Information Technology Journal 7 (3): 409-419, 2008
ISSN 1812-5638

Weighting Characteristic Test for Data Dependence Testing on Loops

Chen-Feng Wu
Department of Information Management, Yu-Da College of Business Miaoli, Taiwan

Abstract: Due to the importance of data dependence test for exploiting parallelisms in loop, an approach that integrates existing test methods and makes good use of their advantages is proposed in this study. The proposed method, called weighting characteristic test, will be suitable for all cases due to considering their characteristics of input data. By computing the weight of characteristics for loop input data, the proposed approach chooses the appropriate test and applies it to detect data dependences on loops. The weighting characteristic test gives exact solution in most of input cases and less overhead than other integrated methods from our experimental results. Thus, the proposed method is more applicable than existed data dependence tests for exploiting loop parallelism indeed.

Key words: Parallelizing compiling, data dependence, weighting characteristic test

INTRODUCTION

For several decades, the commercial supercomputers and mini-supercomputers have been a major class of highly parallel and widely applicable machines. The parallel compilers play the most important role of exploiting the parallelization in programs, especially, for the array usage in loops. Because the data dependence testing is one of the cruses of parallel compiler, developing an effective data dependence test on loops is the first step for parallel compiler.

There have been extensive studies of decision methods for array data dependences (Zima and Chapman, 1990; Zi *et al*., 1990; Kong *et al*., 1991; Banerjee, 1998; Wolfe and Tseng, 1992; Pugh, 1992; Goff *et al*., 1991; Maydan *et al*., 1991; Shih *et al*., 1994; Ahamed *et al*., 2001; Mineo *et al*., 2003, 2004; Chang *et al*., 2004; Van Engelen *et al*., 2004; Bertrand *et al*., 2005; Baude *et al*., 2007; Damevski *et al*., 2007). Data dependence testing problems are equivalent to deciding if two references to the same array within nested loops may reference to the same element of array. In general, to solve such a problem would be considered as integer programming and the time complexity of the best integer programming algorithm is $O(n^{O(n)})$, where n is the number of loop indices. Clearly, the cost of the algorithms is too expensive to be applied, so more feasible, faster and less exact algorithms might be desirable in some problems.

In order to investigate an efficient and precise technique, we can consider how to solve well an integer solution to a set of linear equalities and inequalities since data dependence testing problem can be thought as an integer programming resolution problem. The problem is reduced to be an m-dimensional referenced array with linear indices and to decide whether a system of m linear equations, each of the form, as shown in Eq. 1, has a simultaneous integer solution which satisfies of the form, as shown in Eq. 2, where n is the depths of loops and each I_k is either a loop index variable or some other code variable. If I_k is a loop index variable, it is bounded by M_k, the lower bound of indices and N_k, the upper bound of indices. Usually, M_k and N_k can be decided statically, however, they still are unknown in some cases.

$$a_1I_1 + a_2I_2 + \ldots + a_nI_n = a_0 \quad (1)$$

$$M_k \leq I_k \leq N_k,\ 1 \leq k \leq n, \quad (2)$$

For data dependence testing algorithms, a new approach, characteristic weighting test is proposed in this research. Because existing algorithms have their different flavors of input cases, they are integrated by extracting characteristics of input cases and choose an appropriate one with taking efficiency and precision into consideration. In our approach, a library of testing algorithms is constructed and some attributes are selected by testing algorithms to assign weight. A weighting characteristic system, the kernel of our approach, is constructed in this study. The kernel assigns a proper weight to each attribute and selects an appropriate algorithm by computing the total weight of attributes according to loop behavior.

In past decades, a great deal of researches (Wolfe and Tseng, 1992; Pugh, 1992) had worked on the trade-off between the precision on data-dependence testing and its overhead. The key points of our approach are concentrated on both efficiency and precision. There are some researches (Goff *et al*., 1991; Maydan *et al*.,

1991) to prove that once very simple subscripts are filtered away and tested inexpensively, the expensive test is required for only a few remaining subscripts. To detect simple data dependences by simple tests and complex data dependence by expensive tests, a weighting characteristic approach is proposed and chooses a simple algorithm, such as the Banerjee test or the GCD test, for data dependence testing, when simple subscripts appear and left complex ones to the algorithms of expensive cost, such as the Power test.

BACKGROUND

To exploit parallelism in program, parallelizing compilers need a precise dependence test to allow the most freedom in applying restructuring transformation (Wolfe and Tseng, 1992) which is necessary for iterations of loops in programs to distribute them on different processors and concurrently execute to achieve speedup of whole execution time.

Definition of data dependence testing: The process of computing all the data dependences in a program is called data dependence analysis. If statement S_1 and statement S_2 access the same location of array in loops, we say that data dependence exists between S_1 and S_2. In general, three types of data dependences are often mentioned by Hwang (1986):

Flow dependence: As shown in Fig. 1, the location of array A in statement S_1 is written by the sum of x and y, then read by statement S_2 later.

Anti-dependence: As shown in Fig. 2, Statement S_1 reads array A which will be written by the sum of w and z later.

