J Supercomput (2015) 71:340–368
DOI 10.1007/s11227-014-1298-3

An improved nonlinear data dependence test

Jie Zhao · Rongcai Zhao · Xi Chen · Bo Zhao

Published online: 20 September 2014

Abstract To further parallelize large-scale nonlinear scientific computing applications, some data dependence techniques for nonlinear subscripts, especially for quadratic subscripts, were proposed in the past. The quadratic programming (QP) test and polynomial variable interval (PVI) test are two representative techniques. The QP test, which serves as an exact but time-consuming technique, always gives conservative results when the coefficient matrix of the quadratic terms is not positive semi-definite, while the PVI test will lose efficiency when there exist mixed polynomials in the dependence equation. Focusing on the dependences caused by quadratic subscripts in nonlinear and irregular programs, we propose an improved nonlinear data dependence test in this paper. We first normalize a quadratic equation which is written in a general form, and determine whether the canonical equation is integer solvable in the region of interest based on the interval equation theory. Experimental results show that, compared with the QP test, our method maintains a much lower time complexity. Furthermore, it can detect more general dependences than other dependence testing methods like the PVI test in terms of quadratic subscripts.

Keywords Parallelizing compiler · Data dependence test · Nonlinear subscript · Interval equation

1 Introduction

Data dependence analysis is an essence to identify and exploit parallelism in programs. It has been more than 40 years since the first data dependence test was proposed. Data dependence techniques for linear subscripts were well developed. They have been widely used in modern optimizing compilers. However, the multi-core technology

J. Zhao (✉) · R. Zhao · X. Chen · B. Zhao
State Key Laboratory of Mathematical Engineering and Advanced Computing, Wuxi 214125, People's Republic of China
e-mail: zjbc2005@163.com; zhaojie8037@gmail.com

raises an in-depth parallelization request for almost all the applications of modern computer systems, especially the light-weight multi-core mobile computing platforms. In the meantime, various kinds of scientific and engineering computing tasks, including the environmental and biological large-scale applications, are in need of parallelization. Conventional analysis approaches are not able to satisfy these demands, as they are all based on an assumption that each array subscript is an affine reference of its enclosing loops' indexes. As a result, they usually return a conservative result for nonlinear references, which are becoming more and more ubiquitous in large-scale irregular applications, thereby preventing the parallelization.

It has been proved [1] that the dependence testing problem is actually to determine whether there is an integer solution to the system of equations

$$\begin{cases} f_1(i_1,\ldots,i_n,\ldots,j_1,\ldots,j_n) = h_1(i_1,\ldots,i_n) - g_1(j_1,\ldots,j_n) = 0 \\ f_2(i_1,\ldots,i_n,\ldots,j_1,\ldots,j_n) = h_2(i_1,\ldots,i_n) - g_2(j_1,\ldots,j_n) = 0 \\ \vdots \\ f_m(i_1,\ldots,i_n,\ldots,j_1,\ldots,j_n) = h_m(i_1,\ldots,i_n) - g_m(j_1,\ldots,j_n) = 0 \end{cases} \tag{1}$$

which subjects to the region $\boldsymbol{R} = \{L_k \le i_k, j_k \le U_k | 1 \le k \le n, i_k, j_k \in \mathrm{Z}\}$. Here, L_k and U_k are the lower and upper bounds of the kth enclosing loop. h and g in each equation represent the referenced subscript expressions, respectively. In recent years, due to the challenges of multi-core technology and parallelism requirements of large-scale irregular computing tasks, data dependence techniques focusing on nonlinear subscripts have also been widely studied.

Blume and Eigenmann [2] proposed a nonlinear dependence test called Range test. They consider h and g of each equation, respectively, and calculate their extreme values individually. If they are both monotonous and the maximum value of h $h_{\max}$ is less than the minimum value of g $g_{\min}$, the analysis determines that there is no dependence. Literature [3] described the Quadratic test, which is designed for cases when a dependence equation is quadratic and only one loop index appears in the subscripts. Since loop-carried dependences should be considered, there will be two variables x and y in the equation, which are different instances of the same loop index. The algorithm iterates one variable with the other so as to narrow their value intervals. When either of their intervals becomes null, the iteration process will be terminated and "no dependence" is reported. Otherwise, each interval will be reduced to one point, hereby resulting in the solution of the equation. If it is not an integer solution, there is no dependence. Literature [4] presented an integer interval theory-based nonlinear dependence test called polynomial variable interval (PVI) test. It views a dependence equation as an interval equation and eliminates its variables one by one. Finally, only one constant term and an integer interval in the left- and right-hand expressions are left, respectively. If the constant term is not in the interval, there is no dependence.

It is not so easy to solve a system of nonlinear equations. In practice, a quadratic subscript is relatively common than other cases [3,5,6]. Hence, the research community developed specific data dependence techniques for quadratic subscripts. For instance, the Quadratic test described above is such a technique. It is based on a subscript-by-subscript idea, which solves each equation step by step instead of the whole system.

The dependence equation is required to be in the form of

$$ax^2 + by = c \tag{2}$$

where x and y are different instances of the same loop index. The drawback of the Quadratic test is manifest, but it is an exact test, saying it will find the integer solutions of the equation when there is a dependence. To solve this problem, literature [5] designed a quadratic programming-based nonlinear dependence test called quadratic programming (QP) test. It regards the left-hand expression of a dependence equation as the objective function of a quadratic programming problem, and calculates its extreme value. If the minimum value is greater than zero, there is no dependence. When it is not greater than zero, literature [6] analyzed whether a solution making the objective function equal to zero is integer-valued via branch and bound method. If it is not an integer solution, there is no dependence. However, the QP test is efficient only when the coefficient matrix of the quadratic polynomials is positive semi-definite. In other words, the QP test will always return there are dependences when the hypothesis is not satisfied. The PVI test proposed in [4] is not a specific quadratic test, but it can also be applied to analyze a quadratic equation, but mixed polynomials cannot appear in the equation.

