Enhancing Parallelism by Removing Cyclic Data Dependencies

Fubo Zhang and Erik H. D'Hollander

University of Ghent, Department of Electrical Engineering
B-9000 Ghent, Belgium

Abstract. The parallel execution of loop iterations often is inhibited by recurrence relations on scalar variables. Examples are the use of induction variables and recursive functions. Due to the cyclic dependence between the iterations, these loops have to be executed sequentially. A method is presented to convert a family of coupled linear recurrence relations into explicit functions of a loop index. When the cyclic dependency is the only factor preventing a parallel execution, the conversion effectively removes the dependency and allows the loop to be executed in parallel. The technique is based on constructing and solving a set of coupled linear difference equations at compile-time. The method is general for an arbitrary number of coupled scalar variables and can be implemented by a straight-forward algorithm. Results show that the parallelism of several sequential EISPACK do-loops is significantly enhanced by the converting them into do-all loops.

Keywords: formal program development methodologies, language constructs, implementation issues, induction variable removal, linear recurrence relations.

1 Introduction

A DO loop is excitable in parallel when there are no loop carried data dependencies. There exist many techniques to handle data dependencies of arrays and to parallelize DO loops[2, 4, 6, 8, 11, 12, 14]. In the case of scalar variables, data dependencies can be removed by renaming scalar variables or expanding scalar variables into array references[5]. However the cyclic data dependencies arising from induction variables or recurrences generally cannot be removed by renaming scalar variables or scalar variables extensions. A *basic induction variable* is a variable whose value is systematically incremented or decremented by a constant value in a loop and a variable defined by combination of basic induction variables or other induction variables is an induction variable[1, 7, 13]. For example, if a loop contains a basic induction variable such as $i = i + 1$, the loop cannot be parallelized due to the cyclic data dependence. Scalar variable extension means removing the scalar i by converting it into an array $I[k]$ of loop index k.

Traditional compiler optimizations are able to eliminate simple induction variables by expressing them as a linear function of the loop index[1, 7]. Those induction variables are defined as *basic linear induction variables*. For instance,

$i = i + 1$ is expressed as $i = i_0 + k$. Where i_0 is the initial value of i and k is the loop index.

M.Wolfe[13] improved the analysis of induction variables to include cases such as

- wrap-around variables,
- flip-flop variables,
- families of periodic variables,
- non-linear induction variables (polynomial and geometric), and
- monotonically increasing and decreasing variables.

Wolfe eliminates some non-linear induction variables by expressing them as polynomial and geometric functions of the loop index. For example, when the bases of a geometric induction variable are $g_1, g_2, \ldots, g_n$, the induction variable can be represented as the function:

$$\sum_{i=0}^{m} s_i h^i + \sum_{j=1}^{n} s_{j+m} g_j^h$$

There are $m + n + 1$ unknowns s_k which are found by matrix inversion when the sequence of initial $m + n + 1$ states of the induction variable is known. Here h is the loop counter starting from zero and m is the order of the polynomial. Normally, a geometric induction variable I is produced by an assignment of the form $I = b * I + c$. Here the geometric base, b, is immediately apparent from the statement. The following program, however, gives an example where the geometric bases are not immediately visible.

Example 1.

```
u=v=0
do i =1, N
   u= 2*u+v+2
   v= 2*u-3*v-3
   a(u,v) = a(u,v)+c(v, u)
enddo
```

The reason is that the two induction variables, u and v form two coupled difference equations. In this case the geometric bases of u and v are determinated by the homogeneous solutions of the difference equations [10].

In this paper a general method is proposed to solve the problem of n simultaneous induction variables which form n a set of coupled linear difference equations. An algorithm is proposed to find the geometric bases of the induction variables and to express them as a polynomial and geometric functions of the loop index. In this way the cyclic dependency is removed and the loop is reshaped as a *do-all* loop. The conversion technique is based on solving a set of coupled difference equations and is described in section 2.

The method is general for an arbitrary number of coupled scalar variables and can be implemented by a straight-forward algorithm. This is shown in section 3, where also the implementation steps are explained using a specific example.