Output dependence: As shown in Fig. 3, the sum of x and y is written to array A which will be written by the sum of w and z later.

The methods of data dependence testing are used to determine whether there exist dependences between two subscripted references to the same array in a nested loop. Suppose we wish to test where there exists a dependence from S_1 to S_2 in loop model of Fig. 4, loop nest of n levels, represented by n integer indices $i_1, i_2, \ldots i_n$.

Let α and β be vectors of n integer indices $i_1, i_2, \ldots, i_n$ written the range of the upper and lower bounds of the n loops and f_i, g_i be the mapping function of vectors α and β. There is a dependence from S_1 to S_2 if and only if there exist α and β, such that α is lexicographically less than or equal to β and the following system of dependence Eq. 3 is satisfied:

$$f_i(\alpha) = g_i(\beta)\ \forall i,\ 1 \le i \le m \qquad (3)$$

```
      Do I = 1,N
          …
S1 :      A[I] = x +y;
S2 :      z = A[I];
          …
      Enddo
```

Fig. 1: Example of flow dependence

```
      Do I = 1, N
          …
S1 :      y = A[I] + x;
S2 :      A[I] = w + z;
          …
      Enddo
```

Fig. 2: Example of anti-dependence

```
      Do I = 1, N
          …
S1 :      A[I] = x + y;
S2 :      A[I] = w + z;
          …
      Enddo
```

Fig. 3: Example of output dependence

```
Do i1 = L1, U1
    Do i2 = L2, U2
        …
        Do in = Ln, Un
S1 :        A(f1(i1,i2,…,in),…, fm(i1,i2,…,in)) = …
S2 :        … = A(g1(i1,i2,…,in),…, gm(i1,i2,…,in))
        Enddo
        …
    Enddo
Enddo
```

Fig. 4: A model of n levels nested loop

Otherwise the two references are independent. The dependence equation, as shown in Eq. 1, is linear expressions of the loop index variables. Thus, dependence testing is equivalent to the problem of solving simultaneous equations, which is a NP-complete problem. Exact tests are dependence tests that will detect if only if they exist.

Data dependence testing methods: Data dependence tests are classified into three classes, single dimension, multi-dimension and integrated methods in this research.

Single dimension

Banerjee test: The Banerjee test (Zima and Chapman, 1990; Banerjee *et al*., 1979) considers a limit only if it is statically determinable to have a constant value.

GCD test: The GCD test (Zima and Chapman, 1990; Banerjee *et al*., 1979) is based on the theorem of greatest common divisor, where it determines whether the gcd of coefficients, (a_1, a_2, …, a_n), of Eq. 1 is a divisor of a_0. The GCD test ignores limits entirely and determines whether the equation has a solution for any integer values at all of the variables.

I test: The I test (Kong *et al*., 1991) is a refinement of a combination of the GCD and Banerjee tests. It checks for the existence of integer solution and takes limits into consideration. Furthermore, it can produce a definite solution when the GCD and Banerjee tests produce only tentative ones.

Multi-dimension

Extended GCD test: The extended GCD test (Zima and Chapman, 1990), the ideas behind Euclid's GCD algorithm being extended to find a general integer solution for a set of linear equations with integer coefficients, was investigated by Knuth (1981). The Extended GCD test, reviewed and used in the Power test for data dependence testing by Wolfe and Tseng (1992), was described as a matrix form of the algorithm (Banerjee, 1998).

Power test: The power test (Wolfe and Tseng, 1992) is a combination of the Extended GCD test and the Fourier-Motzkin methods to eliminate variables in a system of inequalities. The name of Power test is derived from the power and precision of the method, but in fact that it takes exponential time (in the number of loop index variables) in the worst case.

Omega test: The omega test (Pugh, 1992) determines whether there is an integer solution to an arbitrary set of linear equalities and inequalities, referred to as a problem.

Lambda test: The lambda test (Zi *et al*., 1990) is an efficient and accurate data dependence analysis. It extends the numerical methods to allow all dimensions to be tested concurrently.

Integrated methods

Practical test: The practical test (Goff *et al*., 1991) is based on classifying pairs of subscripted variable references and constrained to some kind of input cases.

MHL test: The MHL test (Maydan *et al*., 1991) is a cascade method, that is to say, the extended GCD test is applied first, if it fails, another testing methods are applied next.

K test: The K (knowledge-based) (Shih *et al*., 1994; Yang *et al*., 1997) test is an approach, which chooses an appropriate test by knowledge-based techniques and then applies the resulting test to detect data dependence, solving as normal tests.

Weighting methods: The weighting methods are based on evaluation of weight to draw a conclusion, an appropriate suggestion for the answer. The kernel of weighting system consists of an evaluation function of weight and the definition weight of characteristics, which dominate the whole process of the evaluation. The feasibility of this method is that when we find a new factor which deeply affects the result of evaluation and then the new factor is joined into the evaluation function of weight without obvious change in the weighting system. The evaluation function of weight can be defined as shown in Eq. 4:

$$E\_w = Imf_1 * Attr\_w_1 + Imf_2 * Attr\_w_2 + \ldots + Imf_n * Attr\_w_n \quad (4)$$

where, E_w is the total weight after evaluating, Imf_i is the important factor of the attribute and the $Attr\_w_i$ is the weight of the attribute, chosen by evaluation. The E_w, computed by evaluation function, is the basis of possible solution or for the suggestion of deciding the next step. For example, the approach has some similar aspects in common with IBM Deep Blue (Hamilton and Garber, 1997; Newborn, 1997).

