclusion.

II. COUPLED SUBSCRIPTS IN MULTIDIMENSIONAL ARRAY REFERENCES

In this section, we identify coupled subscripts as the main difficulty in data dependence analysis on multidimensional array references.

First, we show a couple of examples.

Example 2.1

\[
N = 50 \\
\text{DO } i = 1, N \\
\qquad \text{DO } j = 2, N \\
\qquad \quad A(2 * i + 3 * j + N, 3 * i + j + N - 1) = \ldots \quad (r1) \\
\qquad \ldots = A(i - j + N + 1, 2 * i - j + N - 2) \quad (r2) \\
\text{END END}
\]

In the above, \( r1 \) and \( r2 \) are two references to array \( A \). If there is no data dependence between \( r1 \) and \( r2 \), then both loops in the example can be parallelized. In order to determine whether there is a data dependence between \( r1 \) and \( r2 \), a system of equations and inequalities should be examined. Let \( x_1 = i \) and \( x_2 = j \) for reference \( r1 \), \( x_3 = i \) and \( x_4 = j \) for reference \( r2 \). We equate the subscript expressions in \( r1 \) and \( r2 \) and get the following:

\[
2x_1 + 3x_2 - x_3 + x_4 = 1 \quad (2.1) \\
3x_1 + x_2 - 2x_3 + x_4 = -1 \quad (2.2)
\]

where \( 1 \leq x_1, x_3 \leq 50, 2 \leq x_2, x_4 \leq 50 \), because of the loop bounds. Data dependence exists between \( r1 \) and \( r2 \) if and only if equations (2.1) and (2.2) have common integer solutions within the loop bounds. We can show that such common solutions do not exist in this example. Consider a linear combination of (2.1) and (2.2) by \((2.1) \times 3 - (2.2) \times 2\), which gives a new equation:

\[
7x_2 + x_3 + x_4 = 5. \quad (2.3)
\]

Given the loop bounds, the minimum value on the left hand side of (2.3) is 17, which is larger than the right hand side 5. This means (2.1) and (2.2) have no solutions. Thus, \( r1 \) and \( r2 \) are independent.

However, in most existing numerical methods, each dimension of the array \( A \) is treated separately. Therefore, instead of examining both (2.1) and (2.2) simultaneously, most algorithms examine each equation separately. If any equation can be shown to have no solution within the loop bounds, then there is no data dependence. But if each equation has solutions independently, the algorithm has to assume the existence of data dependence. In (2.1) and (2.2), some loop indexes appear in both dimensions of array \( A \) (either in \( r1 \) or in \( r2 \)). As a result, equations (2.1) and (2.2) share some common unknowns. We say the references have coupled subscripts. Due to such coupled subscripts, while the simultaneous equations do not have common solutions (as shown earlier), each individual equation may have independent solutions. As a matter of fact, \((x_1, x_2, x_3, x_4) = (1, 2, 9, 2)\) is a solution to (2.1), and \((x_1, x_2, x_3, x_4) = (1, 2, 4, 2)\) is a solution to (2.2). Both solutions are within the loop bounds. No algorithms based on dimension-by-dimension approach could detect the independence between \( r1 \) and \( r2 \).

In the next example, we consider the effect of coupled subscripts on the determination of dependence directions [22].

Example 2.2

\[
\text{DO } i = 1, 100 \\
\text{DO } j = 1, 100 \\
\quad A(i, j) = (r1) \\
\quad A(j, i) = (r2) \\
\text{END END}
\]

Obviously, there is data dependence between references \( r1 \) and \( r2 \) in the loop above. However, if the dependence does not cross loop iterations, then the loop can still be parallelized. This is exactly the case with the inner loop: when index \( i \) of the outer loop is fixed, \( r1 \) and \( r2 \) could never access the same data element from different iterations of the inner loop. So the inner loop can be parallelized.

Testing cross-iteration dependences is normally done by examining dependence directions [22]. The procedure of checking these directions starts with setting up the following equations and inequalities:

\[
i_1 = j_1 \quad (2.4) \\
j_1 = i_2 \quad (2.5)
\]
1 \leq i_1, i_2, j_1, j_2 \leq 100 \quad (2.6)

i_1 = i_2, j_1 < j_2. \quad (2.7)

The dependence directions to be examined are specified in (2.7). We have $i_1 = i_2$ because the iteration of the outer loop should be fixed. $j_1 < j_2$ is the condition for a data dependence (from $r_1$ to $r_2$) to cross the iterations of the inner loop. Note that we also need to examine cross-iteration dependences from $r_2$ to $r_1$, for which we should set $j_2 < j_1$. The discussion is similar so we omit it.

It is not hard to see that equations (2.4) and (2.5) do not have common solutions satisfying both (2.6) and (2.7). However, earlier numerical methods cannot detect this, again because they treat each dimension of an array separately. They check on two subsystems. One contains (2.4), (2.6), and (2.7). The other contains (2.5), (2.6), and (2.7). Now that each subsystem has solutions [e.g., $i_1 = j_1 = 100$ for (2.4), (2.6), and (2.7) and $j_1 = i_2 = 1$ for (2.5), (2.6), and (2.7)], the algorithms must assume that the inner loop has cross-iteration dependences. Dimension-by-dimension approach fails to detect impossible dependence directions in this case because some loop indexes (say, index $i$) appear in both dimensions of the array $A$, i.e., there are coupled subscripts.

According to an empirical study reported in [16], coupled subscripts appear quite frequently in real programs. In that study, twelve Fortran program packages which contain 1074 subroutines and more than one hundred thousand lines of statements were examined. The packages include LINPACK, EISPACK, ITPACK, FISHPACK, SPICE, and others. Of all the array references examined, two-dimensional array references account for 36.23 percent, and three-dimensional array references account for 7 percent. The percentage of array references with more than three dimensions are negligible. In more than four thousand pairs of two-dimensional array references with linear subscripts in DO loops, about 46 percent have coupled subscripts. \footnote{The data do not account for references whose subscripts in all dimensions are nonlinear or have unknown symbolic terms.} As for array references with more than two dimensions, only 2 percent have coupled subscripts. The data show several interesting things. First, multidimensional array references are very common. Second, coupled subscripts appear quite frequently. Third, two-dimensional arrays are dominant in references with coupled subscripts. Although a single dimension test could sometimes succeed in detecting data independence despite the coupled subscripts, often it may fail. Therefore, it is important to have an efficient test algorithm to handle coupled subscripts. As the data indicates, two-dimensional array references are especially important.