In section 4 it is shown that the removal of the cyclic dependencies in a set of EISPACK benchmark routines is able to increase the number of do-all loops.

2 Removal of cyclic induction variables

A cyclic data dependence is generated when a scalar variable uses the value of a scalar variable in the previous iteration. We address this scalar variable as a induction variable. There are several methods to find these induction variables [3, 13]. A *conditional* induction variable is an induction variable which appears in branch statements i.e. if-statement, and conditionally incremented or decremented. This induction variable cannot generally be expressed as a function of the loop index. Hence, we exclude this case from our discussion.

Assume, a loop contains a set of n *coupled induction variables* as in the following example.

$$\begin{array}{l} do\ K = 1,\ N \\ \quad D_1 = h_1(U_1) \\ \quad T = AT + B \\ \quad D_2 = h_2(U_2) \\ enddo \end{array} \tag{1}$$

Here $A_{n\times n}$ and B_n represent constant arrays and $T = [t_1, \cdots, t_n]^T$ is the set of recursively defined *induction variables*. Furthermore D_1, D_2, U_1 and U_2 denote the set of *defined* and the set of *used* variables in the loop. If D_k and U_k $(1 \leq k \leq 2)$ do not overlap, the parallel execution is only prevented by the cyclic dependence in the ud-chain of T.

Consider the induction variables T in the loop (1). Let T_k $(k \geq 0)$ be the state of T after iteration k and let T_0 be the initial state. Clearly, the induction variables only depend on the state of the previous iteration by the set of coupled difference equations:

$$T_{k+1} = AT_k + B \tag{2}$$

When substituting T_k by T_{k-1} and T_{k-1} by T_{k-2}, ..., T_1 by T_0, one obtains

$$T_{k+1} = A^{k+1}T_0 + \sum_{i=0}^{k} A^i B \tag{3}$$

Equation (2) represent n-first order difference equations with the constant coefficients. In general, simultaneous difference equations are solved by manually eliminating $n-1$ variables, solving the resulting $n-$th order difference equation and using back substitution [10].

However, for the automatic conversion at compile time, a direct method is needed. Therefore define the *difference* operator $\Delta y_k = y_{k+1} - y_k$ and the *shifting* operator $E = 1 + \Delta$ or $Ey_k = y_{k+1}$. The following property of operators Δ and E is useful [10].

Lemma 1. Let $P_k = a_n k^n + a_{n-1} k^{n-1} + \ldots + a_0$ be a polynomial function. Then $\Delta^n P_k = a_0 n!$ and $\Delta^{n+m} P_k = 0,\ m \geq 1$.

By using shifting operator E, the equation (2) is modified into $ET_k = AT_k + B$. Hence

$$(EI - A)T_k = B$$

Here I is the identity matrix. The solutions of $t_k^{(i)}$ are

$$t_k^{(i)} = \frac{|H_i|}{|EI - A|}$$

where H_i is derived from the matrix $(EI - A)$ by replacing column i with the array B. When the shifting operator E is applied to a constant, the result is the constant itself. Therefore any shifting function $f(E)$ applied to a constant c satisfies the following identity $f(E)c = f(1)c$. Because B and H_i are constants, the operator E may be replaced by unity in H_i. Therefore $|H_i|$ is a constant too. Denote $h_i = |H_i|$. One has the following difference equation:

$$|EI - A| t_k^{(i)} = h_i, (1 \leq i \leq n)$$

The homogeneous solutions are the roots of the function $|EI - A| = 0$, i.e the eigenvalues of the matrix A. Let $\lambda_i, (1 \leq i \leq n)$ be the n eigenvalues of the matrix A. Hence each $t^{(i)}$ is independently represented as one $n-$th order difference equation,

$$\prod_{j=1}^{n} (E - \lambda_j) t_k^{(i)} = h_i, (1 \leq i \leq n) \tag{4}$$

Therefore, the complete solution of (4) is given as follows.

$$t_i = c_{i1}\lambda_1^k + c_{i2}\lambda_2^k + \ldots + c_{in}\lambda_n^k + p_i(k) \tag{5}$$

Where the $c_{ij}, 1 \leq i, j \leq n$ are n sets of arbitrary constants; $p_i(k)$ is a particular solution of equation (4).