Focusing on the quadratic subscripts in modern nonlinear and irregular programs, we propose an improved nonlinear data dependence test in this paper. First, we transform a general quadratic equation into its canonical form. Second, we determine whether the canonical equation is integer solvable in the region of interest with the interval equation theory. Compared with existing techniques, our method can either analyze more general quadratic dependences [4] or has a lower time complexity [5,6].

The remainder of this paper is structured as follows. Section 2 reviews existing quadratic data dependence tests and presents our motivation. Section 3 describes the process of normalizing a general quadratic equation. Section 4 introduces the interval equation theory and some corresponding theorems to eliminate polynomials of an equation. Section 5 describes the improved test algorithm. In Sect. 6, a case study is illustrated to help readers to understand the algorithm. Section 7 performs our experimental evaluations and Sect. 8 shows the theoretical analysis. The related work is reviewed in Sect. 9. Finally, our conclusions are placed in the last section.

2 Motivation

As described above, the Quadratic test [3] can only analyze a set of equations with one loop iteration variable in each equation. It requires that the dependence equation must be in the form of Eq. (2). Reorganize it as following

$$y = (c - ax^2)/b \tag{3}$$

then the monotony of the variable y in the range of interest can be inferred according to the restricted range $[L, U]$ of x and the difference function when $\Delta x = 1$. L and U are the lower and upper bounds of the enclosing loop. If y is monotonically increasing or decreasing, the dependence equation (2) can also be written as

$$x^2 = (c - by)/a \tag{4}$$

Accordingly, the generated lower and upper bounds of x are

$$\begin{aligned} L' &= \sqrt{\left(-\tfrac{b}{a}\right)^{+} L - \left(-\tfrac{b}{a}\right)^{-} U + \tfrac{c}{a}} \\ U' &= \sqrt{\left(-\tfrac{b}{a}\right)^{+} U - \left(-\tfrac{b}{a}\right)^{-} L + \tfrac{c}{a}} \end{aligned} \tag{5}$$

If we intersect $[L, U]$ with $[L', U']$, a new range of x is obtained. Since the Eq. (3) is monotonous, the repeating process is convergent and x's range will be either reduced to a point or null.

If y is not monotonous, the corresponding figure of the Eq. (3) is a parabola. In this case, it will try to find the extreme value of the parabola and then divide the interval into two parts to proceed.

To analyze more general dependence equations, the QP test is proposed to calculate quadratic programming problem whose objective function is the left-hand expression $f(\boldsymbol{x})$

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) = \boldsymbol{C}^T \boldsymbol{x} + \frac{1}{2} x^T \boldsymbol{H} \boldsymbol{x}, \\ \text{s.t.} \quad & A\boldsymbol{x} \leq b, \boldsymbol{x} \geq 0, \end{aligned} \tag{6}$$

In this model, $f(\boldsymbol{x})$ represents the objective function of the system, while $A\boldsymbol{x} \leq b$, $\boldsymbol{x} \geq 0$ are the constraints introduced by the bounds of loop nests. A is a matrix in which each element is the coefficient of the loop index in its reference. First, the technique determines whether the coefficient matrix $\boldsymbol{H}$ of the quadratic polynomials is positive semi-definite. If it is, the minimum $f(\boldsymbol{x})_{\min}$ of the quadratic programming system (6) can be figured out. If $f(\boldsymbol{x})_{\min}$ is greater than zero, there is no solution to the dependence equation, saying there is no dependence. If $f(\boldsymbol{x})_{\min}$ is not greater than zero, try to find the integer solutions making $f(\boldsymbol{x}) = 0$. If no integer feasible solution is found, there is no dependence.

The PVI test is not a specific quadratic technique. For a quadratic subscript, a dependence equation must be in the following form

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} a_i x_i^2 + \sum_{i=1}^{n} b_i x_i = c \tag{7}$$

which implies that a mixed polynomial should not appear in the equation. Here, $x_i (1 \leq i \leq n)$ represents an instance of a loop index. That is to say, the PVI test can analyze a quadratic dependence equation only when the coefficient matrix $\boldsymbol{H}$ of the quadratic polynomials is in the form of

Table 1 Quadratic dependence tests and their limitations

Tests	Limitations
Quadratic test	Only one loop iteration variable is allowed
QP test	Matrix $\boldsymbol{H}$ must be positive semi-definite
PVI test	A mixed polynomial is not permitted

$$\boldsymbol{H} = 2 \times \begin{bmatrix} h_{11} & & & \\ & h_{22} & & \\ & & \ddots & \\ & & & h_{nn} \end{bmatrix}$$

Therefore, we summarize the quadratic data dependence tests and their limitations in Table 1. To solve the problems listed in Table 1, we attempt to propose and implement an improved nonlinear data dependence technique for quadratic subscripts. We suppose that the dependence equation is of the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} a_i x_i^2 + \sum_{i=1}^{n} \sum_{j=i+1}^{n} b_{ij} x_i x_j + \sum_{i=1}^{n} c_i x_i = d \tag{8}$$

In the first place, it can be seen that any $x_i (1 \leq i \leq n)$ can appear in Eq. (8), while only two variables are allowed in Eq. (2). As a result, there is no constraint on the number of variables of this equation, which means any number of loop index variables can appear in the subscripts. In the second place, any mixed polynomial $x_i x_j (1 \leq i \leq n, i \leq j \leq n)$ can be seen in Eq. (8), so mixed polynomials are also seen in this equation, which breaks the PVI test's limitation. Finally, it can be seen from the following sections that our method will not require the matrix $\boldsymbol{H}$ to be positive semi-definite.

