Abstract

A dependence relation between two data references is linear if it generates dependence vectors that are linear functions of the loop indices. A linear dependence relation often induces a large number of dependence vectors. Empirical studies also show that linear dependencies often intermix with uniform dependencies in loops [5, 6]. These factors make it difficult to analyze such loops and extract the inherent parallelism. In this paper, we propose to manipulate such dependencies in the dependence vector space and summarize the large number of dependence vectors with their convex hull. The convex hull, as a profile of the dependence vectors, can be used to deduce many important properties of the vectors. We will show how to find the convex hull and then apply it to loop parallelization. The proposed approach will be compared with other schemes.

1. Introduction

According to an empirical study of scientific and engineering programs [5], 44% of two-dimensional array references have coupled subscripts. Two subscript expressions are coupled if they contain the same loop index. Coupled subscripts may cause result in linear data dependencies, i.e., dependencies that are linear functions of the loop indices. A linear dependence relation usually generates a large number of distinct dependence vectors. An ability to exploit the relationships among these vectors is very important, especially for loop parallelization [10].

Recently several techniques aimed at loops with linear dependencies were developed [2, 8, 11]. However these schemes do not deal directly with the relationships among the dependence vectors, and they consider linear dependencies separately from uniform dependencies. Based on our study on EISPACK [6], linear dependencies usually intermix with uniform dependencies in nested loops. Thus, to extract useful parallelism, we must exploit the relationships among linear dependence vectors and manipulate both linear and uniform dependencies simultaneously. A framework which unifies both uniform and linear dependencies and allows the manipulation of a large number of dependence vectors is thus desirable.

In this paper we introduce such a framework, which handles loops in the dependence vector space instead of in the iteration space. To simplify the manipulation of the large number of dependence vectors generated by a linear dependence, we propose to summarize these vectors with their enclosing convex hull, called the dependence vector convex hull (DVCH). Finding the DVCH corresponding to a nested loop is referred to as dependence profiling.

In the dependence vector space, there is no distinction between uniform and linear dependence vectors, and the relationships among the vectors can be observed closely. From the DVCH, many properties of the dependence vectors can be deduced from its shape, location, and corners. This thus allows us to effectively parallelize loops with linear dependencies. Furthermore, many traditional parallelization techniques can be extended directly to handle such loops.

In this paper, we will show how to construct the DVCH from a given loop and discuss how to parallelize a loop using its DVCH. Due to space limitation, we will consider only on doubly nested loops. Most discussions will be based on the loop model $L$ shown below:

$$\begin{align*}
\text{DO } I & = 1, N \\
\text{DO } J & = l(I), u(I) \\
S_2 : & A(f_1(I,J), f_2(I,J)) \equiv \cdots \\
S_1 : & \cdots \equiv A(f_3(I,J), f_4(I,J)) \cdots \\
\text{ENDDO} \\
\text{ENDDO}
\end{align*}$$

In the loop, the functions $f_1(I,J)$, $f_2(I,J)$, $f_3(I,J)$ and

---

*This work was supported in part by the National Science Council under Grant NSC 83-0408-E-007-092 and 84-2213-E-007-014.*
The dependency region is in one of the regions. This region is a two-dimensional space. The inequalities in Eq. (2) then define the set which contains every distinct dependence vector in the iteration space, called the dependence vector space.

There is a dependence between statements $S_i$ and $S_l$ if we can find four integers $(i_1, j_1, i_2, j_2)$ satisfying the system of Diophantine equations:

$$
\begin{align*}
    f_1(i_1, j_1) &= f_3(i_2, j_2) \\
    f_2(i_1, j_1) &= f_4(i_2, j_2)
\end{align*}
$$

and the set of inequalities formed from the loop bounds:

$$
\begin{align*}
    1 &\leq i_1 \leq N \\
    l(i_1) &\leq j_1 \leq u(i_1) \\
    1 &\leq i_2 \leq N \\
    l(i_2) &\leq j_2 \leq u(i_2)
\end{align*}
$$