III. THE $\lambda$ TEST: A NEW ALGORITHM

We present our new algorithm, the $\lambda$ test, in this section. We consider a data dependence problem where subscript expressions are linear in terms of loop indexes. Loop bounds are assumed to be constant, otherwise they are replaced with their closest constant bounds. Dependence directions may also be given if required. We consider only constant loop bounds in this paper because an efficient test in the presence of variable loop bounds is a research topic by itself and is beyond the scope of this paper. Reference [5] gives a very good discussion of dependence tests for single dimensional array references in the presence of variable loop bounds. Handling coupled subscripts in loops with variable bounds is discussed in [12].

Given the data dependence problem as specified, the $\lambda$ test examines a system of equalities and inequalities and determines whether the system has real-valued solutions. (Section IV will discuss integer solutions.) Some notations need be defined before we describe the test.

A. Notation

1) $(r_1, r_2)$ is a pair of references to an array $A$ of $m$ dimensions.

2) $r_1$ is nested in $l_1$ loops. $r_1 = A(f_1(i_1, j_1, \ldots, i_l, j_l), g_1(i_1, j_1, \ldots, i_l, j_l))$, where $i_1, j_1, \ldots, i_l, j_l$ are the loop indexes (from the outermost to the innermost). The loops have constant lower bounds $L_1, L_2, \ldots, L_{l_1}$, and constant upper bounds $U_1, U_2, \ldots, U_{l_1}$.

3) $r_2$ is nested in $l_2$ loops. $r_2 = A(f_2(i_1, j_1, \ldots, i_l, j_l), g_2(i_1, j_1, \ldots, i_l, j_l))$, where $i_1, j_1, \ldots, i_l, j_l$ are the loop indexes (from the outermost to the innermost). The loops have constant lower bounds $L_1, L_2, \ldots, L_{l_2}$, and constant upper bounds $U_1, U_2, \ldots, U_{l_2}$.

4) Equating the subscripts of $r_1$ and $r_2$, we have the following:

\[
\begin{align*}
&f_1(i_1, i_2, \ldots, i_l) = g_1(J_1, J_2, \ldots, J_l) \\
&f_2(i_1, i_2, \ldots, i_l) = g_2(J_1, J_2, \ldots, J_l) \\
&\quad \quad \vdots \\
&f_m(i_1, i_2, \ldots, i_l) = g_m(J_1, J_2, \ldots, J_l).
\end{align*}
\] (3.1)

$(r_1, r_2)$ intersect if and only if (3.1) has an integer solution $(i_1^0, i_2^0, \ldots, i_l^0, j_1^0, j_2^0, \ldots, j_l^0)$ such that $L_{j_1} \leq i_1^0 \leq U_{j_1}, L_{j_2} \leq i_2^0 \leq U_{j_2}, \ldots, L_{j_l} \leq i_l^0 \leq U_{j_l}$.

5) A dependence direction is $\theta = \{<, >, =\}$. A dependence direction vector is $\theta = (\theta_1, \theta_2, \ldots, \theta_m)$ where each $\theta$ is a dependence direction [22]. Suppose $r_1$ and $r_2$ have $l$ common loops of which the innermost is indexed by $i_1$ in $r_1$, (or $j_1$ in $r_2$), $l \leq \min(l_1, l_2)$. $(r_1, r_2)$ is said to intersect with dependence direction vector $\theta = (\theta_1, \theta_2, \ldots, \theta_m)$ if they intersect at $(i_1^0, i_2^0, \ldots, i_l^0, j_1^0, j_2^0, \ldots, j_l^0)$ such that $i_1^0, j_1^0, \ldots, i_l^0, j_l^0$ are exactly coupled with one dependence direction vector.

6) We denote the set of loop indexes in $(r_1, r_2)$ by $IND = \{i_1, i_2, \ldots, i_l, j_1, j_2, \ldots, j_l\}$ and denote the index set of $(r_1, r_2)$ that appears in the array dimension $j_1, 1 \leq j \leq d$, by $IND_j = \{i | i \in IND$ and $i$ appears in either $f_j$ or $g_j\}$.

a) If $IND_{j_1} \cap IND_{j_2} \neq \emptyset$, then dimension $j_1$ and dimension $j_2$ are said to be coupled. $(r_1, r_2)$ is said to have coupled subscripts;

b) If dimension $j_1$ and $j_2$ are coupled, and $j_1$ and $j_2$ are coupled, then $j_1$ and $j_2$ are also coupled.
B. A Geometrical Analysis

Our algorithm is best explained with the aid of a geometrical illustration. Suppose $f_i$’s and $g_i$’s in (3.1) are linear equations with $n$ unknowns, we can rewrite (3.1) as

\[
\begin{align*}
\alpha_1^{(1)} v^{(1)} + \alpha_2^{(1)} v^{(2)} + \cdots + \alpha_n^{(1)} v^{(n)} + c_1 &= 0 \\
\alpha_1^{(2)} v^{(1)} + \alpha_2^{(2)} v^{(2)} + \cdots + \alpha_n^{(2)} v^{(n)} + c_2 &= 0 \\
&\quad\vdots \\
\alpha_1^{(n)} v^{(1)} + \alpha_2^{(n)} v^{(2)} + \cdots + \alpha_n^{(n)} v^{(n)} + c_n &= 0.
\end{align*}
\]

We assume that there are no redundant equations in (3.2). Otherwise, they can simply be eliminated. Furthermore, all array dimensions are assumed to be 2. Otherwise, (3.2) can be broken into several disjoint subsystems. Partial solutions can be obtained for each subsystem and later merged together to form a complete solution. Of course, the number of coupled dimensions $m$ is very small in practice. An especially important case is $m = 2$, for which extra effort should be expended to derive a fast test.