PROPOSED WEIGHTING CHARACTERISTIC TEST

For each existing test algorithms, they have their own favors on some characteristics and that is the motivation to propose an integrated testing algorithm which makes good use of their advantages in different cases. Although the exactness of the result of test algorithm is more important, the overhead is also an important factor to an integrated test algorithm; otherwise, it would loose its meaning, making good use of test algorithm's advantages.

Framework of the weighting characteristic method: The proposed weighting characteristic method is constructed by two major parts: one is the evaluation function of weighting characteristics and the other is the test

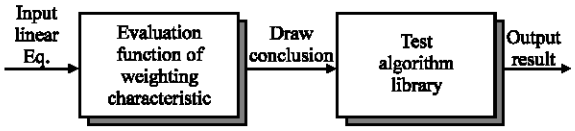

Fig. 5: Framework of the weighting characteristic test

algorithm library. First, the input, a set of equations, is fed into the evaluation function of weight. Then, the evaluation unit evaluates the total weight by computing the evaluation equation, constructed by some important attributes which deeply affect the result of evaluation and draws a conclusion, an appropriate test being applied to data dependence testing. Finally, the resulting test is applied and the answer generated. To simplify the complexity of weight-based system in our approach, a simple framework is proposed for being applied for data dependence testing and a weighting characteristic system is chosen to reduce overhead and to increase feasibility of extension since it has effective evaluation ability that has a good evidence of application in IBM Deep Blue. The framework of the weighting characteristic test is as shown in Fig. 5.

The kernel of weighting characteristic system is based on the evaluation function of weight, constructed by simple equations. In order to construct the evaluation function, important characteristics, existing in each test algorithm, must be extracted and the most affective characteristics are chosen for constructing the evaluation function. The input equations always include necessary information, which is applied to the evaluation phase. By using the information from input equations, the necessary characteristics can be obtained and applied to the evaluation function and get the evaluation weight. Moreover, an appropriate algorithm is suggested being applied to data dependence testing.

As shown in Fig. 5, four test algorithms are chosen in the Test Algorithm Library (TAL). There are Banerjee test, GCD test, I test and Power test. The four tests are chosen to show the exactness and efficiency. The Banerjee test finds a real solution within limits, the GCD test, on the contrary, finds an integer solution ignoring the limits, the I test is a combination of the Banerjee test and The GCD test and the Power test is a very exact and powerful test, but expensive cost. The basis of our choice is based on experiences. Although it may be not the best choice, the experimental results show that it can satisfy for some extent. Traditionally, the Banerjee test, the GCD test and the I test are single dimensional test algorithms, so we prefer to apply the three tests when data dependence testing is needed for array references of single dimension in our approach. The Power test is applied when complex simultaneous equalities or inequalities appear to solve data dependences since it can handle complex simultaneous loop limits, but suffer expensive cost of execution time.

For examining the test algorithms, there are four major characteristics which dominate the selection of test algorithm in the TAL to be described as follows:

- **Bound:** Whether indices bounds of loop are known or not.
- **Coeff:** Whether the coefficients of variables in simultaneous equation system are 1, 0, -1, or not.
- **Dim:** Whether the array reference is single dimensional or multidimensional.
- **Var_num:** Whether the number of variables in the simultaneous equation system is small or large.

Although the four characteristics dominate the selection of test algorithms, we don't know what the dominating degree is for each characteristic. Therefore, the dominating degree for each characteristic can be quantified as Table 1.

For simplicity, two quantities, 4 and 1, are defined as weights for four characteristics in Table 1. For example, the weight of Bound is defined to be 4, when a loop bound is known, otherwise, to be 1.

After defining characteristic weight, the best characteristic weights for each algorithm in the TAL can be obtained and the corresponding weights of the four characteristics for each test algorithm in the TAL can be shown in Table 2.

There are two important terms which are characteristic weights and important factors of characteristic in our evaluation function of weight. The weights of characteristic have been defined above and the important factor of characteristic, based on experience, is defined by each test algorithm. The result of definition is shown in Table 3.

The important factors of characteristic are bounded between 1 to 4 based on the number of test algorithms in the TAL and the number of dominating characteristics of test algorithm. For example, the Banerjee test is dominated by loop bounds, so the important factor of Bound characteristic is defined to be the largest one, 4. In fact, the range of important factor in Table 3 can be expanded as the number of test algorithms is increased in the TAL.

