An Efficient Data Dependence Analysis
for Parallelizing Compilers

1. 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 multi-dimensional array references, each dimension has to be tested separatedly. These methods are generally efficient and can detect data independence in many practical cases. Nevertheless, for complicated array subscripts, especially for multi-dimensional 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 the loop residue method (see e. g. [6], [17]) for data dependence analysis [11], [20], [13][1]. 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 indices even for *single-dimensional* array references. Even though empirical data on the efficiency of these methods is 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 multi-dimensional 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 multi-dimensional arrays in a data dependence test. The general methods proposed before assume very general convex sets; this assumption causes their inefficiency. We have implemented a new algorithm called the λ-test and performed some measurements. Results were quite encouraging (see Section 4). 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 2, we give some examples to illustrate the difficulties in data dependence analysis on multi-dimensional 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 3, we describe our new algorithm and provide its theoretical background. In Section 4, we present our experimental results and give a brief conclusion.

2. Coupled Subscripts in Multi-Dimensional Array References In this section, we identify *coupled subscripts* as the main difficulty in data dependence analysis on multi-dimensional array references. First, we show a couple of examples.

Example 2.1 N = 50 DO i = 1, N

```
DO j = 2, N
A(2*i+3*j+N, 3*i+j+N-1) = ...        (r1)
... = A(i-j+N+1, 2*i-j+N-2)          (r2)
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 $x_1 \equiv i$ and $x_2 \equiv j$ for reference r1, $x_3 \equiv i$ and $x_4 \equiv j$ for reference r2. We equate the subscript expressions in r1 and r2 and get the followings:

$$2x_1+3x_2-x_3+x_4=1 \quad (2.1)$$

$$3x_1+x_2-2x_3+x_4=-1 \quad (2.2)$$

where $1 \le x_1,\ x_3 \le 50$, $2 \le x_2,\ x_4 \le 50$, 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) × 3 - (2.2) × 2, which gives a new equation:

$$7x_2 + x_3 + x_4 = 5 \qquad (2.3)$$

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 indices 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 un-knowns. 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

```
DO i = 1, 100 DO j = 1, 100
   A(i, j) =        (r1)
   A(j, i) =        (r2) 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.

$$i_1 = j_2 \qquad (2.4)$$

$$j_1 = i_2 \tag{2.5}$$

$$1 \le i_1, i_2, j_1, j_2 \le 100 \tag{2.6}$$

$$i_1 = i_2,\ j_1 < j_2 \tag{2.7}$$

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 r1 to r2) to cross the iterations of the inner loop. Note that we also need to examine cross-iteration dependences from r2 to r1, 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. $i_1 = j_2 = 100$ 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 indices (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, FISHPAK, SPICE and others. Of all the array references examined, two-dimensional array references account for 36.23%, and three-dimensional array references account for 7%. 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% have coupled subscripts*[2]. As for array references with more than two dimensions, only 2% have coupled subscripts. The data show several interesting things. First, multi-dimensional 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.

3. The λ-test: a new algorithm We present our new algorithm, the λ-*test*, in this section. We consider a data dependence problem where subscript expressions are linear in terms of loop indices. 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. [5] gave 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 λ-test examines a system of equalities and inequalities and determines whether the system has *real-valued* solutions. (Section 4 will discuss integer solutions.) Some notations need be defined before we describe the test.

3.1 Notation (r_1, r_2), is a pair of references to an array A of m dimensions. r_1 is nested in l_1 loops. $r_1 \equiv A(f_1(i_1, i_2, ..., i_{l_1}), f_2(i_1, i_2, ..., i_{l_1}), ..., f_m(i_1, i_2, ..., i_{l_1}))$, where $i_1, i_2, ..., i_{l_1}$ are the loop indices (from the outermost to the innermost). The loops have constant lower bounds $L_{i_1}, L_{i_2}, ..., L_{i_{l_1}}$, and constant upper bounds $U_{i_1}, U_{i_2}, ..., U_{i_{l_1}}$. r_2 is nested in l_2 loops. $r_2 \equiv A(g_1(j_1, j_2, ..., j_{l_2}), g_2(j_1, j_2, ..., j_{l_2}), ..., g_m(j_1, j_2, ..., j_{l_2}))$, where $j_1, j_2, ..., j_{l_2}$ are the loop indices (from the outermost to the innermost). The loops have constant lower bounds $L_{j_1}, L_{j_2}, ..., L_{j_{l_2}}$, and constant upper bounds $U_{j_1}, U_{j_2}, ..., U_{j_{l_2}}$. Equating the subscripts of r_1 and r_2, we have the following:

$$\begin{cases} f_1(i_1, i_2, ..., i_{l_1}) = g_1(j_1, j_2, ..., j_{l_2}) \\ f_2(i_1, i_2, ..., i_{l_1}) = g_2(j_1, j_2, ..., j_{l_2}) \\ \qquad\qquad ...... \\ f_m(i_1, i_2, ..., i_{l_1}) = g_m(j_1, j_2, ..., j_{l_2}) \end{cases} \tag{3.1}$$