3 The canonical form of dependence equations

The limitation of the PVI test is that a mixed polynomial is not allowed. It is manifest that the only difference between the Eqs. (7) and (8) is the mixed polynomials. Therefore, if we can eliminate the mixed polynomials from an equation without altering its meaning, then we can still determine whether the equation is integer solvable.

Theorem 1 *All mixed polynomials of the Eq.* (8) *can be eliminated without altering its meaning.*

Proof First, we should show a matter of fact as everyone knows. For any integer x_i and x_j which represent different instances of loop indexes, the following is preserved

$$(x_i + x_j)^2 = x_i^2 + x_j^2 + 2x_i x_j$$

Hence, we have

$$x_i x_j = ((x_i + x_j)^2 - (x_i^2 + x_j^2))/2$$

and we can reorder the Eq. (8) as

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} a_i x_i^2 + \sum_{i=1}^{n} \sum_{j=i+1}^{n} b_{ij} x_i x_j + \sum_{i=1}^{n} c_i x_i$$
$$= \sum_{i=1}^{n} a_i x_i^2 + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{b_{ij}}{2}((x_i + x_j)^2 - x_i^2 - x_j^2) + \sum_{i=1}^{n} c_i x_i = d \quad (9)$$

At this point, there is no mixed polynomial in the Eq. (9), and no new variable is introduced. So, the mixed polynomials of the Eq. (8) can be eliminated without altering its meaning. □

However, the generated equation brings a quadratic polynomial like $(x_i + x_j)^2$ for each mixed polynomial $x_i x_j$, although it will not produce new variables. If we make $y_{ij} = x_i + x_j$, there will also be a constraint on y_{ij} according to its two components. So, the dependence analysis problem is equal to determining whether the Eq. (9) is integer solvable subject to

$$\boldsymbol{R}' = \{L_k \le x_k \le U_k, y_{ij} = x_i + x_j | 1 \le i, j, k \le n, 1 \le ij \le n(n-1)/2, x_k \in \mathrm{Z}\} \quad (10)$$

4 Interval equation and its theorems

The interval equation theory can be used to address dependence testing problem. For example, the I test [7], which is a conventional dependence analysis designed for linear subscripts, and the PVI test described in previous sections, which is a dependence test for nonlinear subscripts. The I test provides an approach to eliminate a linear polynomial from the left-hand expression of the equation, while the PVI test eliminates nonlinear polynomials.

4.1 Interval equation

Since finding the integer-valued solutions to an equation is extraordinary difficult, we determine whether the equation is integer solvable with interval equation instead of finding such integer-valued solutions. The following are some related definitions about interval equation.

Definition 1 For each integer a, say a^+ and a^- are the positive and negative parts of integer a, respectively, where

$$a^+ = (|a| + a)/2$$
$$a^- = (|a| - a)/2$$

Definition 2 Given an integer region $\boldsymbol{R} \in \boldsymbol{Z}^n$, and two functions L and U from $\boldsymbol{R}$ to $\boldsymbol{Z}$, then the integer interval $[L(\boldsymbol{x}), U(\boldsymbol{x})]$ is the union of all integer intervals for each

x_i of x in $\boldsymbol{R}$

$$[L(\boldsymbol{x}), U(\boldsymbol{x})] = \bigcup_{\boldsymbol{x}_i \in \boldsymbol{R}} [L(\boldsymbol{x}_i), U(\boldsymbol{x}_i)]$$

Definition 3 Given an integer region $\boldsymbol{R} \in \boldsymbol{Z}^n$, and three functions F, L and U from $\boldsymbol{R}$ to $\boldsymbol{Z}$, then the following

$$F(\boldsymbol{x}) = [L(\boldsymbol{x}), U(\boldsymbol{x})] \tag{11}$$

is an interval equation.

The interval equation (11) is integer solvable in the integer region $\boldsymbol{R}$, if and only if there exists such an $\boldsymbol{x}_0 \in \boldsymbol{R}$ that $L(\boldsymbol{x}_0) \leq F(\boldsymbol{x}_0) \leq U(\boldsymbol{x}_0)$. Hence, we can rewrite the dependence equation in the following form

$$\sum_{i=1}^{n} a_i x_i^2 + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{b_{ij}}{2}(y_{ij}^2 - x_i^2 - x_j^2) + \sum_{i=1}^{n} c_i x_i = [d, d] \tag{12}$$

If we can eliminate all the quadratic polynomials from the left-hand side of the Eq. (12), then it is transformed into a linear interval equation. Considering such equations can be determined by the I test, we can address the dependence problem in this way. As for quadratic polynomials, the integer equation theory will transfer them from the left-hand side to right side, and eliminate them on the basis of their own constraints.

4.2 Polynomial elimination

The PVI test provides a theorem to eliminate nonlinear polynomials from a dependence equation.