There must be a corresponding reference weight for every test algorithm to map the result of evaluation function. The reference weight for each test algorithm can be formulated as Eq. 5.

$$Ref\_W = Imf_1 * Attr\_W_1 + Imf_2 * Attr\_W_2 + Imf_3 * Attr\_W_3 + Imf_4 * Attr\_W_4 \quad (5)$$

Table 1: Characteristic weight

	Bound		Dim		Coeff		Var_num	
	Known	Unknown	Single	Multi	1, 0, -1	^ (-1, 0, 1)	< 3	> = 3
Weight	4	1	4	1	4	1	4	1

Table 2: Suggestion weights of characteristic for each test algorithm

	Algorithm			
Attri	Banerjee	GCD	I	Power
Bound	4	1	1	1
Coeff	4	1	1	1
Dim	4	4	4	1
Var_num	4	4	1	1

Table 3: Important factor of characteristics

	Algorithm			
Attri	Banerjee	GCD	I	Power
Bound	4	1	2	1
Coeff	1	2	2	2
Dim	3	3	2	1
Var_num	2	2	1	1

Table 4: Reference weights of test algorithms

	Algorithm			
Weight	Banerjee	GCD	I	Power
Ref_w	40	23	13	5

The Ref_W represents the reference weight after summing up each product of importance factor and characteristic weight. The Imf_i and $Attr\_W_i$ represent the importance factor of the ith characteristic and its corresponding weight, respectively. The necessary computing data, Imf_i and $Attr\_W_i$, in Eq. 5 can be obtained according to Table 2 and 3. For example, the reference weight of the Banerjee test can be computed as 4*4+1*4+3*4+2*4 = 40. The reference weight of the test algorithm in library can be summarized in Table 4 by Applying Eq. 5. Obviously, the reference weight of the Banerjee test is the largest one, because of its more constrains to be applied, on the contrary, the reference weight of the Power is the smallest since it has better ability of solving problem with less constrain.

Proposed weighting characteristic test algorithm: The details of the weighting characteristic test can be summarized as an algorithm, which consists of two passes as shown in Fig. 6.

For the input parts of the weighting characteristic test, first, the *m* linear simultaneous equations which have n coefficients of a_0^i, a_1^i, …, a_n^i are fed into the test algorithm. The corresponding limits of coefficients are lower bound, M_j^I and upper bound, N_j^I, respectively, where i is from 1 to m and j is from 1 to n. Secondary, the weights of the four characteristics, Coeff_W, Bound_W, Dim_W and Var_num_W, are also fed into the test algorithm. There are three results, 'No', 'Yes', or 'Maybe', appearing in the output. According to the output result of the test algorithm in the TAL, there is only one chance produced 'Maybe' result by one algorithm-the I test, however, it may be more precise than the Banerjee test

Algorithm: (Weighting Characteristic Test)
Input:
$(a_0^1, a_1^1, \ldots, a_n^1, M_1^1, N_1^1, \ldots, M_n^1, N_n^1,$
$\ldots\ldots\ldots\ldots$
$a_0^m, a_1^m, \ldots, a_n^m, M_1^m, N_1^m, \ldots, M_n^m, N_n^m,$
Coeff_W, Bound_W, Dim_W, Var_num_W)
Output:

No : The input is not integer solvable
or Yes : The input is integer solvable
or Maybe : The input may be integer solvable
Pass 1:
Call Evaluation_Function to draw a conclusion, that is, the most suitable dependence test.
Pass 2:
Call the corresponding testing algorithm in TAL.

Fig. 6: The weighting characteristic test

and the GCD test, where they may produce tentative 'Yes', but the I test doesn't. In Pass 1, the weighting characteristic test calls Evaluation_Function to draw a conclusion for choosing the most appropriate test in the TAL. Once the conclusion is drawn by Evaluation_Function and a suitable test is applied for detecting data dependences at Pass 2.

To reduce the computation overhead, the evaluation function algorithm is constructed as low as possible for time complex. The evaluation function algorithm of weight is summarized into an algorithm-Evaluation_Function as shown in Fig. 7.

The two-dimension array Imf is declared for storing the data of Table 3. The one-dimension arrays Total_W and Ref_W are declared for storing the evaluation results and the data of Table 4, respectively. A flag variable is declared for indicating which test algorithm in the TAL should be applied and flag is initialized by zero. The total weight of test for each test is computed by evaluation function during execution phase. The evaluation function is formulated as Eq. 6.

$$\begin{aligned} Total\_W &= Imf\_of\_bound * Bound\_W \\ &+ Imf\_of\_coeff * Coeff\_W \\ &+ Imf\_of\_dim * Dim\_W + Imf\_of\_var\_num \\ &* Var\_num\_W \end{aligned} \quad (6)$$

```
Algorithm: (Evaluation_Function)
  begin{ Evaluation_Function}
      declare
          integer Imf[4,4], Tatol_W[4], Ref_W[4],
              flag;

      initial
          Imf array by Table 3,
          Ref_w array by Table 4,
          flag by 0;

       for i = 0 to 3 do
          begin
              Total_w[i] = Imf[i,0] * Bound_w +
                           Imf[i,1] * coeff_w +
                           Imf[i,2] * Dim_w +
                           Imf[i,3] * Var_num_w;
              If Total_w[i] ≥ Ref_w[i]
                  then
                           set the i^th bit of flag
                           equal to 1
                  else
                           set the i^th bit of flag
                           equal to 0;
          end{for}

          switch (flag)
              begin
                  case 15 :
                           call Banerjee;
                           break;
                  case 7 :
                           call GCD;
                           break;
                  case 3 :
                           call I;
                           break;
                  case 1 :
                           call Power;
                           break;
                  default :
                           exception;
                           break;
              end{switch}
end{Evaluation_function}
```

Fig. 7: The evaluation function

The total weight of each test is evaluated within each of loop iteration and compares the reference weight with the total weight of each test to set the corresponding position in flag. The corresponding bit position of each test algorithm in flag is as shown in Fig. 8.