(r_1, r_2) intersect if and only if (3.1) has an integer solution $(i_1^0, i_2^0, ..., i_{l_2}^0, j_1^0, j_2^0, ..., j_{l_2}^0)$ such that $L_{i_1} \le i_1^0 \le U_{i_1}$, $L_{i_2} \le i_2^0 \le U_{i_2}$, ..., $L_{j_1} \le j_1^0 \le U_{j_1}$, $L_{j_2} \le j_2^0 \le U_{j_2}$, ..., etc. (r_1, r_2) is said to intersect at $(i_1^0, i_2^0, ..., i_{l_2}^0, j_1^0, j_2^0, ..., j_{l_2}^0)$. Assume that f_i and g_i are all linear. A *dependence direction* is $\theta \in \{<, >, =\}$. A *dependence direction vector* is $\vec{\theta} = (\theta_1, \theta_2, ..., \theta_{\mathbf{n}})$, where each $\theta_{\mathbf{i}}$ is a dependence direction [22]. Suppose r_1 and r_2 have l_c common loops of which the innermost is indexed by i_{l_c} in r_1, (or j_{l_c} in r_2), $l_c \le \min(l_1, l_2)$. (r_1, r_2) is said to intersect with dependence direction vector $\vec{\theta} = (\theta_1, \theta_2, ..., \theta_{\mathbf{l_c}})$, if they intersect at $(i_1^0, i_2^0, ..., i_{l_1}^0, j_1^0, j_2^0, ..., j_{l_2}^0)$ such that $i_1^0\ \theta_1\ j_1^0$, $i_2^0\ \theta_2\ j_2^0$, ..., $i_{l_c}^0\ \theta_{\mathbf{l_c}}\ j_{l_c}^0$. (r_1, r_2) may intersect with more than one dependence direction vector.

We denote the set of loop indices in (r_1, r_2) by IND $\equiv \{i_1, i_2, ..., i_{l_1}, j_1, j_2, ..., j_{l_2}\}$ and denote the index set of (r_1, r_2) that appears in the array dimension j, $1 \le j \le d$, by $IND_j \equiv \{\text{i} \mid \text{i} \in \text{IND and i appears in either } f_j \text{ or } g_j\}$.

(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_2 and j_3 are coupled, then j_1 and j_3 are also *coupled*.

3.2 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:

$$\begin{array}{l} a_1^{(1)}v^{(1)} + a_1^{(2)}v^{(2)} + \cdots + a_1^{(n)}v^{(n)} + c_1 = 0 \\ a_2^{(1)}v^{(1)} + a_2^{(2)}v^{(2)} + \cdots + a_2^{(n)}v^{(n)} + c_2 = 0 \\ \qquad \cdots \quad \cdots \\ a_m^{(1)}v^{(1)} + a_m^{(2)}v^{(2)} + \cdots + a_m^{(n)}v^{(n)} + c_m = 0 \end{array} \tag{3.2}$$

We assume that there are no redundant equations in (3.2). Otherwise, they can simply be eliminated. Further, all array dimensions are assumed to be coupled. 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 π in $\mathbf{R^n}$ 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 $\mathbf{R^n}$. An equation has a real-valued solution satisfying the loop bounds and the dependence directions if and only if its corresponding hyperplane π intersects V. A dimension-by-dimension test would be able to determine whether each π intersects V. What we want is 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 Figure 1, π_1 and π_2 are two such hyperplanes representing two equations from the system, each of which intersects V. But the intersection of π_1 and π_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 Figure 1, π_3 is such a new hyperplane. The following theorem guarantees that if S and V are disjoint, then there must be a hyperplane in $\mathbf{R^n}$ which contains S and is disjoint from V. Further, 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, π, which corresponds to a linear combination of equations in (3.2): $<\sum_{i=1}^{m} \lambda_i \vec{a_i}, \vec{v}> + \sum_{i=1}^{m} \lambda_i c_i = 0$, such that $\pi \cap V = \emptyset$. $<\vec{a_i}, \vec{v}>$ denotes the inner product of $\vec{a_i} \equiv (a_i^{(1)}, a_i^{(2)}, ..., a_i^{(n)})$ and $\vec{v} \equiv (v^{(1)}, v^{(2)}, ..., v^{(n)})$.

[proof] See Appendix. □ An array $(\lambda_1, \lambda_2, ..., \lambda_m)$ in Theorem 1 determines a hyperplane that contains S. There are an infinite number of such hyperplanes. The tricky part in the λ-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.

3.3 The case of 2-dimensional array references In the case of 2-dimensional array references, the equations in (3.2) are $f_1 = 0$ and $f_2 = 0$, where $f_i \equiv a_i^{(1)}v^{(1)} + a_i^{(2)}v^{(2)} + \cdots + a_i^{(n)}v^{(n)} + c_i$. For convenience, we directly refer to a linear equation as a hyperplane in $\mathbf{R^n}$. An arbitrary linear combination of the two equations can be written as $\lambda_1 f_1 + \lambda_2 f_2 = 0$. The domain of (λ_1, λ_2) is the whole $\mathbf{R^2}$ space. Let $f_{\lambda_1,\lambda_2} \equiv \lambda_1 f_1 + \lambda_2 f_2$, that is $f_{\lambda_1,\lambda_2} \equiv (\lambda_1 a_1^{(1)} + \lambda_2 a_2^{(1)})v^{(1)} + (\lambda_1 a_1^{(2)} + \lambda_2 a_2^{(2)})v^{(2)} + ... + (\lambda_1 a_1^{(n)} + \lambda_2 a_2^{(n)})v^{(n)}$