In most cases, the solution to the Diophantine equations can be expressed in terms of two free variables, $x$ and $y$. It is often written as $(i_1, j_1, i_2, j_2) = (g_1, g_2, g_3, g_4)$, where each $g_i$ is a linear function of $x$ and $y$. It means that the instance of $S_i$ in the iteration $(g_1, g_2)$ and the instance of $S_l$ in the iteration $(g_3, g_4)$ both reference the same memory location. The resultant dependence induces a dependence vector

$$(g_3 - g_1, g_4 - g_2) = (a_1, a_2)x + (b_1, b_2)y + (c_1, c_2),$$

where $a_1, b_1$ and $c_2$ are all integers. If the expression does not involve free variables $x$ and $y$, then every iteration in the loop will have the same dependence vector. Such a dependence relation is a uniform dependence; otherwise it is a linear dependence. Define the dependence vector set $D(L)$ to be the set which contains every distinct dependence vector in loop $L$.

The two free variables, $x$ and $y$, can be viewed as forming a two-dimensional space. The inequalities in Eq. (2) then divide the space into regions. The final solution to the dependence relation is in one of the regions. This region will be called the dependence region and is denoted $\text{DR}(L)$. Specifically, it is bounded by the following constraints:

$$
\begin{align*}
    1 &\leq g_1 \leq N \\
    l(g_1) &\leq g_2 \leq u(g_1) \\
    1 &\leq g_3 \leq N \\
    l(g_3) &\leq g_4 \leq u(g_3),
\end{align*}
$$

where $(g_1, g_2, g_3, g_4)$ is the general solution to the Diophantine equations in Eq. (1) [8]. The dependence region formed from the above constraints will be a convex polygon on the $xy$-plane. Each grid point inside $\text{DR}(L)$ corresponds to one iteration in $L$ which induces a data dependence.

### 3. Dependence profiling

Given a doubly nested loop $L$, every dependence vector of the loop can be mapped to a grid point in a two-dimensional lattice space, called the dependence vector space. Fig. 1(b) shows the dependence vectors of the loop in Fig. 1(a) in the dependence vector space. As can be seen in the figure, the relationship among the vectors is very regular and ready to be exploited.

Since $\text{DR}(L)$ is a convex region in the $xy$-plane and the dependence relations are all linear, the region formed by the dependence vectors in the dependence vector space is also a convex region. Define the dependence vector convex hull of a loop $L$, denoted $\text{DVCH}(L)$, to be the convex hull in the dependence vector space which encloses the above region. Then, every grid point inside $\text{DVCH}(L)$ will correspond to a grid point inside $\text{DR}(L)$. In addition a grid corner of $\text{DR}(L)$ must also be mapped to a corner of $\text{DVCH}(L)$, except in some degenerated cases. Thus to find $\text{DVCH}(L)$, all we need to do is to first find the smallest convex hull that encloses all the grid points in $\text{DR}(L)$ and whose corners are grid points themselves. Such a convex hull will be called the grid dependence region of $L$, denoted $\text{GDR}(L)$. Using $\text{GDR}(L)$, we can then compute their corresponding points.

![Figure 1. Dependence vectors in the dependence vector space.](image-url)
Given: A doubly nested loop \( L \) and the solution to its Diophantine equations.

1. Find \( DR(L) \).
2. Find \( GDR(L) \).
3. Compute \( H = \{(a_1, a_2) x + (b_1, b_2) y + (c_1, c_2) | (x, y) \in GDR(L)\} \)
4. Return the convex hull of the points in \( H \).

**Figure 2. The algorithm DVCH.**

Step 1 of the algorithm can be solved by any algorithm for finding the intersection region of a set of constraints [3]. This step takes \( O(m \log m) \) time, where \( m \) is the number of inequalities. \( GDR(L) \) in the second step can be found with the algorithm introduced in [7]. This step will have a time complexity of \( O(m \log m + \max(m, n)L) \), where \( n \) is the number of points inside \( GDR(L) \). In step 3, \( H \) is obtained with a time complexity of \( O(n) \). Finally, DVCH(L) can be constructed with any algorithm for finding the convex hull of a set of points [3], which has a complexity of \( O(n \log n) \).