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	Banerjee	GCD	I	Power

Fig. 8: Corresponding bits for flag

The arranging bit position is according to the reference weight of each test algorithm. By the decreasing order of reference weight, the Banerjee test is arranged at the b_3 bit position and the Power test arranged at the b_0 bit position. For example, the content of corresponding flag, (00000111), is equal to 7, that is, the corresponding bits position of the GCD test, the I test and the Power test are set to 1, but the Banerjee test is set 0. Obviously, a successor bit of flag is also set to 1 when a predecessor bit is set to 1. Because the corresponding bit of test algorithm is arranged by its reference weight in the flag, that is, when a test algorithm, being the former bit position of the flag, can be applied and a test algorithm, being the later bit position of the flag, can be also applied, too. Finally, a 'case-switch' structure is used to select an appropriate test algorithm according to the content of the corresponding flag.

Example for the weighting characteristic test: To describe the execution process of the weighting characteristic test clearly, an example of data dependence testing, as shown in Fig. 9.

The program segment shows that it is a bound unknown loop and the data dependences exist between S_1 and S_2 due to the two references to array A. The subscripts of array A referenced in S_1 should be equal to that in S_2. Therefore, the linear equation is derived as follows:

$$i_1 = i_2 + 2 => i_1 - i_2 = 2$$

As the linear equation of data dependence is generated from the loop, the weights of characteristics are set as follows:

- Bound_W = 1: The loop bounds are unknown.
- Coeff_W = 4: The coefficients are -1's, 1's.
- Dim_W = 4: This is a single dimensional array references.
- Var_num_W = 4: The number of variables are two, thus less than three.

After setting the weights of characteristics, the weighting characteristic test enters the Pass 1 for execution and the evaluation function is called to draw a conclusion. The process of the total evaluating weight for each test algorithm is shown as follows:

- Banerjee_W = 4 * 1 + 1 * 4 + 3 * 4 + 2 * 4
  = 28 ≤ 40

```
      Do I = M, N
           …
S1 :        A[I] = …
S2 :        … = A[I+2]
           …
      EndDO
```

Fig. 9: An example of program segment for data dependence

The corresponding bit of the Banerjee test in flag is set to zero.

- GCD_W = 1 * 1 + 2 * 4 + 3 * 4 + 2 * 4
  = 29 ≥ 28

The corresponding bit of the GCD test in flag is set to one.

- I_W = 2 * 1 + 2 * 4 + 2 * 4 + 2 * 4
  = 22 ≥ 13

The corresponding bit of the I test in flag is set to one.

- Power_W = 1 * 1 + 2 * 4 + 1 * 4 +1 * 4
  = 17 ≥ 5

The corresponding bit of the power test in flag is set to one.

Based on the setting of evaluation function, the content of flag is shown in two forms, base 10 and base 2, shown as:

$$flag = 00000111_2 => flag = 7_{10}$$

With the result of flag, the evaluation function draws a conclusion, the GCD test algorithm being the most suitable test to be applied. After Pass 2, the GCD test is invoked and the result shows that the array references are data dependent. Hence, they could not be executed in parallel fully.

RESULTS AND DISCUSSION

Experimental assumptions: The experiments are performed on Pentium 4 PC with Microsoft XP O.S. and a CISC processor of Intel 2-GHz. First, four test algorithms, the Banerjee test, the GCD test, the I test and the Power test, are encoded in C programming language. To simplify the process of constructing experimental environment, a simple program which reads the input patterns of input equations and four characteristic weights is constructed

```
Do I = 1 to 100
    Do J = 1 to 50
        …
S1:     A[I,J+10] = …
S2:         … = A[I-10,J]
        …
    Enddo
Enddo
```

Fig. 10: The program segment for data dependence testing

```
1 : 2 4 4 4 1 1
2 : 1 1 100
3 : -1 1 100
4 : 0 1 50
5 : 0 1 50
6 : 10
7 : 0 1 100
8 : 0 1 100
9 : 1 1 50
10 : -1 1 50
11 : -10
```

Fig. 11: The input pattern for test algorithms

and implemented for the evaluation methods of the weighting characteristic test, then draws a conclusion and the corresponding test is invoked. Array subscripted references from input case is selected for dependence testing and encoded into an input file by hand.

For example, as shown in Fig. 10, the program segment which shows a two-dimension array reference pair should be tested for true dependences and the program can be transformed to solve the following simultaneous equations.

$$\begin{cases} i_1 - i_2 = 10 \\ \qquad \text{where} 1 \le i \le 100, 1 \le j \le 50 \\ j_1 - j_2 = -10 \end{cases}$$

The simultaneous equations can be encoded into the form of Fig. 11 by applying the methods which had been proposed by Pugh (1992) and Shen *et al.* (1990).