f_{λ_1,λ_2} can be viewed in two ways. With (λ_1, λ_2) fixed, f_{λ_1,λ_2} is a linear function of $(v^{(1)}, v^{(2)}, ..., v^{(n)})$ in $\mathbf{R^n}$. With $(v^{(1)}, v^{(2)}, ..., v^{(n)})$ fixed, it is a linear function of (λ_1, λ_2) in $\mathbf{R^2}$. Further, the coefficient of each $v^{(i)}$ in f_{λ_1,λ_2} is a linear function of (λ_1, λ_2) in $\mathbf{R^2}$, i. e. $\psi^{(i)} \equiv \lambda_1 a_1^{(i)} + \lambda_2 a_2^{(i)}$.

Definition The equation $\psi^{(i)} = 0$, $1 \le i \le n$, is called a ψ*-equation*. A ψ-equation corresponds to a line in $\mathbf{R^2}$ which is called a ψ*-line*.

Definition Each ψ-line is the boundary of two half-spaces: $\Psi_i^+ \equiv \{(\lambda_1, \lambda_2) \mid \psi^{(i)} \ge 0\}$ and $\Psi_i^- \equiv \{(\lambda_1, \lambda_2) \mid \psi^{(i)} \le 0\}$. There are at most n ψ-lines which together divide $\mathbf{R^2}$ into at most 2n regions. Each region is a cone (Figure 2) called a λ*-cone. In each λ-cone, none of the functions $\psi^{(i)}$ can change the sign of its value (except change to zero in a ψ-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 (λ_1, λ_2) in every ψ-line, then $f_{\lambda_1,\lambda_2} = 0$ also intersects *V* for every (λ_1, λ_2) in $\mathbf{R^2}$.

[Proof] From the well-known intermediate value theorem, $f_{\lambda_1,\lambda_2} = 0$ intersects *V* if and only if $\min(f_{\lambda_1,\lambda_2}) \le 0 \le$

$\max(f_{\lambda_1,\lambda_2})$ on V. Since V is bounded and f_{λ_1,λ_2} is continuous on V, when (λ_1,λ_2) is fixed, there exist $\vec{v}_{\min} \in V$ such that $\min(f_{\lambda_1,\lambda_2}) = f_{\lambda_1,\lambda_2}(\vec{v}_{\min})$, and $\vec{v}_{\max} \in V$ such that $\max(f_{\lambda_1,\lambda_2}) = f_{\lambda_1,\lambda_2}(\vec{v}_{\max})$. With a fixed (λ_1, λ_2), it is easy to verify that $\vec{v}_{\max}$ and $\vec{v}_{\min}$ are determined by the sign of the coefficient of each $v^{(i)}$:

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

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

where $L^{(i)}$, $U^{(i)}$ are the lower bound and the upper bound of $v^{(i)}$ respectively. x, y are arbitrary values in $[L^{(i)}, U^{(i)}]$. The coefficient of each $v^{(i)}$ in f_{λ_1,λ_2}, $\psi^{(i)}$, does not change its sign in each λ-cone. It follows that $\vec{v}_{\max}$ and $\vec{v}_{\min}$ remain the same in each λ-cone. Therefore, $f_{\lambda_1,\lambda_2}(\vec{v}_{\min}) \equiv \lambda_1(a_1^{(1)}v^{(1)}_{\min} + a_1^{(2)}v^{(2)}_{\min} + \cdots + a_1^{(n)}v^{(n)}_{\min} + c_1) + \lambda_2(a_2^{(1)}v^{(1)}_{\min} + a_2^{(2)}v^{(2)}_{\min} + \ldots + a_2^{(n)}v^{(n)}_{\min} + c_2)$ is a linear function of (λ_1,λ_2) in each λ-cone. From the assumption of the lemma, we have $f_{\lambda_1,\lambda_2}(\vec{v}_{\min}) \le 0$ for every (λ_1, λ_2) on the boundaries of each cone. It is a well-known fact in the convex theory that any point (λ_1, λ_2) in a cone can be expressed as a linear combination of some points on the cone's boundaries. It immediately follows that $f_{\lambda_1,\lambda_2}(\vec{v}_{\min}) \le 0$ is true in each λ-cone. Of course it is also true in the whole $\mathbf{R^2}$ space. By the same argument, we have $f_{\lambda_1,\lambda_2}(\vec{v}_{\max}) \ge 0$ for every (λ_1, λ_2) in $\mathbf{R^2}$ space. Therefore, for any (λ_1, λ_2), f_{λ_1,λ_2} intersects V in $\mathbf{R^n}$ 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 $\vec{v}_{\min}$ and $\vec{v}_{\max}$ when dependence directions are given. Note that each dependence direction θ relates a unique pair of loop indices $v^{(i)}$, $v^{(j)}$ which are associated with one of the common loops. $L^{(i)} = L^{(j)}$, $U^{(i)} = U^{(j)}$. Obviously $v^{(i)}_{\min}$, $v^{(j)}_{\min}$, $v^{(i)}_{\max}$ and $v^{(j)}_{\max}$ should be chosen such that the *partial sum* $a^{(i)}v^{(i)} + a^{(j)}v^{(j)}$ has the minimum value at $(v^{(i)}_{\min}, v^{(j)}_{\min})$ and has the maximum value at $(v^{(i)}_{\max}, v^{(j)}_{\max})$. Where $v^{(i)}$ is not constrained by a dependence direction, $v^{(i)}_{\max}$ and $v^{(i)}_{\min}$ should be chosen as in the proof of Lemma 1. The following rules determine the minimum and maximum points of a function f, $f \equiv a^{(1)}v^{(1)} + a^{(2)}v^{(2)} + \ldots + a^{(n)}v^{(n)} + \mathrm{c}$, in the convex set V defined by both loop bounds and dependence directions.