Geometrically, each linear equation is a hyperplane $\pi$ in $\mathbb{R}^n$ space. The intersection $S$ of the $m$ hyperplanes corresponds to the common solutions to all the equations in (3.2). Obviously, if $S$ is empty then there is no data dependence. Checking whether $S$ is empty is trivial in linear algebra. Therefore we consider only nonempty $S$ henceforth. The loop bounds and the given dependence directions correspond to a bounded convex set $V$ in $\mathbb{R}^n$. An equation has a real-valued solution satisfying the loop bounds and the dependence directions if and only if its corresponding hyperplane $\pi$ intersects $V$. A dimension-by-dimension test would be able to determine whether each $\pi$ intersects $V$. What we want to determine whether $S$ itself intersects $V$. If any of the hyperplanes does not intersect $V$, then obviously $S$ cannot intersect $V$. However, even if every hyperplane in (3.2) intersects $V$, it is still possible that $S$ and $V$ are disjoint. In Fig. 1, $\pi_1$ and $\pi_2$ are two such hyperplanes representing two equations from the system, each of which intersects $V$. But the intersection of $\pi_1$ and $\pi_2$ is outside of $V$. If one can find a new hyperplane which contains $S$ but is disjoint from $V$, then it immediately follows that $S$ and $V$ do not intersect. In Fig. 1, $\pi_3$ is such a new hyperplane. The following theorem guarantees that if $S$ and $V$ are disjoint, then there must be a hyperplane in $\mathbb{R}^n$ which contains $S$ and is disjoint from $V$. Furthermore, this hyperplane is a linear combination of the hyperplanes in (3.2). On the other hand, if $S$ and $V$ intersect, then no such linear combination exists.

**Theorem 1:** $S \cap V = \emptyset$ if and only if there exists a hyperplane, $\pi$, which corresponds to a linear combination of equations in (3.2): $(\Sigma_{i=1}^{m} \lambda_i \vec{a}_i \cdot \vec{v}) + \Sigma_{i=1}^{m} \lambda_i c_i = 0$, such that $\pi \cap V = \emptyset$. ($\vec{a}_i \cdot \vec{v}$) denotes the inner product of $\vec{a}_i = (a_1^{(1)}, a_1^{(2)}, \ldots, a_1^{(n)})$ and $\vec{v} = (v^{(1)}, v^{(2)}, \ldots, v^{(n)})$.

**Proof:** See Appendix. $\square$

An array $(\lambda_1, \lambda_2, \ldots, \lambda_m)$ in Theorem 1 determines a hyperplane that contains $S$. There are an infinite number of such hyperplanes. The tricky part in the test is to examine as few hyperplanes as necessary to determine whether $S$ and $V$ intersect. We start from the case of $m = 2$, both for the convenience of presentation and for the practical importance of this case, as described above.

C. The Case of Two-Dimensional Array References

In the case of two dimensional array references, the equations in (3.2) are $f_1 = 0$ and $f_2 = 0$ where $f_i = a_1^{(i)} v^{(1)} + a_2^{(i)} v^{(2)} + \cdots + a_n^{(i)} v^{(n)} + c_i$. For convenience, we directly refer to a linear equation as a hyperplane in $\mathbb{R}^n$. An arbitrary linear combination of the two equations can be written as $\lambda_1 f_1 + \lambda_2 f_2 = 0$. The domain of $(\lambda_1, \lambda_2)$ is the whole $\mathbb{R}^2$ space. Let $f_{\lambda_1, \lambda_2} = \lambda_1 f_1 + \lambda_2 f_2$, that is

$$f_{\lambda_1, \lambda_2} = (\lambda_1 a_1^{(1)} + \lambda_2 a_2^{(2)}) v^{(1)} + (\lambda_1 a_1^{(2)} + \lambda_2 a_2^{(2)}) v^{(2)} + \cdots + (\lambda_1 a_1^{(n)} + \lambda_2 a_2^{(n)}) v^{(n)}.$$

$f_{\lambda_1, \lambda_2}$ can be viewed in two ways. With $(\lambda_1, \lambda_2)$ fixed, $f_{\lambda_1, \lambda_2}$ is a linear function of $(v^{(1)}, v^{(2)}, \ldots, v^{(n)})$ in $\mathbb{R}^n$. With $(v^{(1)}, v^{(2)}, \ldots, v^{(n)})$ fixed, it is a linear function of $(\lambda_1, \lambda_2)$ in $\mathbb{R}^2$.

Furthermore, the coefficient of each $v^{(i)}$ in $f_{\lambda_1, \lambda_2}$ is a linear function of $(\lambda_1, \lambda_2)$ in $\mathbb{R}^2$, i.e., $\psi_i = \lambda_1 a_i^{(1)} + \lambda_2 a_i^{(2)}$.\(\psi_i\) is called a $\psi$ equation. A $\psi$ equation corresponds to a line in $\mathbb{R}^2$ which is called a $\psi$ line.

**Definition:** Each $\psi$ line is the boundary of two half-spaces: $\Psi_i^- = \{(\lambda_1, \lambda_2) | \psi_i^-(\lambda_1, \lambda_2) \geq 0\}$ and $\Psi_i^+ = \{(\lambda_1, \lambda_2) | \psi_i^+(\lambda_1, \lambda_2) \leq 0\}$. There at most $n$ $\psi$ lines which together divide $\mathbb{R}^2$ into at most $2n$ regions. Each region is a cone (Fig. 2) called a $\lambda$ cone.

In each $\lambda$ cone, none of the functions $\psi^{(i)}$ can change the sign of its value (except change to zero in a $\psi$ line). This leads to the following lemma.

**Lemma 1:** Suppose $V$ is defined by loop bounds but not by dependence directions. (Note: we will consider dependence directions later.) If $f_{\lambda_1, \lambda_2} = 0$ intersects $V$ for every $(\lambda_1, \lambda_2)$
in every \(\psi\) line, then \(f_{x_1}, x_2 = 0\) also intersects \(V\) for every \((\lambda_1, \lambda_2)\) in \(R^2\).

**Proof:** 1) From the well-known intermediate value theorem, \(f_{x_1}, x_2 = 0\) intersects \(V\) if and only if \(\min(f_{x_1}, x_2)\) on \(V\).

2) Since \(V\) is bounded and \(f_{x_1}, x_2\) is continuous on \(V\), when \((\lambda_1, \lambda_2)\) is fixed, there exist \(\bar{v}_{\max} \leq V\) such that \(\min(f_{x_1}, x_2) = f_{x_1}, x_2(\bar{v}_{\max})\), and \(\bar{v}_{\max} \in V\) such that \(\max(f_{x_1}, x_2) = f_{x_1}, x_2(\bar{v}_{\max})\).