Obviously, this is a multidimensional array reference and the bounds of indices are known. The coefficients of the simultaneous equations are 1 and -1, respectively and the number of variables is greater than threes.

To avoid the situation of contriving special case to demonstrate the power of the approach, in the first case, there are 29 array reference pairs chosen from Livermore Loops (Ben-Asher and Haber, 2001; Mashey, 2005) to show that our approach is not the best choice for the very

simple array reference pairs and they can be judged directly by simple tests. In the second case, there are 91 array reference pairs of loops chosen from LAPACK (Psarris and Kyriakopoulos, 2001; Cunha *et al*., 2002; Kajiyama *et al*., 2005) which is a physical and great deal of codes to demonstrate the exactness of our approach.

Execution time and overhead: In order to compare the execution performance for the proposed method, the execution time of five tests, the Banerjee test, the GCD test, the I test, the Power test and the W test are shown in Table 5 for the array references of Fig. 12.

The entries of the first two rows in Table 5 represent the execution time of inspecting the array reference on our machine.

According to the first row data which does not apply weighting characteristic test (W test) in Table 5, the Power test spends much more execution time obviously, so it is an expensive test algorithm. The second row data denotes the execution time when the W test applies the four tests. The entries of the first two rows imply if array references pairs are simple, then a simple test, such as the Banerjee test, the GCD test, or the I test, is chosen to be applied for a short execution time.

Table 5 shows the introduced overhead caused by the W test for the four tests, respectively. Although the overhead of the W test is almost 100% comparing with simple test, the W test is smaller overhead than other integrated methods (Goff *et al*., 1991; Shih *et al*., 1994) which cause the thousands of times attached overhead when they are compared with simple test algorithms.

Exactness: Livermore Loops and LAPACK are adopted for the benchmark programs. Twenty-nine array reference pairs are chosen from Livermore Loops for data dependence testing. The experimental results using the Livermore Loops are as shown in Table 6. Notations, A, D, I and M denote the number of times that the test is applied, the test proves the pair is data dependent, independent and the pair maybe has data dependence, respectively. The array subscripts of the Livermore Loops are simple and can be solved well by simple tests, such as the GCD test and the I test, but the Banerjee not. The applied frequencies of the four tests in the W test are analyzed as shown in Fig. 13. The W test almost applies the GCD test and the I test within twenty-nine times of array references pairs and gives exact result, compared with the Power test. Although the Power test always gives the correct result for data dependence testing, the cost of the Power is too expensive. The W test also can give correct result without applying the Power test any time, because it evaluates an appropriate test to be applied with reasonable cost and exactness.

```
    Do I = 1, 10
        Do J = 1, 10
S1:         A[I, J] = …
S2:                 =A[J, J-1]
        Enddo
    Enddo
```

Fig. 12: The input loop pattern of experiment for execution time and overhead

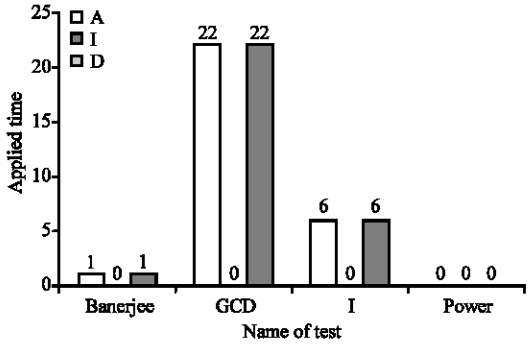

Fig. 13: Applied frequencies of four tests within W test for livermore loops

Table 5: Execution time (sec) and overhead on PC

Variables	Banerjee	GCD	I	Power
Without W	1.53×10^{-4}	1.56×10^{-4}	1.61×10^{-4}	0.921
With W	3.15×10^{-4}	3.20×10^{-4}	3.23×10^{-4}	0.9212
Overhead	105.2%	101.9%	100.0%	0.018%

Ninety-one array references pairs are chosen from the LAPACK benchmark program for data dependence testing. The Banerjee can not be applied at all, because the bounds of array subscripts are unknown since they are parameters of subroutines (Table 6). The GCD test and the I test loose their precision since seven incorrect results are found. Although the Power test always gets the correct result in a variety of benchmark programs, it is too expensive for all array reference pairs. The W test shows high ratio of precision can be achieved and only one incorrect result, as shown in Table 7, is produced. The applied frequencies of the four tests within the W test are analyzed as shown in Fig. 14. The GCD test and the I test also take a major proportion of applied frequencies within 91 array reference pairs for the W test. The W test just applies the Power test 7 times and gets high accuracy.