Lemma 1 [4] *Consider the following variable interval equation*

$$F(\boldsymbol{x}) + aX^n = [L(\boldsymbol{x}) + bX^n, U(\boldsymbol{x}) + cX^n]$$

subject to a set of constraints on $\boldsymbol{x}$ *in* $\boldsymbol{R}$, $P^n(\boldsymbol{x}) \leq X^n \leq Q^n(\boldsymbol{x})$ *and* $n \in \boldsymbol{Z}^+$, *where* X^n *does not appear in any of the constraints in* $\boldsymbol{R}$. *If* $(b-a)(c-a) \leq 0$, *or* $(b-a)(c-a) > 0$ *and* $\min(U(\boldsymbol{x}) - L(\boldsymbol{x}) + (c-b)^+ P^n(\boldsymbol{x}) - (c-b)^- Q^n(\boldsymbol{x}) + 1) \geq \min(|b-a|, |c-a|)$, *then the equation above is integer solvable if and only if the interval equation*

$$\begin{aligned} F(\boldsymbol{x}) = [&L(\boldsymbol{x}) + (b-a)^+ P^n(\boldsymbol{x}) - (b-a)^- Q^n(\boldsymbol{x}), \\ &U(\boldsymbol{x}) + (c-a)^+ Q^n(\boldsymbol{x}) - (c-a)^- P^n(\boldsymbol{x})] \end{aligned}$$

is integer solvable subject to the same constraints on $\boldsymbol{x}$ *in* $\boldsymbol{R}$ *and the constraint* $P^n(\boldsymbol{x}) \leq Q^n(\boldsymbol{x})$.

Because we only need to consider the linear and quadratic polynomials in this paper, we extract two theorems from this lemma as follows.

Theorem 2 *Consider the following variable interval equation*

$$F(\boldsymbol{x}) + aX^2 = [L(\boldsymbol{x}) + bX^2, U(\boldsymbol{x}) + cX^2]$$

subject to a set of constraints on $\boldsymbol{x}$ *in* $\boldsymbol{R}$, $P^2(\boldsymbol{x}) \leq X^2 \leq Q^2(\boldsymbol{x})$, *where* X^2 *does not appear in any of the constraints in* $\boldsymbol{R}$. *If* $(b-a)(c-a) \leq 0$, or $(b-a)(c-a) > 0$ *and* $(\min U(\boldsymbol{x}) - L(\boldsymbol{x}) + (c-b)^+ P^2(\boldsymbol{x}) - (c-b)^- Q^2(\boldsymbol{x}) + 1) \geq \min(|b-a|, |c-a|)$, *then the equation above is integer solvable if and only if the interval equation*

$$\begin{aligned} F(\boldsymbol{x}) = [&L(\boldsymbol{x}) + (b-a)^+ P^2(\boldsymbol{x}) - (b-a)^- Q^2(\boldsymbol{x}), \\ &U(\boldsymbol{x}) + (c-a)^+ Q^2(\boldsymbol{x}) - (c-a)^- P^2(\boldsymbol{x})] \end{aligned}$$

is integer solvable subject to the same constraints on $\boldsymbol{x}$ *in* $\boldsymbol{R}$ *and the constraint* $P^2(x) \leq Q^2(x)$.

Theorem 3 *Consider the following variable interval equation*

$$F(\boldsymbol{x}) + aX = [L(\boldsymbol{x}) + bX, U(\boldsymbol{x}) + cX]$$

subject to a set of constraints on $\boldsymbol{x}$ *in* $\boldsymbol{R}$, $P(\boldsymbol{x}) \leq X \leq Q(\boldsymbol{x})$, *where* X *does not appear in any of the constraints in* $\boldsymbol{R}$. *If* $(b-a)(c-a) \leq 0$, *or* $(b-a)(c-a) > 0$ *and* $\min(U(\boldsymbol{x}) - L(\boldsymbol{x}) + (c-b)^+ P(\boldsymbol{x}) - (c-b)^- Q(\boldsymbol{x}) + 1) \geq \min(|b-a|, |c-a|)$, *then the equation above is integer solvable if and only if the interval equation*

$$\begin{aligned} F(\boldsymbol{x}) = [&L(\boldsymbol{x}) + (b-a)^+ P(\boldsymbol{x}) - (b-a)^- Q(\boldsymbol{x}), U(\boldsymbol{x}) \\ &+ (c-a)^+ Q(\boldsymbol{x}) - (c-a)^- P(\boldsymbol{x})] \end{aligned}$$

is integer solvable subject to the same constraints on $\boldsymbol{x}$ *in* $\boldsymbol{R}$ *and the constraint* $P(x) \leq Q(x)$.

As a matter of fact, Theorem 2 and 3 are two specific forms of Lemma 1. We will therefore not repeat their proofs. We put Theorem 3 here, as the constraint of a loop iteration variable may be a function of other loop indexes. The elimination of such a quadratic variable will introduce the linear form of these indexes to the right-hand expression. We would like to construct an integer interval on the right-hand side, so we also need to eliminate linear variables which appear in the right-hand expression.

After eliminating all the quadratic and linear polynomials which may appear on the right-hand side, a linear interval equation will generate. At this point, the equation can be determined in two ways. One is to eliminate all the remaining linear polynomials with Theorem 3 until the left-hand expression becomes a zero. As the right-hand expression represents an integer interval, the original dependence equation is integer solvable if the zero is in this interval. However, two accuracy conditions are needed to check before using Theorem 3. The other is to invoke the I test. Compared with

Theorem 3, no accuracy conditions should be checked when using the I test. Hence, we decide to invoke the I test algorithm at this point instead of using Theorem 3. So, whether the original equation is integer solvable depends on the result of the I test.