3) With a fixed \((\lambda_1, \lambda_2)\), it is easy to verify that \(\bar{v}_{\max}\) and \(\bar{v}_{\min}\) are determined by the sign of the coefficient of each \(v^{(i)}\):

\[
v^{(i)}_{\min} = \begin{cases} U^{(i)} & \text{if } \psi^{(i)} > 0, \\ L^{(i)} & \text{if } \psi^{(i)} < 0, \\ x & \text{if } \psi^{(i)} = 0 \end{cases}, \quad v^{(i)}_{\max} = \begin{cases} U^{(i)} & \text{if } \psi^{(i)} > 0, \\ L^{(i)} & \text{if } \psi^{(i)} < 0, \\ y & \text{if } \psi^{(i)} = 0 \end{cases}
\]

where \(L^{(i)}, U^{(i)}\) are the lower bound and the upper bound of \(v^{(i)}\), respectively. \(x, y\) are arbitrary values in \([L^{(i)}, U^{(i)}]\).

4) The coefficient of each \(v^{(i)}\) in \(f_{x_1}, x_2\), \(\psi^{(i)}\), does not change its sign in each \(\lambda\) cone. It follows that \(\bar{v}_{\min}\) and \(\bar{v}_{\max}\) remain the same in each \(\lambda\) cone. Therefore, \(f_{x_1}, x_2(\bar{v}_{\min}) = \lambda_1(a_1^{(1)}v^{(1)}_{\min} + a_2^{(1)}v^{(2)}_{\min} + \cdots + a_n^{(1)}v^{(n)}_{\min} + c_1) + \lambda_2(a_1^{(2)}v^{(1)}_{\min} + a_2^{(2)}v^{(2)}_{\min} + \cdots + a_n^{(2)}v^{(n)}_{\min} + c_2)\) is a linear function of \((\lambda_1, \lambda_2)\) in each \(\lambda\) cone.

5) From the assumption of the lemma, we have \(f_{x_1}, x_2(\bar{v}_{\min}) \leq 0\) for every \((\lambda_1, \lambda_2)\) on the boundaries of each cone. It is a well-known fact in the convex theory that any point \((\lambda_1, \lambda_2)\) in a cone can be expressed as a linear combination of some points on the cone’s boundaries. It immediately follows that \(f_{x_1}, x_2(\bar{v}_{\min}) \leq 0\) is true in each \(\lambda\) cone. Of course it is also true in the whole \(R^2\) space. By the same argument, we have \(f_{x_1}, x_2(\bar{v}_{\max}) \geq 0\) for every \((\lambda_1, \lambda_2)\) in \(R^2\) space. Therefore, for any \((\lambda_1, \lambda_2), f_{x_1}, x_2\) intersects \(V\) in \(R^2\) space.

If \(V\) is defined by loop bounds plus dependence directions, we have a similar lemma. First we should discuss the rules for the selection of \(\bar{v}_{\min}\) and \(\bar{v}_{\max}\) when dependence directions are given. Note that each dependency direction \(\theta\) relates a unique pair of loop indices \(v^{(i)}, u^{(i)}\) which are associated with one of the common loops, \(L^{(i)} = L^{(i)}, U^{(i)} = U^{(i)}\). Obviously \(v^{(i)}\), \(u^{(i)}\), \(v^{(i)}_{\min}\), and \(v^{(i)}_{\max}\) should be chosen such that the partial sum \(a^{(i)}v^{(i)} + a^{(i)}u^{(i)}\) has the minimum value at \(v^{(i)}_{\min}\), \(v^{(i)}_{\max}\), and has the maximum value at \(u^{(i)}_{\max}\), \(u^{(i)}_{\min}\). Where \(v^{(i)}\) is not constrained by a dependence direction, \(v^{(i)}_{\min}\) and \(v^{(i)}_{\max}\) should be chosen as in the proof of Lemma 1.

The following rules determine the minimum and maximum points of a function \(f^{(i)} = a_1^{(i)}v^{(i)} + a_2^{(i)}u^{(i)} + \cdots + a_n^{(i)}u^{(i)} + c\) in the convex set \(V\) defined by both loop bounds and dependence directions.

**Rules:**

i) If \(v^{(i)} = v^{(i)}_{\min}\),

\[
v^{(i)}_{\min} = \begin{cases} L^{(i)} & \text{if } a_1^{(i)} + a_2^{(i)} > 0, \\ U^{(i)} & \text{if } a_1^{(i)} + a_2^{(i)} < 0, \\ x & \text{if } a_1^{(i)} + a_2^{(i)} = 0 \end{cases}
\]

ii) If \(v^{(i)} = v^{(i)}_{\max}\),

\[
v^{(i)}_{\max} = \begin{cases} U^{(i)} & \text{if } a_1^{(i)} + a_2^{(i)} > 0, \\ L^{(i)} & \text{if } a_1^{(i)} + a_2^{(i)} < 0, \\ y & \text{if } a_1^{(i)} + a_2^{(i)} = 0 \end{cases}
\]

where \(L^{(i)} \leq x \leq U^{(i)} - 1\) and \(x + 1 \leq y \leq U^{(i)}\). iiia) Unless \(a_1^{(i)} < 0\) and \(a_2^{(i)} > 0\), we have\( v^{(i)}_{\min} = \begin{cases} L^{(i)} & \text{if } a_1^{(i)} + a_2^{(i)} > 0, \\ U^{(i)} - 1, & \text{if } a_1^{(i)} < 0, \\ x, & \text{if } a_1^{(i)} > 0 \end{cases}\)

\(v^{(i)}_{\max} = \begin{cases} L^{(i)} + 1, & \text{if } a_1^{(i)} > 0, \\ U^{(i)}, & \text{if } a_1^{(i)} < 0, \\ y, & \text{if } a_1^{(i)} = 0 \end{cases}\)

\(v^{(i)}_{\min} = v^{(i)}_{\max}\) where \(v^{(i)}_{\min}\) and \(v^{(i)}_{\max}\) are related by a dependence direction, i.e., \(v^{(i)}_{\min} = a_1^{(i)}v^{(i)} + a_2^{(i)}u^{(i)} + a_3^{(i)}v^{(i)} + a_4^{(i)}u^{(i)}\). From the rules stated above, it is clear that the minimum point and the
maximum point of $f_{\lambda_1, \lambda_2}$ in $V$, in the presence of dependence directions, depend not only on the sign of the coefficient of each $v^{(j)}$ but also on the sign of $\phi^{(i)}$.