The W test gets high precision (100 and 98.90%, respectively) for both benchmark programs. The Banerjee test, the GCD test, the I test, the Power test and the W test are compared for the exactness of LAPACK and the result is as shown in Fig. 15. The exactness of the Power test is the base of normalization, that is, the exactnesses

Table 6: Frequencies of application/independence/dependence/maybe for livermore loops

Banerjee				GCD				I				Power				W			
A	I	D	M	A	I	D	M	A	I	D	M	A	I	D	M	A	I	D	M
1	0	1	0	29	0	29	0	29	0	29	0	29	0	29	0	29	0	29	0

Table 7: Frequencies of Application/Independence/Dependence/Maybe for LAPACK

Banerjee				GCD				I				Power				W			
A	I	D	M	A	I	D	M	A	I	D	M	A	I	D	M	A	I	D	M
0	0	0	0	91	0	91	0	91	0	91	0	91	7	84	0	91	6	85	0

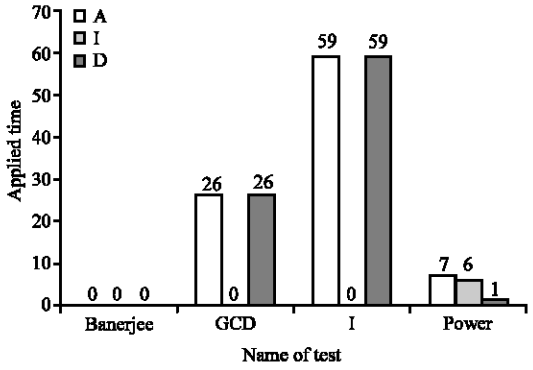

Fig. 14: Applied frequencies of four tests within W test for LAPACK

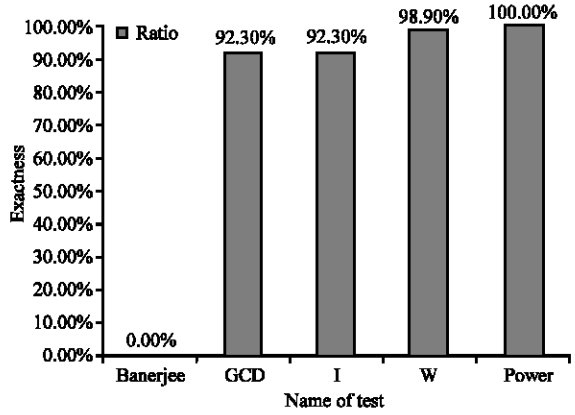

Fig. 15: Exactness of five tests for LAPACK

of the other four tests are normalized to the Power test. Clearly, the W test has the relatively high ratio of exactness, approaching to 99%. The other three simple tests are obviously low, because they are only suitably used in simple array references.

To demonstrate the efficiency of the W test, the execution time of the LAPACK is illustrated in Fig. 16. Similarly, the execution time of the Power test is used as the base of normalization and the other four tests are normalized to the Power test. Although the execution time of the W test is higher than the other three tests, it is

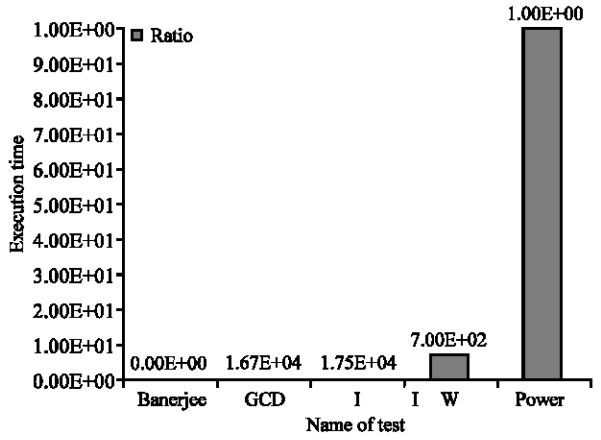

Fig. 16: Execution time of five tests for LAPACK

significantly lower than the Power test and it is an acceptable method for parallelizing compilers.

CONCLUSION AND FUTURE WORKS

In this study, we present an integrated approach, a weighting character method by evaluating the weight of input data of loop, for data dependence testing. For implementing the weighting characteristic test, the evaluation function is established to generate the total weight for each of the tests in the TAL. The experimental results demonstrate that both good efficiency and precision can be achieved by the weighting characteristic test.

The proposed method improves overhead evidently and the experimental results demonstrate the efficiency. Our approach has proven the precision of two benchmark programs, Livermore Loops and LAPACK and the experimental results demonstrate that the precision of Livermore Loops is 100% and the LAPACK is approaching to 99%.

The traditional parallelizing compilers select one kind of data dependence test for detecting data dependences of loops within a program. However, the proposed approach provides the feasibility of test algorithm selection by the input data of loops within a program, that

is, there may be one or more test algorithms being applied within a program. According to the conclusion of the evaluation, an appropriate test algorithm can be selected automatically.

REFERENCES

Ahamed, M., H. Eldeeb, S. Nassar and N. Bagherzadeh, 2001. Design and implementation of automatic parallel detection layer. Proceedings of IEEE International Conference on Systems, Man and Cybernetics, 4: 2371-2376.

Banerjee, U., S.C. Chen, D.J. Kuck and R.A. Towle, 1979. Time and parallel processor bounds for Fortran-like loops. IEEE Trans. Comput., 28 (9): 660-670.