For the loop in Fig. 1(a), the DVCH is shown in thick lines in Fig. 1(b) and contains \{(-4, 1), (-8, 3), (-8, 2), (-4, -4)\}. We can see that the DVCH provides a profile of all the dependence vectors in \( L \). Note that the shape of the DVCH for a doubly nested loop is not necessarily a two-dimensional lattice. It can be a point or a line [6].

**4. Loop parallelization with the DVCH**

In this section we will study one application of the DVCH concept, namely loop parallelization. Since most loop parallelization techniques are based on distance vectors [1, 9], we will discuss briefly how to find the DVCH for the distance vectors first. Then the generalized cycle shrinking method [4] is extended to parallelize loops with linear dependencies. We will show how to determine the best wavefronting scheme using the DVCH. It should be noted that the information contained in the DVCH is very rich, and it is possible to extend other techniques with the DVCH to handle loops with linear dependencies.

**4.1. DVCH for dependence distance vectors**

Consider the loop \( L \) in Section 2. The general expression for the dependence vectors is \((d_1, d_2) = (g_3 - g_1, g_4 - g_2)\). The dependence vector represents a loop-carried flow dependence if \((g_1, g_2) \prec (g_3, g_4)\). The vector is lexicographically positive and is equal to the distance vector of the dependence [1]. On the other hand, the dependence vector represents a loop-carried anti-dependence if \((g_3, g_4) \prec (g_1, g_2)\). The vector is lexicographically negative and, by complementing the signs of the components, it becomes the distance vector. Note that the same pair of array references may induce not only many dependence vectors but anti- and flow dependence vectors simultaneously. This is another complication for manipulating nested loops with linear dependencies.

To obtain the DVCH for the distance vectors, we can proceed as follows.

1. Find the convex hull that encloses every flow dependence vector and call it \( FLOW(L) \).
2. Find the convex hull that encloses every anti-dependence vector and call it \( ANTI(L) \).
3. Obtain the DVCH as \( FLOW(L) \cup \{(-d_1, -d_2) | (d_1, d_2) \in ANTI(L)\} \).

The algorithms to find \( FLOW(L) \) and \( ANTI(L) \) can be found in [6].

**4.2. Wavefronthing through the DVCH**

One common approach to parallelizing nested loops is to arrange the iterations into waves of concurrent executions. All iterations in the same wave are executed in parallel, while iterations in different waves are executed sequentially. Assume that \((a, b)\) is the normal vector of a wavefront for a doubly nested loop \( L \). The wavefront is a legal wavefront, i.e., all iterations in the same wave can be executed in parallel, if every dependence vector \((d_1, d_2)\) in \( L \) satisfies the condition: \( ad_1 + bd_2 \geq 1 \) [4]. Define \( c = \min\{(ad_1 + bd_2) | (d_1, d_2) \in D(L)\} \). Then every iteration \((i, j)\), where \( 0 \leq ai + bj \leq c - 1 \), can be executed at time step 1, and every iteration satisfying \( c \leq ai + bj \leq 2c - 1 \) can be executed at time step 2, and so on. Thus, the total execution steps of \( L \) using this wavefront is equal to

\[
\left\lceil \frac{\max\{(ai_1 + bj_1) - (ai_2 + bj_2)\}}{c} \right\rceil,
\]

where \((i_1, j_1)\) and \((i_2, j_2)\) are iterations. It follows that the best wavefront for the given loop is the one which minimizes the total execution steps, assuming that there are enough processors.