Rules If $v^{(i)} = v^{(j)}$,

$$v_{\min}^{(i)} = v_{\min}^{(j)} = \begin{cases} L^{(i)}, & \text{if } a^{(i)} + a^{(j)} > 0, \\ U^{(i)}, & \text{if } a^{(i)} + a^{(j)} < 0, \\ x, & \text{if } a^{(i)} + a^{(j)} = 0 \end{cases}$$

$$v_{\max}^{(i)} = v_{\max}^{(j)} = \begin{cases} U^{(i)}, & \text{if } a^{(i)} + a^{(j)} > 0, \\ L^{(i)}, & \text{if } a^{(i)} + a^{(j)} < 0, \\ y, & \text{if } a^{(i)} + a^{(j)} = 0 \end{cases}$$

If $v^{(i)} < v^{(j)}$ (i.e. $v^{(i)} \leq v^{(j)}$ - 1), consider (iia) and (iib). Unless $a^{(i)} < 0$ and $a^{(j)} > 0$, we have

$$v_{\min}^{(i)} = \begin{cases} L^{(i)}, & \text{if } a^{(i)} > 0, \\ U^{(i)} - 1, & \text{if } a^{(i)} < 0, \\ x, & \text{if } a^{(i)} = 0 \end{cases}$$

$$v_{\min}^{(j)} = \begin{cases} L^{(j)} + 1, & \text{if } a^{(j)} > 0, \\ U^{(j)}, & \text{if } a^{(j)} < 0, \\ y, & \text{if } a^{(j)} = 0 \end{cases}$$

where $L^{(i)} \leq$ x $\leq U^{(i)}$-1 and x+1 $\leq$ y $\leq U^{(i)}$. If $a^{(i)} < 0$ and $a^{(j)} > 0$, rules in (iia) would conflict with the dependence direction. Instead, since we have

$$a^{(i)} v^{(i)} + a^{(j)} v^{(j)} \geq a^{(i)} v^{(i)} + a^{(j)} (v^{(i)} + 1) = (a^{(i)} + a^{(j)}) v^{(i)} + a^{(j)}$$

the rules in this case should be

$$v_{\min}^{(i)} = \begin{cases} L^{(i)}, & \text{if } a^{(i)} + a^{(j)} > 0, \\ U^{(i)} - 1, & \text{if } a^{(i)} + a^{(j)} < 0, \\ x, & \text{if } a^{(i)} + a^{(j)} = 0 \end{cases}$$

$$v_{\min}^{(j)} = \begin{cases} L^{(j)} + 1, & \text{if } a^{(i)} + a^{(j)} > 0, \\ U^{(j)}, & \text{if } a^{(i)} + a^{(j)} < 0, \\ y, & \text{if } a^{(i)} + a^{(j)} = 0 \end{cases}$$

where $L^{(i)} \leq$ x $\leq U^{(i)}$-1 and x+1 $\leq$ y $\leq U^{(i)}$. Rules for $v_{\max}^{(i)}$ and $v_{\max}^{(j)}$ are similar. Consider (iic) and (iid). Unless $a_{(i)}$>0 and $a^{(j)}$<0, we have

$$v_{\max}^{(i)} = \begin{cases} U^{(i)} - 1, & \text{if } a^{(i)} > 0, \\ L^{(i)}, & \text{if } a^{(i)} < 0, \\ x, & \text{if } a^{(i)} = 0 \end{cases}$$

$$v_{\max}^{(j)} = \begin{cases} U^{(j)}, & \text{if } a^{(j)} > 0, \\ L^{(j)} + 1, & \text{if } a^{(j)} < 0, \\ y, & \text{if } a^{(j)} = 0 \end{cases}$$

where $L^{(i)} \le x \le U^{(i)}$-1 and x+1 ≤ y ≤ $U^{(i)}$. If $a^{(i)}>0$ and $a^{(j)}<0$,

$$v^{(i)}_{\max} = \begin{cases} U^{(i)}-1, & \text{if } a^{(i)}+a^{(j)}>0, \\ L^{(i)}, & \text{if } a^{(i)}+a^{(j)}<0, \\ x, & \text{if } a^{(i)}+a^{(j)}=0 \end{cases}$$

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

where $L^{(i)} \le x \le U^{(i)}$-1 and x+1 ≤ y ≤ $U^{(i)}$. Rules for the case $v^{(i)} > v^{(j)}$ are symmetrical to those in (ii). Details are omitted.