Banerjee, U., 1998. Dependence Analysis for Supercomputing. Kluwer Academic Publishers.

Baude, F., D. Caromel, L. Henrio and M. Morel, 2007. Collective interfaces for distributed components. Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid, pp: 599-610.

Ben-Asher, Y. and G. Haber, 2001. Parallel solutions of simple indexed recurrence equations. IEEE Trans. Parallel Distributed Syst., 12 (1): 22-37.

Bertrand, F., R. Bramley, A. Sussman, D.E. Bernholdt, J.A. Kohl, J.W. Larson and K.B. Damevski, 2005. Data redistribution and remote method invocation in parallel component architectures. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Washington, DC, USA., pp: 40b-40b.

Damevski, K., Z. Keming and P. Steven, 2007. Practical parallel remote method invocation for the Babel compiler. Proceedings of the 2007 Symposium on Component and Framework Technology in High-performance and Scientific, pp: 131-139.

Chang, W.L., J.W. Huang and C.P. Chu, 2004. Using elementary linear algebra to solve data alignment for arrays with linear or quadratic references. IEEE Trans. Parallel Distributed Syst., 15 (1): 28-39.

Cunha, M.T.F., J.C.F. Telles and A.L.G.A. Coutinho, 2002. Parallel boundary elements using LAPACK and ScaLAPACK. Proceedings of 14th Symposium on Computer Architecture and High Performance Computing, pp: 51-58.

Goff, G., K. Kennedy and C.W. Tseng, 1991. Practical dependence testing. Proceedings of ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, Toronto, Canada, pp: 15-29.

Hamilton, S. and L. Garber, 1997. Deep blue's hardware-software synergy. Computer (Innovative Technology for Computer Professionals), 30 (10): 29-35.

Hwang, K., 1986. Advance Computer Architecture. McGraw-Hill International Editions.

Kajiyama, T., A. Nukada, R. Suda, A. Nishida and H. Hasegawa, 2005. LAPACK in SILC: Use of a flexible application framework for matrix computation libraries. Proceedings of the 8th International Conference on High-Performance Computing in Asia-Pacific Region.

Knuth, D.E., 1981. The Art of Computer Programming. Seminumerical Algorithms. 2nd Edn. Vol. 2, Reading, MA: Addison-Wesley.

Kong, X., D. Klappholz and K. Psarris, 1991. The I test: An improved dependence test for automatic parallelization and vectorization. IEEE Trans. Parallel Distributed Syst., 2 (3): 342-349.

Mashey, J.R., 2005. Summarizing performance is no mean feat [computer performance analysis]. Proceedings of the IEEE International Workload Characterization Symposium.

Maydan, D.E., J.L. Henny and M.S. Lam, 1991. Efficient and exact data depence analysis. Proceedings of ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, Toronto, Canada, pp: 15-29.

Mineo, M., T. Uehara, S. Saito and Y. Kunieda, 2003. A new practical array data dependence analysis for parallelizing compilers. Proceedings of Innovative Architecture for Future Generation High-Performance Processors and Systems, pp: 78-87.

Mineo, M., S. Saito, T. Uehara and Y. Kunieda, 2004. Implementation details and evaluation of a new exact and fast test for array data dependence analysis based on simplex method. Proceedings of Innovative Architecture for Future Generation High-Performance Processors and Systems, pp: 89-100.

Newborn, M., 1997. Kasparrov Versus Deep Blue: Computer Chess Comes of Age. Springer-Verlog, New York.

Psarris, K. and K. Kyriakopoulos, 2001. Data dependence analysis for complex loop regions. Proceedings of International Conference on Parallel Processing, pp: 195-204.

Pugh, W., 1992. A Practical algorithm for exact array dependence analysis. Commun. ACM., 35 (8): 102-114.

Shen, Z., Z. Li and P.C. Yew, 1990. An empirical study of Fortran programs for parallelizing compilers. IEEE Trans. Parallel Distributed Syst., 1 (3): 356-364.

Shih, W.C., C.T. Yang and S.S. Tseng, 1994. Knowledge-based data dependence testing on loops. Proceedings of International Computer Symposium, Hsinchu, Taiwan R.O.C., pp: 961-966.

Van Engelen, R.A., J. Birch and K.A. Gallivan, 2004. Array data dependence testing with the chains of recurrences algebra. Proceedings of Innovative Architecture for Future Generation High-Performance Processors and Systems, pp: 70-81.

Wolfe, M. and C.W. Tseng, 1992. The Power test for data dependence. IEEE Trans. Parallel Distributed Syst., 3 (3): 591-601.

Yang, C.T., S.S. Tseng, C.D. Chuang and W.C. Shih, 1997. Using knowledge-based techniques on loop parallelization for parallelizing compilers. Parallel Comput., 23 (3): 291-309.

Zi, Z., P.C. Yew and C.Q. Zhu, 1990. An efficient data dependence analysis for parallelizing compilers. IEEE Trans. Parallel Distributed Syst., 1 (1): 26-34.

Zima, H.P. and B. Chapman, 1990. Supercompilers for Parallel and Vector Computers. Addison-Wesley Publishing, ACM Press, New York.