5 The algorithm

From the above, we can summarize the improved quadratic dependence algorithm. It can be divided into four steps.

(1) Reorganize the quadratic dependence equation into its canonical form (9) with Theorem 1, and add each constraint of variable y_{ij} to the region of interest $\boldsymbol{R}$.
(2) Transfer the Eq. (9) into an interval equation (11) according to Theorem 2, and repeat eliminating the quadratic polynomials from the left-hand expression until only linear polynomials are left at this side.
(3) If a linear polynomial is introduced when eliminating a quadratic polynomial or if its constraint is not constant, then eliminate it from the equation with Theorem 3.
(4) At this point, the Eq. (11) has been transformed into a linear dependence equation. Invoke the I test algorithm to determine whether the equation has an integer-valued solution. If it is integer solvable, return there are dependences, otherwise there are no dependences.

Besides, we need to check two accuracy conditions before using Theorems 2 or 3. They are as follows [4].

Accuracy Condition 1. For each variable X, eliminated from the variable interval equation

$$F(\boldsymbol{x}) + aX^k = [L(\boldsymbol{x}) + bX^k, U(\boldsymbol{x}) + cX^k]$$

subject to a set of constraints on $\boldsymbol{x}$ in $\boldsymbol{R}$, $P^k(\boldsymbol{x}) \leq X \leq Q^k(\boldsymbol{x})$, where X does not appear in any of the constraints in $\boldsymbol{R}$, the following inequalities need to be satisfied

$$\begin{aligned}(b-a)(c-a) &\leq 0 \quad \text{or}\\ (b-a)(c-a) &> 0, \min(U(\boldsymbol{x}) - L(\boldsymbol{x}) + (c-b)^+ P^k(\boldsymbol{x}) - (c-b)^- Q^k(\boldsymbol{x}) + 1)\\ &\geq \min(|b-a|, |c-a|)(k=1,2)\end{aligned}$$

Accuracy Condition 2. For each variable X, eliminated from the variable interval equation

$$F(\boldsymbol{x}) + aX^k = [L(\boldsymbol{x}) + bX^k, U(\boldsymbol{x}) + cX^k]$$

subject to a set of constraints on $\boldsymbol{x}$ in $\boldsymbol{R}$, $P^k(\boldsymbol{x}) \leq X \leq Q^k(\boldsymbol{x})$, where X does not appear in any of the constraints in $\boldsymbol{R}$, the following inequalities need to be satisfied

$$\min(Q^k(\boldsymbol{x}) - P^k(\boldsymbol{x})) \geq 0(k=1,2)$$

We therefore list the algorithm of this dependence test as follows.

Algorithm: an improved quadratic dependence test
Input: the dependence equation
Output: true, saying there are dependences, or false means there are no dependences

Procedure Improved_Test(*dependence_equation*)
{
/*step0: Initialization*/
$A \leftarrow \{a_1, a_2, \ldots, a_n\}$;
$B \leftarrow \{b_{12}, b_{12}, \ldots, b_{1n}, b_{23}, b_{24}, \ldots, b_{2n}, \ldots, b_{nn}\}$;
$C \leftarrow \{c_1, c_2, \ldots, c_n\}$;
Quadratic_Terms $\leftarrow \{X_1, X_2, \ldots, X_n\}$;
Linear_Terms $\leftarrow$ Quadratic_Terms;
Lower_Bounds $\leftarrow \{L_1^{(1)}, L_1^{(2)}, L_2^{(1)}, L_2^{(2)}, \ldots, L_n^{(1)}, L_n^{(2)}\}$;
Upper_Bounds $\leftarrow \{U_1^{(1)}, U_1^{(2)}, U_2^{(1)}, U_2^{(2)}, \ldots, U_n^{(1)}, U_n^{(2)}\}$;

/*step1: Standardization*/
$i \leftarrow 1$; $k \leftarrow 1$;
while($i<n$){
while($j>i \&\& j<n$){
/* standardization */
$B \leftarrow B - b_{ij}$;
$b_{ij} \leftarrow b_{ij}/2$;
$a_i \leftarrow a_i - b_{ij}$;
$a_j \leftarrow a_j - b_{ij}$;
$j \leftarrow j+1$;
/* add new bounds */
$L_{n+k}^{(2)} \leftarrow (L_i^{(1)} + L_j^{(1)})^2$;
$U_{n+k}^{(2)} \leftarrow (U_i^{(1)} + U_j^{(1)})^2$;
Lower_Bounds $\leftarrow$ Lower_Bounds + $\{L_{n+k}^{(2)}\}$;
Upper_Bounds $\leftarrow$ Upper_Bounds + $\{U_{n+k}^{(2)}\}$;
Quadratic_Terms $\leftarrow$ Quadratic_Terms + $\{X_{n+k}\}$;
$a_{n+k} \leftarrow b_{ij}$;
$A \leftarrow A + \{a_{n+k}\}$;
$k \leftarrow k+1$;
}
$i \leftarrow i+1$;
}