**Definition:** $\phi^{(i)} = 0$ is called a $\phi$ equation. Each $\phi$ equation corresponds to a $\phi$ line in $R^2$ space.

There are at most $n/2$ $\phi$ lines. All $\phi$ lines and $\psi$ lines divide $R^2$ space into at most $3n$ regions. Each region is a cone, still called a $\lambda$ cone. We have the following lemma similar to Lemma 1.

**Lemma 2:** Suppose $V$ is defined by loop bounds as well as dependence directions. If $f_{\lambda_1, \lambda_2} = 0$ intersects $V$ for every $(\lambda_1, \lambda_2)$ in every $\psi$ line and every $\phi$ line, then $f_{\lambda_1, \lambda_2} = 0$ also intersects $V$ for every $(\lambda_1, \lambda_2)$ in $R^2$.

**Proof:** Follow the same argument as for Lemma 1. □

As a matter of fact, in order to determine whether $f_{\lambda_1, \lambda_2}$ intersects $V$ for every $(\lambda_1, \lambda_2)$ in a $\phi$ line or a $\psi$ line, it suffices to test a single point in the line. This is from the following lemma.

**Lemma 3:** Given a line in $R^2$ corresponding to an equation $a_{\lambda_1} + b_{\lambda_2} = 0$, if $f_{\lambda_1, \lambda_2} = 0$ intersects $V$ in $R^2$ for any fixed $(\lambda_1, \lambda_2) \neq (0, 0)$ in the line, then for every $(\lambda_1, \lambda_2)$ in the line, $f_{\lambda_1, \lambda_2} = 0$ also intersects $V$.

**Proof:** Note that $f_{\lambda_1, \lambda_2} = 0$ intersects $V$ if and only if $\min (f_{\lambda_1, \lambda_2}) \leq 0 \leq \max (f_{\lambda_1, \lambda_2})$ on $V$. Every point in the line can be expressed as $(\lambda_1^0, \lambda_2^0) \in \Gamma$, where $\lambda_1^0 < \epsilon < \lambda_2^0$. If $\epsilon \geq 0$, then $\bar{\nu}_{\lambda_1, \lambda_2}(\lambda_1^0, \lambda_2^0)$ should be the same as $\bar{\nu}_{\lambda_1^0, \lambda_2^0}(\lambda_1^0, \lambda_2^0)$. Therefore, we have $\min (f_{\lambda_1^0, \lambda_2^0}) = \epsilon \times \min (f_{\lambda_1^0, \lambda_2^0})$. If $\epsilon \leq 0$, then $\bar{\nu}_{\lambda_1, \lambda_2}(\lambda_1^0, \lambda_2^0)$ is the same as $\bar{\nu}_{\lambda_1^0, \lambda_2^0}(\lambda_1^0, \lambda_2^0)$, and we have $\min (f_{\lambda_1^0, \lambda_2^0}) = \epsilon \times \max (f_{\lambda_1^0, \lambda_2^0})$. In either case, we should have $\min (f_{\lambda_1^0, \lambda_2^0}) \leq 0$, because $\min (f_{\lambda_1^0, \lambda_2^0}) \leq 0$ or $\max (f_{\lambda_1^0, \lambda_2^0}) \geq 0$. For the same reason, we have $\max (f_{\lambda_1^0, \lambda_2^0}) \geq 0$.

**Definition:** Given an equation of the form $a_{\lambda_1} + b_{\lambda_2} = 0$ where $a$, $b$ are not zero simultaneously, we define a canonical solution of the equation as follows:

$(\lambda_1, \lambda_2) = (1, 0)$, if $a = 0$;

$(\lambda_1, \lambda_2) = (0, 1)$, if $b = 0$;

$(\lambda_1, \lambda_2) = (b, -a)$, if neither of $a$, $b$ is zero and $b > 0$;

$(\lambda_1, \lambda_2) = (-b, a)$, if neither of $a$, $b$ is zero and $b < 0$.

**Definition:** The $\Lambda$ set is defined to be the set of all canonical solutions to the $\psi$ equations and $\phi$ equations. The hyperplane in $R^2$ corresponding to $\lambda_1 f_1 + \lambda_2 f_2 = 0$ where $(\lambda_1, \lambda_2)$ is a canonical solution in $\Lambda$, is called a $\lambda$ plane.

Obviously, the size of the $\Lambda$ set is at most $n$ if $V$ is defined by loop bounds only, and is at most $3n/2$ if $V$ is defined by loop bounds as well as dependence directions.

**Theorem 2:** $S$ intersects $V$ if and only if every $\lambda$ plane intersects $V$. $V$ is defined by loop bounds only, then there are no more than $n \lambda$ planes. If $V$ is defined by loop bounds as well as dependence directions, then there are no more than $3n/2 \lambda$ planes.

**Proof:** From Lemmas 1–3 and the definition of $\lambda$ planes.

Theorem 2 provides a foundation for the $\lambda$ test in the case of $m = 2$. The test examines the subscripts from the two coupled dimensions, then determines the $\Lambda$ set from the $\phi$ equations and the $\psi$ equations. Each element of $\Lambda$ determines a $\lambda$ plane. Each $\lambda$ plane is tested to see if it intersects $V$, by checking its minimum and maximum values as done in Banerjee-Wolfe test on each single dimension. If any $\lambda$ plane does not intersect $V$, then there is no data dependence. If every $\lambda$ plane intersects $V$, then data dependence should be assumed, unless further tests on integer solutions are to be performed. For the sake of efficiency, computation of the $\Lambda$ set and the test on $\lambda$ planes are performed alternately, i.e., a new element in $\Lambda$ is computed only after it has been tested that the previous $\lambda$ plane intersects $V$. Obviously, repeated canonical solutions can be ignored.

We illustrate the $\lambda$ test by applying it to Examples 2.1 and 2.2 in Section II. In Example 2.1, no dependence directions are given. We only have $\psi$ equations, from which the $\Lambda$ set is easily determined: $\Lambda = \{(3, -2), (1, -3), (2, -1), (1, -1)\}$. A canonical solution $(3, -2)$ determines a $\lambda$ plane which is exactly the linear combination we used to show the absence of data dependence in Example 2.1. In Example 2.2, the $\psi$ equations have two canonical solutions $(1, 0)$ and $(0, 1)$, each corresponding to an original hyperplane in one of the dimensions. The $\phi$ equations are $\lambda_1 - \lambda_2 = 0$ and $-\lambda_1 + \lambda_2 = 0$. Their canonical solutions are both $(1, 1)$. This gives a $\lambda$ plane: $i_1 - i_2 + j_1 - j_2 = 0$ which does not intersect $V$.