□ Let $\phi^{(i,j)}$ be the sum of the coefficients of $v^{(i)}$ and $v^{(j)}$ in f_{λ_1,λ_2}, where $v^{(i)}$ and $v^{(j)}$ are related by a dependence direction, i. e., $\phi^{(i,j)} \equiv \lambda_1(a_1^{(i)} + a_1^{(j)}) + \lambda_2(a_2^{(i)} + a_2^{(j)})$. From the rules stated above, it is clear that the minimum point and the maximum point of f_{λ_1,λ_2} in *V*, in the presence of dependence directions, depend not only on the sign of the coefficient of each $v^{(i)}$ but also on the sign of $\phi^{(i,j)}$.

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

There are at most n/2 φ-lines. All φ-lines and ψ-lines divide $\mathbf{R}^2$ space into at most 3n regions. Each region is a cone, still called a λ-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 (λ_1, λ_2) in every ψ-line and every φ-line, then $f_{\lambda_1,\lambda_2} = 0$ also intersects *V* for every (λ_1, λ_2) in $\mathbf{R}^2$.

[Proof] Follow the same argument as for Lemma 1. □ As a matter of fact, in order to determine whether f_{λ_1,λ_2} intersects *V* for every (λ_1, λ_2) in a φ-line or a ψ-line, it suffices to test a single point in the line. This is from the following lemma.

Lemma 3 Given a line in $\mathbf{R}^2$ corresponding to an equation $a\lambda_1 + b\lambda_2 = 0$, if $f_{\lambda_1,\lambda_2} = 0$ intersects *V* in $\mathbf{R}^\mathbf{n}$ for *any* fixed $(\lambda_1^0, \lambda_2^0) \ne (0, 0)$ in the line, then for *every* (λ_1, λ_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 $\min(f_{\lambda_1,\lambda_2}) \le 0 \le \max(f_{\lambda_1,\lambda_2})$ on *V*. Every point in the line can be expressed as $(\varepsilon\lambda_1^0, \varepsilon\lambda_2^0)$, $-\infty < \varepsilon < +\infty$. If $\varepsilon \ge 0$, then $\vec{v}_{\min}$ for $f_{\varepsilon\lambda_1^0,\varepsilon\lambda_2^0}$ in *V* should be the same as for $f_{\lambda_1^0,\lambda_2^0}$. Therefore we have $\min(f_{\varepsilon\lambda_1^0,\varepsilon\lambda_2^0}) = \varepsilon \times \min(f_{\lambda_1^0,\lambda_2^0})$. If $\varepsilon \le 0$ then $\vec{v}_{\min}$ for $f_{\varepsilon\lambda_1^0,\varepsilon\lambda_2^0}$ in *V* is the same as $\vec{v}_{\max}$ for $f_{\lambda_1^0,\lambda_2^0}$, and we have $\min(f_{\varepsilon\lambda_1^0,\varepsilon\lambda_2^0}) = \varepsilon \times \max(f_{\lambda_1^0,\lambda_2^0})$. In either case, we should have $\min(f_{\varepsilon\lambda_1^0,\varepsilon\lambda_2^0}) \le 0$, because of $\min(f_{\lambda_1^0,\lambda_2^0})$

≤ 0 or $\max(f_{\lambda_1^0,\lambda_2^0}) \ge 0$. For the same reason, we have $\max(f_{\varepsilon\lambda_1^0,\varepsilon\lambda_2^0}) \ge 0$. □

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 Λ set is defined to be the set of all canonical solutions to the ψ-equations and ϕ-equations. The hyperplane in $\mathbf{R}^{\mathbf{n}}$ corresponding to $\lambda_1 f_1 + \lambda_2 f_2 = 0$, where (λ_1, λ_2) is a canonical solution in Λ, is called a λ-*plane*. Obviously, the size of the Λ 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 λ-plane intersects *V*. If *V* is defined by loop bounds only, then there are no more than *n* λ-planes. If *V* is defined by loop bounds as well as dependence directions, then there are no more than 3n/2 λ-planes.

[Proof] From Lemmas 1-3 and the definition of λ-planes. □ Theorem 2 provides a foundation for the λ-test in the case of m = 2. The test examines the subscripts from the two coupled dimensions, then determines the Λ set from the ϕ-equations and the ψ-equations. Each element of Λ determines a λ-plane. Each λ-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 λ-plane does not intersect *V*, then there is no data dependence. If every λ-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 Λ set and the test on λ-planes are performed alternately, i.e., a new element in Λ is computed only after it has been tested that the previous λ-plane intersects with *V*. Obviously, repeated canonical solutions can be ignored. We illustrate the λ-test by applying it to Examples 2.1 and 2.2 in Section 2. In Example 2.1, no dependence directions are given. We only have ψ-equations, from which the Λ set is easily determined: $\Lambda = \{(3, -2), (1, -3), (2, -1), (1, -1)\}$. A canonical solution (3, -2) determines a λ-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 ψ-equations have two canonical solutions (1, 0) and (0, 1), each corresponding to an original hyperplane in one of the dimensions. The ϕ-equations are $\lambda_1 - \lambda_2$

= 0 and - $\lambda_1 + \lambda_2 = 0$. Their canonical solutions are both (1, 1). This gives a λ-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 λ-planes to be tested. Due to the regularity of coefficients in subscripts, it is extremely rare that more than one λ-plane besides $f_1 = 0$ and $f_2 = 0$ needs be tested. In our experiments, the λ-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 λ test is quite efficient.