/*step2: Quadratic term elimination*/
$L \leftarrow d$;
$U \leftarrow d$;
while(Quadratic_Terms $\neq \varnothing$){
Stack $\leftarrow \varnothing$; $i \leftarrow 1$;
call Find_Approximate_Terms(Quadratic_Terms, Lower_Bounds, Upper_Bounds, Stack);
while(Stack $\neq \varnothing$){
eliminate_term $\leftarrow$ Stack(i); //Suppose its index is k;
$a \leftarrow a_k$;
$b \leftarrow$ coefficient of $L_k^{(2)}$;
$c \leftarrow$ coefficinet of $U_k^{(2)}$;
safe_flag $\leftarrow$ Accurate_Condition(L, U, $L_k^{(2)}$, $U_k^{(2)}$, a, b, c);
if(*safe_flag*){

```
                                        L←L+ Pos(b−a)× L_k^(2) −Neg(b−a)× U_k^(2) ;
                                        U←U+ Pos(c−a)× U_k^(2) −Neg(c−a)× L_k^(2) ;
                                        Stack←Stack−{ eliminate_term };
                                        Quadratic_Terms←Quadratic_Terms−{ eliminate_term };
                                        Lower_Bounds←Lower_Bounds−{ L_k^(2) };
                                        Upper_Bounds←Upper_Bounds−{ U_k^(2) };
                                        A←A−{a_k};
                                }
                                i←i+1;
                        }
                }

                /*step3: Linear term elimination*/
                Stack←∅ ; i←1;
                call Find_Linear_Term(L, U, Linear_Terms, Stack);
                while(Stack≠∅ ){
                        include_term←Stack(i);                  //Suppose its index is k;
                        a←c_k;
                        b←coefficient of L_k^(1) ;
                        c←coefficinet of U_k^(1) ;
                        safe_flag←accurate_condition(L, U, L_k^(1) , U_k^(1) , a, b, c);
                        if(safe_flag){
                                L←L+ Pos(b−a)× L_k^(1) −Neg(b−a)× U_k^(1) ;
                                U←U+ Pos(c−a)× U_k^(1) −Neg(c−a)× L_k^(1) ;
                                Stack←Stack−{ include_term };
                                Linear_Terms←Linear_Terms−{ include_term };
                                Lower_Bounds←Lower_Bounds−{ L_k^(1) };
                                Upper_Bounds←Upper_Bounds−{ U_k^(1) };
                                C←C− {c_k};
                        }
                        i←i+1;
        }

        /*step4: Call the I Test*/
        integer_solvable←I_Test(C, Lower_Bounds, Upper_Bounds, L, U);
        if(integer_solvable)
                return true;
        else
                return false;
}
```

```
Procedure Find_Approximate_Terms(Quadratic_Terms, Lower_Bounds, Upper_Bounds, Stack)
{
        for each element in Quadratic_Terms{
                if element ∉ Lower_Bounds && element ∉ Upper_Bounds
                        Stack←Stack + {element};
        }
}
```

```
Procedure Find_Linear_Term(L, U, Linear_Terms, Stack)
{
        for each element in Linear_Terms{
                // Suppose its index is k
                if strstr(L, element) || strstr(U, element) || !Const( L_k^(1) ) || !Const( U_k^(1) )
                        Stack←Stack + {element};
        }
}
```

Pos and Neg represent the positive and negative parts in Definition 1 of an integer, respectively. The function Const is used to determine whether the input constraint is constant. Readers can find the process of the I test from the literature [7]. Its main idea is to eliminate the variables by two basic theorems until the left-hand expression of the equation becomes a zero without checking any accuracy condition.

6 A case study

To help readers understand the algorithm, we demonstrate a case study in this section. Given the codes shown below

$$\begin{aligned}
&\text{for}(i = 1; i <= 5; i++)\\
&\quad \text{for}(j = 5 - i; j <= i + 10; j++)\\
&\qquad \text{for}(k = 1; k <= 10; k++)\\
&\mathrm{S}_1 A[i * (i - j) + i + 3 * k - 2] = \cdots\\
&\mathrm{S}_2 \cdots = A[i * i - 2 * i + 3]
\end{aligned}$$

We need to determine whether the statement S_1 depends on the statement S_2 or the contrary. According to the principle of dependence testing, we construct the dependence equation as following

$$i_1^2 - i_1 j - i_2^2 + i_1 + 2i_2 + 3k = 5$$

It subjects to the constraint defined by the loop bounds. i_1 and i_2 represent different instances of the loop iteration variable i. As a loop-carried dependence should be taken into account, we must initialize two variables for the index i here. The equation above can be rewritten as follows for the purpose of discussion

$$I_1^2 - I_1 I_3 - I_2^2 + I_1 + 2I_2 + 3I_4 = 5$$

I_1, I_2, I_3 and I_4 represent the variables i_1, i_2, j and k, respectively. We reorganize this equation by Theorem 1 as

$$3I_1^2 - I_5^2 - 2I_2^2 + I_3^2 + 2I_1 + 4I_2 + 6I_4 = 10$$

where $I_5 = I_1 + I_3$. From the definitions of interval equation, we can know that the dependence testing problem is to determine whether the interval equation

$$3I_1^2 - I_5^2 - 2I_2^2 + I_3^2 + 2I_1 + 4I_2 + 6I_4 = [10, 10]$$

subject to the region $\boldsymbol{R} =$

$$\begin{aligned}
&1 \le I_1, I_2 \le 5, 5 - I_1 \le I_3 \le I_1 + 10\\
&1 \le I_4 \le 10, 6 - I_1 \le I_5 = I_1 + I_3 \le I_1 + 15\\
&(5 - I_1)^2 \le I_3^2 \le (I_1 + 10)^2\\
&(6 - I_1)^2 \le I_5^2 \le (I_1 + 15)^2
\end{aligned}$$

is integer solvable.