In practice, the original hyperplanes $f_1 = 0$ and $f_2 = 0$ are usually $\lambda$ planes to be tested. Due to the regularity of coefficients in subscripts, it is extremely rare that more than one $\lambda$ plane besides $f_1 = 0$ and $f_2 = 0$ needs to be tested. In our experiments, the $\lambda$ test takes less than twice as much time as needed for a dimension-by-dimension test in most programs and total increase in compile time is very insignificant. Hence, the $\lambda$ test is quite efficient.

**D. The Case of $m > 2$**

To generalize the $\lambda$ test, we consider $m$ equations in (3.2) with $m > 2$. All $m$ equations are assumed to be connected; otherwise they can be partitioned into smaller systems. This case is more of theoretical interest than of practical concern since it is rare in real programs to have more than two coupled dimensions. As stated before, we can assume that there are no redundant equations. An arbitrary linear combination of the $m$ equations can be written as

$$\lambda_1 f_1 + \lambda_2 f_2 + \cdots + \lambda_m f_m = 0. \quad (3.3)$$

Let $\sum_{j=1}^{m} \lambda_j a^{(j)} = \lambda_1 f_1 + \lambda_2 f_2 + \cdots + \lambda_m f_m$, then

$$f_{\lambda_1, \lambda_2, \ldots, \lambda_m} = \left( \sum_{j=1}^{m} \lambda_j a^{(j)} \right) v^{(1)} + \left( \sum_{j=1}^{m} \lambda_j a^{(j)} \right) v^{(2)} + \ldots + \left( \sum_{j=1}^{m} \lambda_j a^{(m)} \right) v^{(m)},$$

It is to be determined whether $f_{\lambda_1, \lambda_2, \ldots, \lambda_m} = 0$ intersects $V$ in $R^m$ space for arbitrary $(\lambda_1, \lambda_2, \ldots, \lambda_m)$.

The coefficient of each $v^{(j)}$ in $f_{\lambda_1, \lambda_2, \ldots, \lambda_m}$ is a linear function.
of \((\lambda_1, \lambda_2, \cdots, \lambda_m)\) in \(R^n\), which is \(\psi^{(i)} = \lambda_1 a_{1}^{(i)} + \lambda_2 a_{2}^{(i)} + \cdots + \lambda_m a_{m}^{(i)}\).

**Definition:** The equation \(\psi^{(i)} = 0\), \(1 \leq i \leq n\), is called a \(\psi\) equation. A \(\psi\) equation corresponds to a hyperplane in \(R^n\), called a \(\psi\) plane. Each \(\psi\) plane is the boundary of two half-spaces defined as follows.

**Definition:**

\[
\Psi^+_i = \{ (\lambda_1, \lambda_2, \cdots, \lambda_m) | \psi^{(i)} \geq 0 \},
\]

\[
\Psi^-_i = \{ (\lambda_1, \lambda_2, \cdots, \lambda_m) | \psi^{(i)} \leq 0 \}.
\]

Let \(\phi^{(i,j)}\) be the sum of the coefficients of \(\psi^{(i)}\) and \(\psi^{(j)}\) in \(f_{x_1 x_2 \cdots x_m}\) where \(\psi^{(i)}\) and \(\psi^{(j)}\) are related by a dependence direction, i.e., \(\phi^{(i,j)} = \lambda_1 (a_{1}^{(i)} + a_{1}^{(j)}) + \lambda_2 (a_{2}^{(i)} + a_{2}^{(j)}) + \cdots + \lambda_m (a_{m}^{(i)} + a_{m}^{(j)})\).

**Definition:** The equation \(\phi^{(i,j)} = 0\), \(1 \leq i \leq n\), is called a \(\phi\) equation. A \(\phi\) equation corresponds to a hyperplane in \(R^n\) which is called a \(\phi\) plane. Each \(\phi\) plane is the boundary of two half-spaces defined as follows.

**Definition:**

\[
\Phi^+_i = \{ (\lambda_1, \lambda_2, \cdots, \lambda_m) | \phi^{(i,j)} \geq 0 \},
\]

\[
\Phi^-_i = \{ (\lambda_1, \lambda_2, \cdots, \lambda_m) | \phi^{(i,j)} \leq 0 \}.
\]

**Definition:** If \(V\) is defined by loop bounds only, then a nonempty set \(\cap_{i=1}^n \Psi_i\), where \(\Psi_i \in \{ \Psi^+_i, \Psi^-_i \}\), is called a \(\lambda\) region.

**Definition:** If \(V\) is defined by loop bounds as well as dependence directions, then a nonempty set \((\cap_{i=1}^n \Psi_i) \cap (\cap_{i=1}^n \Phi_i)\) where \(\Psi_i \in \{ \Psi^+_i, \Psi^-_i \}\) and \(\Phi_i \in \{ \Phi^+_i, \Phi^-_i \}\), is called a \(\lambda\) region. Note that the intersection of \(\Phi_i\) is taken for all pairs of index variables which are related by a dependence direction.

**Lemma 4:** Every \(\lambda\) region is a cone in \(R^n\) space. The \(\lambda\) regions in \(R^n\) space have several lines as the frame of their boundaries. Each line (called a \(\lambda\) line) is the intersection of some \(\psi\) planes and \(\phi\) planes.

**Proof:** See Appendix.

**Lemma 5:** If \(f_{x_1 x_2 \cdots x_m} = 0\) intersects \(V\) for every \((\lambda_1, \lambda_2, \cdots, \lambda_m)\) in every \(\lambda\) line, then \(f_{x_1 x_2 \cdots x_m} = 0\) also intersects \(V\) for every \((\lambda_1, \lambda_2, \cdots, \lambda_m)\) in \(R^n\).

**Proof:** Follow the same argument as for Lemmas 1 and 2. Note that every point in a cone can be expressed as a linear combination of some points in the cone's one-dimensional boundaries.