3.4 The case of m > 2 To generalize the λ-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

$$\lambda_1 f_1 + \lambda_2 f_2 + \cdots + \lambda_m f_m = 0 \tag{3.3}$$

Let $f_{\lambda_1,\lambda_2,\ldots,\lambda_m} \equiv \lambda_1 f_1 + \lambda_2 f_2 + \ldots + \lambda_m f_m$, then $f_{\lambda_1,\lambda_2,\ldots,\lambda_m} \equiv (\sum_{j=1}^{m} \lambda_j a_j^{(1)}) v^{(1)} + (\sum_{j=1}^{m} \lambda_j a_j^{(2)}) v^{(2)} + \ldots + (\sum_{j=1}^{m} \lambda_j a_j^{(n)}) v^{(n)}$

It is to be determined whether $f_{\lambda_1,\lambda_2,\cdots,\lambda_m} = 0$ intersects V in $\mathbf{R}^\mathbf{n}$ space for arbitrary $(\lambda_1, \lambda_2, \cdots, \lambda_m)$. The coefficient of each $v^{(i)}$ in $f_{\lambda_1,\lambda_2,\ldots,\lambda_m}$ is a linear function of $(\lambda_1, \lambda_2, \ldots, \lambda_m)$ in $\mathbf{R}^\mathbf{m}$, which is $\psi^{(i)} \equiv \lambda_1 a_1^{(i)} + \lambda_2 a_2^{(i)} + \ldots + \lambda_m a_m^{(i)}$.

Definition The equation $\psi^{(i)} = 0$, $1 \le i \le n$, is called a ψ-*equation*. A ψ-equation corresponds to a hyperplane in $\mathbf{R}^\mathbf{m}$, called a ψ-*plane*.

Each ψ-plane is the boundary of two half-spaces defined as follows. $\Psi_i^+ \equiv \{(\lambda_1, \lambda_2, \ldots, \lambda_m) \mid \psi^{(i)} \ge 0\}$,

$\Psi_i^- \equiv \{(\lambda_1, \lambda_2, \ldots, \lambda_m) \mid \psi^{(i)} \le 0\}$.

Let $\phi^{(i,j)}$ be the sum of the coefficients of $v^{(i)}$ and $v^{(j)}$ in $f_{\lambda_1,\lambda_2,\ldots,\lambda_m}$, where $v^{(i)}$ and $v^{(j)}$ are related by a dependence direction, i. e., $\phi^{(i,j)} \equiv \lambda_1 (a_1^{(i)} + a_1^{(j)}) + \lambda_2 (a_2^{(i)} + a_2^{(j)}) + \ldots + \lambda_m (a_m^{(i)} + a_m^{(j)})$.

Definition The equation $\phi^{(i)} = 0$, $1 \le i \le n$, is called a ϕ-*equation*. A ϕ-equation corresponds to a hyperplane in $\mathbf{R}^\mathbf{m}$ which is called a ϕ-*plane*.

Each ϕ-plane is the boundary of two half-spaces defined as follows. $\Phi_i^+ \equiv \{(\lambda_1, \lambda_2, \ldots, \lambda_m) \mid \psi^{(i)} \ge 0\}$,

$\Phi_i^- \equiv \{(\lambda_1, \lambda_2, \ldots, \lambda_m) \mid \psi^{(i)} \le 0\}$.

Definition If V is defined by loop bounds only, then a nonempty set $\bigcap_{i=1}^{n} \Psi_i$, where $\Psi_i \in \{\Psi_i^+, \Psi_i^-\}$, is called a λ-

region.

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

Lemma 4 Every λ-region is a cone in $\mathbf{R}^m$ space. The λ-regions in $\mathbf{R}^m$ space have several lines as the frame of their boundaries. Each line (called a λ-*line*) is the intersection of some ψ-planes and φ-planes.

[Proof] See Appendix. □

Lemma 5 If $f_{\lambda_1,\lambda_2,...,\lambda_m} = 0$ intersects V for every $(\lambda_1, \lambda_2, ..., \lambda_m)$ in every λ-line, then $f_{\lambda_1,\lambda_2,...,\lambda_m} = 0$ also intersects V for every $(\lambda_1, \lambda_2, ..., \lambda_m)$ in $\mathbf{R}^{\mathbf{m}}$.

[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 1-dimensional boundaries. □

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

[Proof] Follow the same argument as for Lemma 3.

Theorem 3 There is a finite set of hyperplanes in $\mathbf{R}^{\mathbf{n}}$ 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 $\begin{bmatrix} n \\ m-1 \end{bmatrix}$ hyperplanes in the set. If V is defined by loop bounds as well as dependence directions, then there are no more than $\begin{bmatrix} 3n/2 \\ m-1 \end{bmatrix}$ hyperplanes in the set.

[Proof] See Appendix. □ Lemmas 4 through 6 and Theorem 3 provide a foundation for the λ-test in the case of m ≥ 3. Obviously, Theorem 3 is also true for the case of m = 2. We do not go into the detail of the λ-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 λ-test is clearly much faster, especially for small m.