The basic idea of our wavefronthing scheme is to examine only “promising” legal wavefronts and choose the one resulting in the minimum execution steps. There are two general classes of “promising” wavefronts. One is from the boundary lines of the iteration space and the other is from the dependence vectors. In the generalized cycle shrinking method [4] the latter is based on the difference of every pair of dependence vectors. Unfortunately, for loops with linear dependencies, it is too costly to find all the dependence vectors and the differences between them. To solve
Given: A doubly nested loop $L$.
1. Find DVCH($L$).
2. For each boundary segment of DVCH($L$), find its normal vectors and put them into a set $W$ if they are legal.
3. For each boundary segment of the iteration space, find its normal vectors and put them into $W$ if they are legal.
4. For each vector in $W$, compute its execution steps.
5. Return the vector in $W$ with the minimum execution steps.

**Figure 3. The algorithm find_wavefront.**

this problem we can use their profile, the DVCH, instead. The boundary lines of the DVCH can be viewed in a way as the differences between some representative dependence vectors and are thus promising candidates for finding the wavefront. Based on the idea, Algorithm find_wavefront in Fig. 3 gives a simple procedure to find the best wavefront direction.

### 5. Comparison with previous approaches

In this section, our scheme is compared with two representative approaches: dependence uniformization [8] and parallel region [11]. In the uniformization approach, linear dependence vectors are replaced by a set of basic dependence vectors (BDVs) during parallelization. The BDV set is obtained by first computing the dependence vector slope, $d_2/d_1$, for each dependence vector $(d_1, d_2)$. Then, the dependence region $DR(L)$ of the loop is used to obtain the range of the dependence vector slopes, $[s_{min}, s_{max}]$, for all dependence vectors in $L$. The BDV set is taken to be either $\{(0, 1), (1, [s_{min}])\}$ or $\{(-1, 1, [s_{max}])\}$. Next, the loop $L$ is parallelized with wavefronting according to BDVs. It is shown that arranging the iteration executions according to the BDV set is sufficient to preserve the dependencies in the original loop.

The parallel region approach, on the other hand, divides the iterations in the given loop $L$ into four regions. The non-DR region contains those iterations $(g_1(x, y), g_2(x, y))$ whose $(x, y) \not\in DR(L)$. The remaining iterations are divided into three regions: Region area$_1$ contains iterations which are a destination of a data dependence and which execute before their corresponding source iteration. The iterations in area$_1$ can be executed first in parallel. Region area$_2$ contains iterations which are a destination of a data dependence and which execute after their corresponding source iteration. Furthermore, the source iteration must be in area$_1$. The iterations in area$_2$ can be executed next in parallel. The remaining iterations are in area$_3$ and are executed sequentially. The iterations in non-DR can be computed any time.

Now we are ready to compare our proposed scheme with the above two approaches in terms of the theoretical speedups. The comparison is based on the examples taken from [8, 11]. To simplify the comparison, it is assumed that the inner and outer loops each iterate 1000 times. The theoretical speedup is obtained assuming there is an infinite number of processors:

$$\text{Theoretical speedup} = \frac{1000 \times 1000}{\text{execution steps}}.$$  

The results are shown in Table 1. Derivations of these results can be found in [6].

According to Table 1, we can see that the proposed DVCH approach extracts more parallelism than does the uniformization scheme for the tested cases. One reason is because the uniformization scheme mainly manipulates with the dependence slopes instead of with the dependence vectors. As a result some useful information may be lost. On the other hand, the DVCH profiles the dependence vectors in the loop and thus provides more accurate information. In fact, with the DVCH we can obtain an even tighter range of dependence slopes than with uniformization [6]. Another reason is because the uniformization method considers only two possible wavefront directions: $(|s_{min}|-1, -1)$ and $(|s_{max}|+1, -1)$. The DVCH approach, on the contrary, considers all plausible candidates and thus can choose a better wavefront.