**Lemma 6:** Given a line in \(R^n\) which crosses the origin of the coordinates, if \(f_{x_1 x_2 \cdots x_m} = 0\) intersects \(V\) in \(R^n\) for any fixed \((\lambda_1, \lambda_2, \cdots, \lambda_m) = (0, 0, \cdots, 0)\) in the line, then for every \((\lambda_1, \lambda_2, \cdots, \lambda_m)\) in the line, \(f_{x_1 x_2 \cdots x_m} = 0\) also intersects \(V\).

**Proof:** Follow the same argument as for Lemma 3.

**Theorem 3:** There is a finite set of hyperplanes in \(R^n\) such that \(S\) intersects \(V\) if and only if every hyperplane in the set intersects \(V\). If \(V\) is defined by loop bounds only, then there are no more than \(\binom{m}{n-1}\) hyperplanes in the set. If \(V\) is defined by loop bounds as well as dependence directions, then there are no more than \(\binom{m}{n-1}\) hyperplanes in the set.

**Proof:** See Appendix.

**Table I**

<table>
<thead>
<tr>
<th>Improvement rate</th>
<th>Number of subroutines</th>
</tr>
</thead>
<tbody>
<tr>
<td>20%</td>
<td>7</td>
</tr>
<tr>
<td>10%</td>
<td>3</td>
</tr>
<tr>
<td>4%</td>
<td>10</td>
</tr>
<tr>
<td>0%</td>
<td>5</td>
</tr>
</tbody>
</table>

Lemmas 4 through 6 and Theorem 3 provide a foundation for the \(\lambda\) test in the case of \(m \geq 3\). Obviously, Theorem 3 is also true for the case of \(m = 2\). We do not go into the detail of the \(\lambda\) test in the general case since the discussion is very similar to the case of \(m = 2\). Compared with the theoretical time complexity of methods based on inequality consistency checking, the \(\lambda\) test is clearly much faster, especially for small \(m\).

**IV. Experimental Results and Conclusion**

**A. Experimental Results**

We have implemented the \(\lambda\) test in a program parallelization restructurer, Parafase [10], [9], [15]. Almost all well-known dimension-by-dimension data dependence test algorithms can be found in Parafase. We added the \(\lambda\) test and performed experiments on a numerical package, EISPACK [18]. The package has 56 subroutines, 31 of which were found to have coupled subscripts. Among these 31 subroutines, 25 had their data dependence analysis improved by the \(\lambda\) test.

In our experiments, parallelization of EISPACK subroutines required examination of 72697 pairs of array references. Array dimensions ranged from one to three. Some involved the same pair of array references with different dependence directions. The dimension-by-dimension algorithms in Parafase found 30973 cases that had no data dependence. In many other cases, the algorithms made a conservative assumption that dependencies existed. We then checked whether coupled subscripts were present. If so, the \(\lambda\) test was applied. The \(\lambda\) test found an additional 3214 cases that had no data dependence. So we had an improvement rate of 10.4 percent. The improvement rate can be affected by two factors. First, the frequency of coupled subscripts. Second, the "success rate" of the \(\lambda\) test, by which we mean how often a \(\lambda\) test detects a case where there is no data independence. In our experiment, coupled subscripts were found in 8943 cases where the dimension-by-dimension algorithms assumed data dependences. 3214 of them were found to have no data dependence, so the "success rate" was 36 percent. Table I shows a rough breakdown of an improvement rate for various subroutines. For instance, the first row shows that there are seven subroutines with the improvement rate between 20 to 40 percent in each subroutine.

In our experiments, \(\lambda\) planes always included the hyperplane from each dimension of an array reference. Note that the dimension-by-dimension algorithms had already tested these hyperplanes. The \(\lambda\) tests examined a total number of 8971 additional \(\lambda\) planes in our experiment. That is, almost every \(\lambda\)
test had examined only one additional λ plane. In light of this fact, the additional time needed by a λ test is very small.

Timing results are shown in Table II. Each row shows how much additional time was needed in the λ tests. For instance, the first row shows that there were 12 subroutines in which the λ tests consumed no more than 20 percent additional time. For most of the subroutines (53 out of 56), the λ tests never need more than 100 percent additional time. Additional time was spent mostly on 1) finding coupled dimensions, 2) calculating λ values, and 3) examining each λ plane.

CONCLUSION

The λ test is a new algorithm that can improve data dependence analysis significantly when there are coupled subscripts in multidimensional array references. It achieves the same testing precision as methods based on inequality consistency checking. However, its testing time is much less, especially for small coupled dimensions which occur most frequently. Therefore, it seems to be a promising practical scheme to overcome many difficulties in existing data dependence analysis methods.

As in the methods based on equality consistency checking, the λ test can only determine whether real-valued solutions exist. However, in most practical cases, it is a very good approximation. Reference [14] found that in many common cases the existence of real-valued solutions and the unconstrained integer solutions, i.e., integer solutions without considering loop bounds as well as dependence directions [5], [14] can guarantee the existence of integer solutions that satisfy the loop bounds as well as dependence directions.

There is still much work to be done in data dependence analysis. A general and efficient way to obtain integer solutions is very desirable. We hope the proposed λ test has moved one step closer toward that goal.

APPENDIX

A. Proof of Theorem 1

The ‘‘if’’ part is obvious. We prove the ‘‘only if’’ part. Without loss of generality, assume that \( c_1 = c_2 = \cdots = c_m = 0 \). (Otherwise, simply make a transformation \( \hat{\mathbf{v}} = \mathbf{v} - \overline{\mathbf{v}}_0 \) where \( \overline{\mathbf{v}}_0 \) is in \( S \) which is nonempty.)

\( S \) is a linear subspace and \( \text{Span}(\overline{\mathbf{a}}_1, \overline{\mathbf{a}}_2, \cdots, \overline{\mathbf{a}}_m) \) is the orthogonal complementary space of \( S \). For any \( \overline{\mathbf{w}} \) in \( V \), consider \( \overline{\mathbf{w}}_a \), the projection of \( \overline{\mathbf{w}} \) on \( S \). Since \( \| \overline{\mathbf{w}}_a - \overline{\mathbf{w}} \| \) is a continuous function on \( V \) and \( V \) is bounded, there must exist \( \overline{\mathbf{w}}_0 \) in \( S \) such that \( \| \overline{\mathbf{w}}_a - \overline{\mathbf{w}}_0 \| = \min_{\overline{\mathbf{w}} \notin S} \| \overline{\mathbf{w}}_a - \overline{\mathbf{w}} \| \). This is the minimum distance between \( S \) and \( V \). Since \( \overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a \) is orthogonal to \( S \), it must be in \( \text{Span}(\overline{\mathbf{a}}_1, \overline{\mathbf{a}}_2, \cdots, \overline{\mathbf{a}}_m) \).