It is not difficult to get the Lower_Bounds and Upper_Bounds sets from the region $\boldsymbol{R}$. Since I_1^2appears in these two sets, we know Stack $= \{I_2, I_3, I_5\}$. I_3^2 or I_5^2 should be eliminated first according to Theorem 2. We may start with I_5^2. In this case, $a = -1$, $b = c = 0$, $L(I) = U(I) = 10$, $P^2(I_5) = (6 - I_1)^2$ and $Q^2(I_5) = (I_1 + 15)^2$. Check the accuracy conditions:

Accuracy Condition 1

$$(b - a)(c - a) = (0 - (-1))(0 - (-1)) = 1 > 0$$

and

$$\begin{aligned}
&\min(U(I) - L(I) + (c - b)^+ P^2(I_5) - (c - b)^- Q^2(I_5) + 1)\\
&\quad = \min(10 - 10 + 0(6 - I_1)^2 - 0(I_1 + 15)^2 + 1)\\
&\quad \ge \min(|0 - (-1)|, |0 - (-1)|)
\end{aligned}$$

Accuracy Condition 2

$$\min(Q^2(I_5) - P^2(I_5)) = \min((I_1 + 15)^2 - (6 - I_1)^2) = \min(42I_1 + 189) \ge 0$$

Both are satisfied, so I_5^2 can be eliminated and the equation is transformed into

$$\begin{aligned}
&3I_1^2 - 2I_2^2 + I_3^2 + 2I_1 + 4I_2 + 6I_4\\
&\quad = [10 + (0 - (-1))(6 - I_1)^2, 10 + (0 - (-1))(I_1 + 15)^2]\\
&\quad = [10 + (6 - I_1)^2, 10 + (I_1 + 15)^2]\\
&\quad = [I_1^2 - 12I_1 + 46, I_1^2 + 30I_1 + 235]
\end{aligned}$$

We continue by eliminating variable I_3^2. In this case, $a = 1$, $b = c = 0$, $L(I) = I_1^2 - 12I_1 + 46$, $U(I) = I_1^2 + 30I_1 + 235$, $P^2(I_3) = (5 - I_1)^2$ and $Q^2(I_3) = (I_1 + 10)^2$. Check the accuracy conditions:

Accuracy Condition 1

$$(b - a)(c - a) \;=\; (0 - 1)(0 - 1) = 1 > 0$$

and

$$\begin{aligned}
&\min(I_1^2 + 30I_1 + 235 - (I_1^2 - 12I_1 + 46) + 0 \times (5 - I_1)^2 - 0 \times (I_1 + 10)^2 + 1)\\
&\quad \ge \min(|0 - 1|, |0 - 1|)
\end{aligned}$$

Accuracy Condition 2

$$\min(Q^2(I_3) - P^2(I_3)) = \min((I_1 + 10)^2 - (5 - I_1)^2) \geq 0$$

Both are satisfied, so I_3^2 can be eliminated and the equation is transformed into

$$\begin{aligned} &3I_1^2 - 2I_2^2 + 2I_1 + 4I_2 + 6I_4 \\ &= [10 + (6 - I_1)^2 - (0 - 1)^-(I_1 + 10)^2, 10 + (I_1 + 15)^2 - (0 - 1)^-(5 - I_1)^2] \\ &= [-32I_1 - 54, 40I_1 + 210] \end{aligned}$$

Next is I_2^2. In this case, $a = -2$, $b = c = 0$, $L(I) = -32I_1 - 54$, $U(I) = 40I_1 + 210$, $P^2(I_2) = 1$ and $Q^2(I_2) = 25$. Check the accuracy conditions:
Accuracy Condition 1 $(b - a)(c - a) = (0 - (-2))(0 - (-2)) = 4 > 0$ and

$$\begin{aligned} &\min(40I_1 + 210 - (-32I_1 + 54) + 0 \times 1 - 0 \times 25 + 1) \\ &\geq \min(|0 - (-2)|, |0 - (-2)|) \end{aligned}$$

Accuracy Condition 2

$$\min(Q^2(I_2) - P^2(I_2)) \geq \min(25 - 1) \geq 0$$

Both are satisfied, so I_2^2 can be eliminated and the equation is transformed into

$$\begin{aligned} &3I_1^2 + 2I_1 + 4I_2 + 6I_4 \\ &= [-32I_1 - 54 + (0 - (-2)) \times 1, 40I_1 + 210 + (0 - (-2)) \times 25] \\ &= [-32I_1 - 52, 40I_1 + 260] \end{aligned}$$

At this point, no variable is included in the Lower_Bounds or the Upper_Bounds set. So, the Stack is $\{I_1\}$. In this case, $a = 3, b = c = 0$, $L(I) = -32I_1 - 52$, $U(I) = 40I_1 + 260$, $P^2(I_1) = 1$ and $Q^2(I_1) = 25$. Check the accuracy conditions:
Accuracy Condition 1

$$(b - a)(c - a) = (0 - 3)(0 - 3) = 9 > 0$$

and

$$\min(40I_1 + 260 - (-32I_1 - 52) + 0 \times 1 - 0 \times 25 + 1) \geq \min(|0 - 3|, |0 - 3|)$$

Accuracy Condition 2

$$\min(Q^2(I_1) - P^2(I_1)) \geq \min(25 - 1) \geq 0$$

Both are satisfied, so I_1^2 can be eliminated and the equation is transformed into

$$\begin{aligned} & 2I_1 + 4I_2 + 6I_4 \\ & = [-32I_1 - 54 - 3 \times 25, 40I_1 + 260 - 3 \times 1] \\ & = [-32I_1 - 129, 40I_1 + 257] \end{aligned}$$

We therefore finish eliminating all quadratic polynomials from the equation. However, a linear polynomial I_1 is introduced when eliminating I_5^2 to the right-hand expression, so it should be eliminated according to the algorithm. In this case, $a = 2$, $b = -32$, $c = 40$, $L(I) = -129$, $U(I) = 257$, $P(I_1) = 1$ and $Q(I_1) = 5$.