Comparing the DVCH approach with parallel region, we can see that each has its own advantages. In the parallel

### Table 1. Testing loops for comparing the theoretical speedups.

|-----|------|-------------|--------------|-------------------|
| C1  | DO I=1,1000  
    DO J=1,1000  
    A(I+J,3+I+3)=...  
    ...=A(I+J+1,1+2J+4) | 1,000,000  
    500,000 | 1,000,000  
    500,000 | 1,000,000  
    500,000 |
| C2  | DO I=1,1000  
    DO J=1,1000  
    A(2I+3,3+I+3)=...  
    ...=A(2I+J+1,1+J+3) | 1,000,000  
    500,000 | 500,000  
    1,000,000 | 1,000,000 |
| C3  | DO I=1,1000  
    DO J=1,1000  
    A(2I+3,3+I+1)=...  
    ...=A(I+J+3,2I+1) | 1,000,000  
    500,000 | 500,000  
    1,000,000 | 1,000,000 |
| C4  | DO I=1,1000  
    DO J=1,1000  
    A(2I,2J)=...  
    ...=A(I+J+10,2I+1) | 1  
    1  
    1  
    1  | 1,000,000  
    1,000,000  
    1,000,000  
    1,000,000 | 1,000,000  
    1,000,000  
    1,000,000  
    1,000,000 |
| C5  | DO I=1,1000  
    DO J=1,1000  
    A(I,J)=...  
    ...=A(I,J) | 1,000,000  
    500,000 | 500,000  
    1,000,000 | 1,000,000 |
Table 2. An extreme case showing the difference between parallel region and the proposed scheme.

<table>
<thead>
<tr>
<th>Loop</th>
<th>Parallel region</th>
<th>Speedup</th>
<th>DVCH</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \text{DO} \text{ I}=1,1000 ] [ \text{DO} \text{ J}=1,1000 ] [ A(I+J+3,2<em>J+1)=... ] [ ...=A(2</em>J+3,I+1) ]</td>
<td>area_1 = \emptyset \quad area_2 = \emptyset \quad area_3 = 500,000 \quad non-DR = 500,000</td>
<td>2</td>
<td>{ (1, 0), (1, -499), } { (2, -499), (500, -250), } { (500, 249), (499, 249) }</td>
<td>1000</td>
</tr>
</tbody>
</table>

region approach, if the amount of anti-dependencies in the loop is large, then area_1 will contain a lot of iterations. These iterations can be executed in parallel and thus the effect of parallelization is very good. However, if regions area_1 and area_2 contain only a few iterations, then the sequential executions in area_3 will dominate the total execution time. Loop C4 in Table 1 is a typical example of such loops.

The DVCH approach is essentially a uniformization scheme. One and only one regular wavefront is used to parallelize the given loop. It does not exploit the irregularity inherent in linear dependencies. However, such a scheme has the advantage that it is not affected by the amount of anti-dependencies. Consider the extreme case shown in Table 2. The DVCH approach can still extract a reasonable amount of parallelism, while the parallel region method falls far behind.

Another problem with the parallel region method is that it considers only linear dependencies. According to our experiences with EISPACK [6], uniform dependencies not only intermix with linear dependencies often, but in fact are the major limiting factor for parallelization. Thus, even if we can use parallel regions to parallelize the iterations, the uniform dependencies in the loop might force them to become partially parallelized. On the other hand, the DVCH approach does not distinguish between uniform and linear dependencies. It considers both when doing parallelization. Thus, the generated code will be more efficient and much simpler.

6. Concluding remarks

Linear dependencies usually generate a large number of dependence vectors. In real programs they also tend to intermix with uniform dependencies. These factors make loops with such dependencies difficult to handle. In this paper, we propose to use the dependence vector space as the unifying framework to manipulate the dependencies and to use the dependence vector convex hull to profile them. We have described a method to find the DVCH for a given doubly nested loop. Using the DVCH, we have also extended previous wavefronting techniques to parallelize loops with linear dependencies. Comparisons with other approaches show that our scheme performs well. Since we consider only doubly nested loops here, a detailed study and more extensive experiments for loops with multiple nesting levels are needed and will be our future work.

References