Therefore, the equation \( (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{v}}) = 0 \) is a linear combination of the equations in (3.2), i.e., \( \overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a = \lambda_1 \overline{\mathbf{a}}_1 + \cdots + \lambda_m \overline{\mathbf{a}}_m \). Let \( \mathbf{v} \) be the hyperplane defined by the equation \( (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{v}}) = 0 \). According to Claim 1 in the following, \( (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{v}}) > 0 \) for any \( \overline{\mathbf{w}} \) in \( V \), therefore \( \mathbf{v} \cap V = \emptyset \).

**Claim 1:** \( (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{v}}) > 0 \) for any \( \overline{\mathbf{w}} \) in \( V \).

**Proof:** Since \( (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{v}}) = 0 \), \( (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a) > 0 \), and \( (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{v}}) = (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a) + (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{v}} - \overline{\mathbf{w}}_0) + (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a) \), we only need to show that \( (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{v}} - \overline{\mathbf{w}}_0) \geq 0 \).

Consider \( \| \overline{\mathbf{v}}(t) - \overline{\mathbf{w}}_0 \| \) where \( \overline{\mathbf{v}}(t) = t(\overline{\mathbf{w}} - \overline{\mathbf{w}}_0) + \overline{\mathbf{w}}_0 \). Since \( \| \overline{\mathbf{v}} - \overline{\mathbf{w}}_0 \| \) is minimum, and \( \overline{\mathbf{v}}(0) = \overline{\mathbf{w}}_0 \), the function \( f(t) = \| \overline{\mathbf{v}}(t) - \overline{\mathbf{w}}_0 \|^2 \) cannot be decreasing at \( t = 0 \). It follows that \( (\overline{\mathbf{w}}_0 - \overline{\mathbf{w}}_a, \overline{\mathbf{v}} - \overline{\mathbf{w}}_0) \geq 0 \), by taking the differentiation of \( f(t) \). Claim 1 is thus proved. This concludes the proof of Theorem 1.

B. Proof of Lemma 4

First we discuss the boundaries of each λ region. A λ region is a convex set of \( m \) dimensions. By our assumption, there are no redundant equations in (3.2). Therefore the coefficient matrix \( A \) of (3.2) has a rank of \( m \leq n \).

\[
A = \begin{pmatrix}
a^{(1)}_1 & a^{(2)}_1 & \cdots & a^{(n)}_1 \\
a^{(1)}_2 & a^{(2)}_2 & \cdots & a^{(n)}_2 \\
\vdots & \vdots & \ddots & \vdots \\
a^{(1)}_m & a^{(2)}_m & \cdots & a^{(n)}_m
\end{pmatrix}
\]

We ignore the trivial case of \( m = n \). Hence, a λ region is an infinite set. A λ region has at least two \((m - 1)\)-dimensional boundaries. Furthermore, for \( 2 \leq d \leq m \), every \( d \)-dimensional boundary has at least two \((d - 1)\)-dimensional boundaries. A \( d \)-dimensional boundary is in effect a part of the convex set defined by equations

\[
(\lambda_1, \lambda_2, \cdots, \lambda_m)A_d = (0, 0, \cdots, 0)
\]

where \( A_d \) is a nonsingular submatrix of \( A \) with \( d \) columns from \( A \). There is at least one more column from \( A \) which, when added to \( A_d \), expands \( A_d \) into a \( d + 1 \) dimensional nonsingular matrix. Hence every one-dimensional boundary can only be a half-line instead of a line. Furthermore, every λ region has only one extremal point that is \((0, 0, \cdots, 0)\). Therefore a λ region can only be a cone. It is obvious that both halves of a line determined by equation (a.1) where \( d = m - 1 \) are one-dimensional boundaries which belong to different λ regions.

C. Proof of Theorem 3

There are \( n \neq \) equations and at most \( n/2 \neq \) equations. If \( V \) is defined by loop bounds only, then there are only \( n \) columns in matrix \( A \). Hence there are at most \((n+1)\) λ lines. If \( V \) is defined by loop bounds as well as dependence directions, then
there are at most \(3n/2\) columns in matrix \(A\) and there are at most \((m-2)(n-1)/\lambda\) lines. From Lemmas 5 and 6, Theorem 3 follows.

REFERENCES


Zhiyuan Li received the B.S. degree in mathematics in 1981 from Xiamen (Amoy) University, Fujian, China, and the M.S. and Ph.D. degrees in 1985 and 1989, respectively, both from the University of Illinois at Urbana-Champaign in computer science.

In 1982, he was with the Institute of Computing Technology, Academia Sinica, Beijing, China. From 1985 to 1989, he was a research assistant in the Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign. He is now an Assistant Professor of Computer Science at York University, Canada. His research interests include optimizing compilers for supercomputers and computer architecture.

Pea-Chung Yew (S'76-M'80-SM'87) received the B.S.E.E. degree from the National Taiwan University in 1972, the M.S. degree in electrical and computer engineering from the University of Massachusetts, Amherst, in 1977, and the Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 1981.

Since 1981, he has been involved in the architectural design and hardware implementation of the Cedar multiprocessor system. He is currently an assistant professor in the Department of Electrical and Computer Engineering and in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His current research interests are parallel processing, computer architecture, parallelizing compiler performance evaluation.

Dr. Yew is a member of ACM.

Chuan-Qi Zhu was born in Shanghai, China, in 1943. He received the B.S. degree in mathematics from Fudan University, Shanghai, in 1965.

From 1965 to 1982, he was with Fudan University where he designed and built two main frame computers and several special purpose computers. From 1982 to 1984 he was a Visiting Scholar in the Department of Computer Science, University of Illinois at Urbana-Champaign. From 1983 to 1988, he was a member of the Academic Staff at the Center for Supercomputing Research Development at the University of Illinois at Urbana-Champaign, where he worked on hardware and architecture design of the Cedar supercomputer system. He is now a Professor at Fudan University, Shanghai. His research interests include interconnection networks, parallel processing, and computer architecture.