Homogeneous solutions of (4) depend on the roots $\lambda_1, \ldots, \lambda_n$. The following cases are possible.

1. The roots are all real and distinct. One has the homogeneous solution

$$t_k^{(i)} = c_{i1}\lambda_1^k + c_{i2}\lambda_2^k + \ldots + c_{in}\lambda_n^k$$

2. Some of the roots are complex numbers. Suppose $\alpha + \beta i$ and $\alpha - \beta i$ are a couple of roots. Then the homogeneous solution is

$$\rho^k (c_1 \cos k\theta + c_2 \sin k\theta)$$

3. Some of the roots are equal. Suppose $\lambda_1 = \lambda_2 = \ldots, = \lambda_m$. Then the homogeneous solution is

$$(c_1 + c_2 k + \ldots + c_m k^{m-1})\lambda_1^k$$

In addition, since the right side of (4) is a constant, a particular solution can be easily found.

2.1 Finding a particular solution

Because the right side of (4) is a constant, a particular solution is a polynomial of the form $p_i(k) = \sum_{j=0}^{n} a_j k^j$.

Let

$$S_m = \prod_{j=m}^{n} (1 - \lambda_j), m = 1, \ldots, n$$

Three cases are considered, depending on the number of unity roots. If equation (4) has no unity roots, then a particular solution is h_i/S_1. If all roots are equal to 1, then by lemma (1) a particular solution is $h_i k^n/n!$. If the number of unity roots is between 1 and n, the following theorem applies.

Theorem 2. If $\lambda_1 = \lambda_2 = \ldots, = \lambda_m = 1$ and $\lambda_j \neq 1, m < j \leq n$, then $p(k) = h_i k^m/(S_{m+1} m!)$ is a particular solution of (4).

Proof.
By replacing $t_k^{(i)}$ with $p(k)$ in the equation (4), the left side of equation becomes

$$\prod_{j=m+1}^{n} (E - \lambda_j)(E-1)^m k^m h_i/(m! S_{m+1})$$

By lemma (1), $(E-1)^m k^m = \Delta^m k^m = m!$. Therefore,

$$\prod_{j=m+1}^{n} (E - \lambda_j)(E-1)^m k^m h_i/(m! S_{m+1}) = \prod_{j=m+1}^{n} (1 - \lambda_j) m! h_i/(m! S_{m+1}) = h_i$$

This proves that $p'(k)$ is a particular solution of (4).
□

2.2 Finding the complete solution

The equation (5) expresses the general solution. In order to completely establish the solution, the constants $c_{ij}, 1 \leq j \leq n$ should be determined for the initial state $t_0^{(i)}$. Because there are n constants, the states of $t_1^{(i)}, t_2^{(i)}, \ldots, t_n^{(i)}$ are required. These can be calculated by the equation (2) from the initial state T_0.

The ith set of constants $C_i = (c_{i1}, \ldots, c_{in})^T$ are determined from the following equations.

$$LC_i = P \tag{6}$$

Where

$$L = \begin{pmatrix} \lambda_1 & \ldots & \lambda_n \\ \lambda_1^2 & \ldots & \lambda_n^2 \\ \ldots & & \\ \lambda_1^n & \ldots & \lambda_n^n \end{pmatrix} \text{ and } P = \begin{pmatrix} t_1^{(i)} - p_i(1) \\ t_2^{(i)} - p_i(2) \\ \ldots \\ t_n^{(i)} - p_i(n) \end{pmatrix}$$

Since $\lambda_j, t_j^{(i)}$ and $p_i(j)$ are constants, this linear system has the solution $(c'_{i1}, c'_{i2}, \ldots, c'_{in})$.

Now the complete solution is

$$t_k^{(i)} = c'_{i1}\lambda_1^k + c'_{i2}\lambda_2^k + \ldots + c'_{in}\lambda_n^k + p_i(k) \tag{7}$$

Where $p_i(k)$ is a polynomial by theorem 2.

3 Algorithm and Application

The method to eliminate induction variables is applicable in a program transformer by applying the following algorithm.