4. Experimental Results and Conclusion

4.1 Experimental results We have implemented the λ-test in a program parallelization restructurer, Parafrase [10], [9], [15]. Almost all well-known dimension-by-dimension data dependence test algorithms can be found in Parafrase. We added the λ-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 λ-test. In our experiments, parallelization of EISPACK subroutines required examination of 72,697 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 Parafrase found 30,973 cases that had no data dependence. In many other cases, the algorithms made a conservative assumption that dependences existed. We then checked whether coupled subscripts were present. If so, the λ-test was applied. The λ-test found an additional 3,214 cases that had no data dependence. So we had an improvement rate of 10.4%. The improvement rate can be affected by two factors. First, the frequency of coupled subscripts. Second, the "success rate" of the λ-test, by which we mean how often a λ-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%*. Table 1 shows a rough breakdown of an improvement rate for various subroutines. For instance, the first row shows that there are 7 subroutines with the improvement rate between 20% to 40% in each subroutine. In our experiments, λ-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 λ-tests examined a total number of 8971 *additional* λ-planes in our experiment. That is, almost every λ-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 2. 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% additional time. For most of the subroutines (53 out of 56), the λ-tests never need more than 100% additional time. Additional time was spent mostly on (1) finding coupled dimensions, (2) calculating λ-values, and

(3) examining each λ-plane.

4.2 Conclusion The λ-test is a new algorithm that can improve data dependence analysis significantly when there are coupled subscripts in multi-dimensional 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 inequality consistency checking, the λ-test can only determine whether real-valued solutions exist. However, in most practical cases, it is a very good approximation. [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.

REFERENCES

[1] J. R. Allen, "Dependence Analysis for Subscripted Variables And Its Application to Program Transformations," Ph.D. Thesis, Department of Mathematical Sciences, Rice University, Houston, TX, April 1983.

[2] J. R. Allen and K. Kennedy, "Automatic Translation of Fortran Programs to Vector Form," Dept. of Computer Science, Rice University, Houston, TX, Rice Comp TR84-9, July, 1984.

[3] U. Banerjee, "Data Dependence in Ordinary Programs," Department of Computer Sciences, University of Illinois at Urbana-Champaign, Rpt. No. 76-837, Nov. 1976.

[4] U. Banerjee, "Speedup of Ordinary Programs," Ph.D Thesis, Department of Computer Sciences, University of Illinois at Urbana-Champaign, Rpt. No. UIUCDCS-R-79-989, 1979.

[5] U. Banerjee, *Dependence Analysis for Supercomputing*, Kluwer Academic Publishers, Norwell, Mass., 1988.

[6] P. Cousot, N. Halbwacks, "Automatic discovery of linear restraints among variables of a program," *Conf. Record of the 5-th Annual ACM Symp. on Principle of Program Languages*, 1978.

[7] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.

[8] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Proc. 16th Annual ACM Symposium on the Theory of Computing*, 1984, pp. 302-311.

[9] D. Kuck, R. Kuhn, B. Leasure and M. Wolfe, "The structure of an advanced vectorizer for pipelined processors," *Proc. of COMPSAC 80, The 4th Int'l Computer Software and Applications Conf.*, pp. 709-715, Oct. 1980.

[10] D. Kuck, "Automatic program restructuring for high-speed computations," *Proc. of CONPAR 81, Conf. on Analyzing Problem-Classes and Programming for Parallel Computing*, Nurnberg, F. R. Germany, ed. by W. Handler, June 1981 (Springer-Verlag).

[11] R. Kuhn, "Optimization And Interconnection Complexity for: Parallel Processors, Single-Stage Networks, And Decision Trees," Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Rpt. No. UIUCDCS-R-80-1009, Feb. 1980.

[12] Z. Li, "Intraprocedural and Interprocedural Data Dependence Analysis for Parallel Computing," Ph.D. Dissertation in preparation, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1989.

[13] A. Lichnewsky and F. Thomasset, "Introducing symbolic problem solving techniques in the dependence testing phase of a vectorizer," *Proc. of 1988 Int'l Conf. on Supercomputing*, St. Malo, France, July 1988.

[14] Z. Li and P.-C. Yew, Practical methods for exact data dependence analysis, to appear in *Proceedings of the Second Workshop on Languages and Compilers for Parallel Computing*, in the series *Research Monographs in Parallel and Distributed Computing* , Pitman Publishing.

[15] D. Padua, D. Kuck, and D. Lawrie, "High-speed multiprocessors and compilation techniques," *IEEE Trans. Computers*, Vol. C-29, No. 9, Sept. 1980, pp. 763-776.

[16] "An empirical study on array subscripts and data dependences," *Proc. of 1989 Int' Conf. on Parallel Processing*, St. Charles, Illinois, Aug. 1989.

[17] R. Shostak, "Deciding linear inequalities by computing loop residues," *ACM Journal* vol. 28(4), pp. 769-779 (Oct 1981).

[18] B. J. Smith, et al., *Matrix Eigensystem Routines - Eispack Guide*, Springer-Verlag, Heidelberg, 1976.

[19] R. Triolet, "Interprocedural Analysis for Program Restructuring with Parafrase," CSRD Rpt. No. 538, University of Illinois at Urbana-Champaign, Dec. 1985.

[20] R. Triolet, F. Irigoin, P. Feautrier, "Direct parallelization of CALL statements," *Proceedings of the ACM SIGPLAN '86 Symp. on Compiler Construction*, SIGPLAN Notices, Vol. 21, No. 6, June 1986.