Algorithm 1 (Elimination of induction variables)
Let $T_{k+1} = AT_k + B$ n be coupled difference equations.
input— A, B and n. A and B are constant arrays.
output the complete solutions $t_k^{(i)} = c_{i1}\lambda_1^k + c_{i2}\lambda_2^k + \ldots + c_{in}\lambda_n^k + p_i(k)$.

Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A. Then

$\prod_{j=1}^{n} (E - \lambda_j) t_k^{(i)} = c'_i, (1 \leq i \leq n)$

For $i = 1$ to n do

1) By theorem 2, a particular solution of $t_k^{(i)}$ is found.
2) The homogeneous solution is found as follows:
Let L be a matrix with kth row equal to $(\lambda_1^k, \ldots, \lambda_n^k)$ $1 \leq k \leq n$.
a) If λ_j and λ_{j+1} are complex numbers, then
λ_j^k and λ_{j+1}^k of L are replaced with $\rho^k \cos k\theta$ and $\rho^k \sin k\theta$.
b) If $\lambda_j = \lambda_{j+1} = \ldots = \lambda_{j+m-1}$, then $\lambda_j^k, \ldots, \lambda_{j+m-1}^k$ in L are replaced with $\lambda_j^k, k\lambda_j^k \ldots, k^{m-1}\lambda_j^k$.
3) Let the vector $P = [t_1^{(i)} - p_i(1), \ldots, t_n^{(i)} - p_i(n)]^T$.
4) Solve the set of linear equations $LC_i = P$ for C_i.
Endfor

To illustrate the different steps of the algorithm, let's recap the example (1). The induction variables $T = [u\ v]^T$ satisfy the difference equations expressed in the program. Using shifting operator E, the set of equations becomes:

$$\begin{aligned} (E-2)u_k - v_k &= 2 \\ -4u_k + (E+1)v_k &= 1 \end{aligned}$$

Therefore

$$|EI - A| = \begin{vmatrix} E-2 & -1 \\ -4 & E+1 \end{vmatrix} = (E+2)(E-3)$$

yielding the eigenvalues -2 and 3.

With

$$|H_1| = \begin{vmatrix} 2 & -1 \\ 1 & E+1 \end{vmatrix} = 5 \ and \ |H_2| = \begin{vmatrix} E-2 & 2 \\ -4 & 1 \end{vmatrix} = 5$$

the decoupled 2-nd order difference equations become

$$\begin{aligned}(E+2)(E-3)u_k &= 5\\ (E+2)(E-3)v_k &= 7\end{aligned}$$

By theorem 2, particular solutions $p_1(k) = 5/((1+2)(1-3)) = -5/6$ and $p_2(k) = 7/((1+2)(1-3)) = -7/6$ are found.

Then the general solutions are:

$$\begin{aligned}u_k &= c_1(-2)^k + c_2 3^k - 5/6\\ v_k &= c_1'(-2)^k + c_2' 3^k - 7/6\end{aligned}$$

In order to determine the coefficients c, note that $u_1 = 2, u_2 = 7$ and $v_1 = 1, v_2 = 8$. We have

$$\begin{array}{l}-2c_1 + 3c_2 = 2 + 5/6\\ 4c_1 + 9c_2 \;\; = 7 + 5/6\end{array} \; and \; \begin{array}{l}-2c_1' + 3c_2' = 1 + 7/6\\ 4c_1' + 9c_2' \;\; = 8 + 7/6\end{array}$$

yielding $c_1 = -1/15, c_2 = 9/10$ and $c_1' = 4/15, c_2' = 9/10$.

As a result of eliminating the induction variables, the cyclic data dependencies are removed. Therefore, the loop is now parallelized as a doall-loop.

```
u=v=0
doall i =1, N
   u1 = (-2)**i
   v1 = 3**i
   u= -u1/15+9*v1/10 -5/6
   v= 4*u1/15 +9*v1/10-7/6
   a(u,v) = a(u,v)+c(v, u)
enddo
```

Next, we discuss the algorithm for some types of induction variables.

Basic Linear Induction Variables are a simple case of non-linear induction variables where the geometric bases are either unity or zero and the number of unity bases is equal or less than 2.

A Non-Linear Induction Variable as defined in [13] can be presented as a polynomial and a geometric function by the algorithm (1). Especially, when the geometric bases are either 1 or 0, the function is a polynomial function. Therefore we have the following theorem.

Theorem 3. The solution of the induction variables in the equation 7 is polynomial if and only if the eigenvalues of the matrix A are either unity or zero.

Proof:
When the solution of induction variables is polynomial, the homogeneous part of equation 7 is a constant. Therefore λ_i $(1 \leq i \leq n)$ of equation 7 are either 1 or 0. Consequently, the eigenvalues of the matrix A are either unity or zero.

Inversely, when the eigenvalues of the matrix A are either 1 or 0, the homogeneous part of equation 7 is a constant. By the theorem 2, the particular solution $p_i(k)$ of equation 7 is polynomial. Hence the solution of the induction equations is polynomial.
□

A Wrap-Around Variable is a variable t which value is also used in U_1 in the equation (1). The algorithm can express it as a function $f(k)$ of the loop index k. In all but the first iteration, in U_1 t's value is equal to $f(k-1)$; in the first iteration t's value is equal to its initial value t_0. Hence a loop-header ϕ-function is added in the front of U_1, $t' = \phi(t_0, f(k-1))$ which is equal to t_0 when $k = 1$, otherwise is $f(k-1)$. Therefore the appearances of t in U_1 are replaced by t'. Hence, the following example from [13]

```
im1=n
do k=1, n
  A(k)=A(im1)+ ...
  im1=k
enddo
```

is converted into

```
im1=n
do k=1, n
  im1'= φ(n,k-1)
  A(k)=A(im1')+ ...
  im1=k
enddo
```

In this way, the cyclic data dependence of $im1$ is eliminated.

Flip-Flop and Periodic Variables are often used to switch the values of two variables [13]:

```
j=1
jold=2
do k=1, n
  ... relaxation code ...
  jtemp = jold
  jold=j
  j=jtemp
enddo
```

The difference equation of j and $jold$ is

$$\begin{aligned} jold_k &= j_{k-1} \\ j \quad &= jold_{k-1} \end{aligned}$$

Which is rewritten by using operator E as below.

$$\begin{aligned} Ejold - j &= 0 \\ -jold + Ej &= 0 \end{aligned}$$

The homogeneous solutions are -1 and 1 respectively, and particular solution is 0. We have

$$\begin{aligned} jold_k &= c_1 1(-1)^k + c_2 1 \\ j_k &= c_2 1(-1)^k + c_2 2 \end{aligned}$$

These coefficients are found by solving the linear equations with the initial values $\{jold_1 = 1, j_1 = 2, jold_2 = 2, j_2 = 1\}$.

$$\begin{aligned} jold_k &= \tfrac{1}{2}(-1)^k + \tfrac{3}{2} \\ j_k &= \tfrac{1}{2}(-1)^{k+1} + \tfrac{3}{2} \end{aligned}$$

Therefore the cyclic data dependencies are removed, and we obtain the following program

```
j=1
jold=2
do k=1, n
  ... relaxation code ...
  jtemp = (1/2)*(-1)**(k+1)+3/2
  jold=(1/2)*(-1)**k+3/2
  j=(1/2)*(-1)**(k+1)+3/2
enddo
```

4 Results

In the previous sections a technique has been developed to remove the linear cyclic dependencies. This type of dependencies arises regularly in common programs. Consider the following loop taken from the EISPACK routine `bqr.f` (program 2.a).

Here a cyclic data dependence is created by variable kj. So the do-loop cannot be executed in parallel. After removing the cyclic dependence (program 2.b), the parallelism of loop is enhanced.

Example 2.

```
      kj =m4 + m2 * m1 + 1 |    kj01 =m4 + m2 * m1 + 1 ! initial kj
      do 200 k = 2, m1     |    doall 200 k = 2, m1    ! doall
         kj = kj + 1       |      kj = kj01 + k - 1
         km = k + m2       |      km = k + m2
         rv(kj) = rv(km)   |      rv(kj) = rv(km)
 200  continue             | 200 continue
                           |
    (a) before removing    |       (b) after removing
```

```
cyclic dependence   |          cyclic dependence
```

In order to show the importance of cycle dependence removal, the last column of table (1) gives the extra do-all loops detected by the methods presented in this paper.