[21] D. R. Wallace, "Dependence of multi-dimensional array references," *Proc. of 1988 Int'l Conf.*

on Supercomputing, St. Malo, France, July 1988.

[22] M. J. Wolfe, "Optimizing Supercompilers for Supercomputers," Ph.D. thesis, Univ. of Illinois at Urbana-Champaign, DCS Report No. UIUCDCS- R-82-1105, Oct. 1982.

Appendix

I. Proof of Theorem 1 The "If" part is obvious. We prove the "Only If" part. Without loss of generality, assume that $c_1 = c_2 = ... = c_m = 0$. (Otherwise, simply make a transformation $\vec{v}' = \vec{v} - \vec{v}_0$, where $\vec{v}_0$ is in S which is nonempty.) S is a linear subspace and $Span(\vec{a}_1, \vec{a}_2, ..., \vec{a}_m)$ is the orthogonal complementary space of S. For any $\vec{w}$ in V, consider $\vec{p}_w$, the projection of $\vec{w}$ on S. Since $||\vec{p}_w - \vec{w}||$ is a continuous function on V and V is bounded, there must exist $\vec{w}_0$ in V such that $||\vec{p}_{w_0} - \vec{w}_0|| = \min_{\vec{w} \in V} ||\vec{p}_w - \vec{w}||$. This is the minimum distance between S and V. Since $\vec{w}_0 - \vec{p}_{w_0}$ is orthogonal to S, it must be in $Span(\vec{a}_1, \vec{a}_2, ..., \vec{a}_m)$. Therefore the equation $<\vec{w}_0 - \vec{p}_{w_0}, \vec{v}> = 0$ is a linear combination of the equations in (3.2), i. e. $\vec{w}_0 - \vec{p}_{w_0} = \lambda_1\vec{a}_1 + \lambda_2\vec{a}_2 + ... + \lambda_m\vec{a}_m$. Let π be the hyperplane defined by the equation $<\vec{w}_0 - \vec{p}_{w_0}, \vec{v}> = 0$. According to Claim 1 in the following, $<\vec{w}_0 - \vec{p}_{w_0}, \vec{w}> > 0$ for any $\vec{w}$ in V, therefore $\pi \cap V = \emptyset$.

Claim 1

$<\vec{w}_0 - \vec{p}_{w_0}, \vec{w}> > 0$ for any $\vec{w}$ in V.

[Proof] Since $<\vec{w}_0 - \vec{p}_{w_0}, \vec{p}_{w_0}> = 0$, $<\vec{w}_0 - \vec{p}_{w_0}, \vec{w}_0 - \vec{p}_{w_0}> > 0$, and $<\vec{w}_0 - \vec{p}_{w_0}, \vec{w}> = <\vec{w}_0 - \vec{p}_{w_0}, \vec{w}_0 - \vec{p}_{w_0}> + <\vec{w}_0 - \vec{p}_{w_0}, \vec{w} - \vec{w}_0> + <\vec{w}_0 - \vec{p}_{w_0}, \vec{p}_{w_0}>$, we only need to show that $<\vec{w}_0 - \vec{p}_{w_0}, \vec{w} - \vec{w}_0> \geq 0$. Consider $||\vec{u}(t) - \vec{p}_{w_0}||$, where $\vec{u}(t) \equiv t(\vec{w} - \vec{w}_0) + \vec{w}_0$. Since $||\vec{w}_0 - \vec{p}_{w_0}||$ is minimum, and $\vec{u}(0) = \vec{w}_0$, the function $f(t) \equiv ||\vec{u}(t) - \vec{p}_{w_0}||^2$ cannot be decreasing at t = 0. It follows that $<\vec{w}_0 - \vec{p}_{w_0}, \vec{w} - \vec{w}_0> \geq 0$, by taking the differentiation of f(t). Claim 1 is thus proved. This concludes the proof of Theorem 1. □

II. 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 ≤ n.

$$\mathbf{A} \equiv \begin{bmatrix} a_1^{(1)} & a_1^{(2)} & \cdots & a_1^{(n)} \\ a_2^{(1)} & a_2^{(2)} & \cdots & a_2^{(n)} \\ & \cdots & \cdots & \\ a_m^{(1)} & a_m^{(2)} & \cdots & a_m^{(n)} \end{bmatrix}$$

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. Further, for $2 \le d \le m$, 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

$$(\lambda_1, \lambda_2, ..., \lambda_m)\mathbf{A_d} = (0, 0, ..., 0) \qquad \text{(a.1)}$$

where $\mathbf{A_d}$ is a nonsingular submatrix of **A** with d columns from **A**. There is at least one more column from **A** which, when added to $\mathbf{A_d}$, expands $\mathbf{A_d}$ into a d+1 dimensional nonsingular matrix. Hence every 1-dimensional boundary can only be a half-line instead of a line. Further, every λ-region has only one extremal point that is (0, 0, ..., 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 1-dimensional boundaries which belong to different λ-regions. □

III. Proof of Theorem 3 There are n ψ-equations and at most n/2 ϕ-equations. If V is defined by loop bounds only, then there are only n columns in matrix **A**. Hence there are at most $\begin{bmatrix} n \\ m-1 \end{bmatrix}$ 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 $\begin{bmatrix} 3n/2 \\ m-1 \end{bmatrix}$ λ-lines. From Lemmas 5 and 6, Theorem 3 follows. □