From the table can be seen that from the original 31 parallel loops 15 extra loops have been parallelized, an increase of 42%.

Code	do-loops	doalls before	doalls after	doalls increase
bandv	22	4	8	4
bqr	21	6	8	2
ratqr	11	3	4	1
cinvit	18	6	7	1
trbak	4	1	3	2
tred3	10	4	8	4
tsturm	22	7	8	1
total	108	31	46	15

Table 1. The improvement of parallelism in a set of EISPACK routines after removing the cyclic data dependencies

5 Conclusions

A method to eliminate a class of cyclic dependencies arising from linear recurrence relations has been developed. As a result cyclic data dependencies caused by induction variables are removed. If there are no other loop carried dependencies, the loop can be transformed into a doall-loop.

The presented technique can be implemented effectively in parallelizing compilers and has been used successfully to eliminate common recurrence constructs in a number of EISPACK routines.

References

1. Alfred V. Aho and Jeffrey D. Vllman. *Compilers: Principles, Techniques, and tools.* Addison-Wesley, Reading, MA, 1986.
2. Allen, J.R., and Kennedy, K. Automatic Translation of FORTRAN Programs to Vector Form. ACM *Transaction on programming Language & Systems.* Vol. 9, No.4(October), pp.491-542, 1984.

3. Z.Ammarguellat and W.L. Harrison. Automatic recognition of induction variables and recurrence relations by abstract interperation. *ACM SIGPLAN'90, SIGPLAN Notices*. Vol. 25, No.6 (June), pp. 283-295, 1990.
4. Banerjee, U., 1988 Dependence Analysis for Supercomputing. *The Kluwer international series in engineering and computer science. Parallel processing and fifth generation computing*. ISBN 0-89838-289-0. Kluwer Academic Publishers, 1988.
5. Ron Cytron and Jeanne Ferrante. What's in a name ? -or- The Value of Renaming for Parallelism Detection and Storage Allocation. In *International Conference on Parallel Processing*, 1987. pp. 19-27.
6. D'Hollander Erik H. Partitioning and Labeling of Loops by Unimodular Transformations *IEEE Transactions on Parallel and Distributed Systems*, 3, 7, 465-476, 1992.
7. Michael P.Gerlek. Detecting Induction Variables Using SSA Form. Dept. of CSE, Oregon Graduate Institute OGI-CES technical report 93-014, 1993.
8. Kuck, D.J., Kuhn, R.H., Padua, D.A., Leasure, B.R., and Wolfe, M.J. Dependence Graphs and Compiler Optimizations. In *Conference Proceedings– The 8th Annual ACM Symposium on Principles of Programming Languages* (Williamsburgh, Virginia, January 26-28), ACM Press, pp.207-218, 1981.
9. Walter G. Kelley and Allan C.Peterson *Difference Equations* Academic Press, Inc. 1991.
10. Ronald E. Mickens *Difference Equations* Second Edition, Van Nostrand Reinhold Library of Congress Card Number 90-34385, 1990.
11. Wolfe, M.J. Optimizing Supercompilers for Supercomputers. *PhD. Thesis*, the Graduate College of the University of Illinois at Urbana-Champaign, 1982.
12. Wolfe M. TINY. A Loop Restructuring Research Tool. Oregon Grad. Inst. of Sc. and Tech., Dept. Comp. Sc.& Eng., Beaverton. *Techical Report*, 19-14-21, 1991.
13. Wolfe M. Beyond Induction Variables. *ACM SIGPLAN'92, SIGPLAN Notices*. Vol. 27, No.7 (July), pp. 162-174, 1992.
14. Wu Youfeng, Lewis Ted G., Parallelizing While Loops, *Proceedings of the Intl. Conf. on Parallel Processing '90, II - Software*, August 13-17, pp. 1-8, 1990.

This article was processed using the LaTeX macro package with LLNCS style