Check the accuracy conditions:

Accuracy Condition 1

$$(b - a)(c - a) = (-32 - 2)(40 - 2) \leq 0$$

Accuracy Condition 2

$$\min(Q(I_1) - P(I_1)) \geq \min(5 - 1) \geq 0$$

Both are satisfied, so I_1 can be eliminated and the equation is transformed into

$$4I_2 + 6I_4 = [-129 - 34 \times 5, 257 + 38 \times 1] = [-299, 295]$$

The right-hand expression becomes an integer interval. According to the algorithm, the I test algorithm should be invoked to analyze this interval equation. It returns the equation is integer solvable, so there are dependences between the statements S_1 and S_2.

The above example illustrates the process of our algorithm well. However, it shows there are dependences. Consider the codes shown below

$$\begin{aligned} & \text{for}(i = 1; i <= 5; i++) \\ & \quad \text{for}(j = 1; j <= 5; j++) \\ & S_1\ A[i * (i + 2j) + j * j + 20] = \cdots \\ & S_2\ \cdots = A[i * i - 2] \end{aligned}$$

We need to determine whether the statement S_1 depends on the statement S_2 or the contrary. According to the principle of dependence testing, we construct the dependence equation as follows

$$i_1^2 + 2i_1 j + j^2 - i_2^2 = -22$$

It subjects to the constraint defined by the loop bounds. i_1 and i_2 represent different instances of the loop iteration variable i. As a loop-carried dependence should be taken

into account, we must initialize two variables for the index i here. The equation above can be rewritten as follows for the purpose of discussion

$$I_1^2 + 2I_1I_3 + I_3^2 - I_2^2 = -22$$

I_1, I_2 and I_3 represent the variables i_1, i_2 and j, respectively. We reorganize this equation by Theorem 1 as

$$I_4^2 - I_2^2 = -22$$

where $I_4 = I_1 + I_3$. From the definitions of interval equation, we can know that the dependence testing problem is to determine whether the interval equation

$$I_4^2 - I_2^2 = [-22, -22]$$

subject to the region $\boldsymbol{R} =$

$$\begin{aligned} &1 \le I_1, I_2, I_3 \le 5, 2 \le I_4 \le 10 \\ &1 \le I_1^2, I_2^2, I_3^2 \le 25 \\ &4 \le I_4^2 \le 100 \end{aligned}$$

is integer solvable.

We can solve it just like the previous example and get the final interval equation

$$0 = [-121, -1]$$

which implies no integer-valued solution is found for the equation, and there is no dependence between S_1 and S_2.

From the analysis process of the case study, we have the following conclusions. First, the PVI test loses its efficiency due to the mixed polynomials. Second, it can be known after a simple calculation that the coefficient matrix of the quadratic polynomials is not positive semi-definite for each example, so the QP test gives conservative results. In the third place, it is manifest that the equation in this case is not similar to the Eq. (2). Hence, the Quadratic test cannot determine this equation effectively. All of these results illustrate that all existing quadratic dependence testing methods are inefficient for these cases, but our approach can analyze then and give exact results.

7 Experimental evaluations

A nonlinear dependence test can be applied more widely provided it can analyze more general subscripts. To maximize parallelism in applications, a parallelizing compiler always expects to reduce conservative results of its dependence tests. As a result, dependence testing is also required to figure out the integer solutions of the system

of Eq. (1) when it confirms that the dependence equations are dependent. If a testing technique can always determine whether a dependence equation has integer solutions, it is supposed to be accurate. However, an accurate test is definitely a time-consuming technique, such as the Omega test [8]. To evaluate the performance of our algorithm, we compare our method with other nonlinear dependence testing techniques from different aspects. All the tested programs are executed on our personal computers equipped with 2.0 GHz CPU, 2 GB memory, and RedHat Linux Enterprise 5. The compiler we used in the experiment is Open64-5.0.

7.1 Efficiency and applicability

First, let us illustrate the applicability of different nonlinear dependence tests by two practical applications. Figure 1a shows the very simplified version of the nonlinear array reference codes in TRFD, which is extracted from the perfect benchmark suites. As there are irregular subscripts in this program, compilers cannot determine its dependences. However, they can be rearranged after the interprocedural analysis. An aggressive interprocedural analysis can prove that $p[i] = i(i-1)/2$, so the optimized codes are in the form of codes in Fig. 1b, in which the subscript has been a quadratic form. Figure 2 shows the kernel loop nest of OCEAN, which is also a member of the perfect benchmark suites. There are also some nonlinear subscripts. We analyze these codes with our algorithm and other nonlinear dependence tests. The results are listed in Table 2.

We can see from Table 2 that all nonlinear tests except Quadratic test can prove that the codes in TRFD are independent. The Quadratic test fails in this case since

Fig. 1 Nonlinear array references in TRFD

```
for i := 1 to n do
  for j := 1 to i do
    a[p[i]+j] := ...
```

(a) TRFD INTGRL 540

```
for i := 1 to n do
  for j := 1 to i do
    a[i(i-1)/2+j] := ...
```

(b) after interprocedural analysis

Fig. 2 Nonlinear array references in OCEAN

```
DO j1= 0,  i2k  -  1
  exj = ...
  DO jj = 0, to x(j1)
    DO mm = 0, 128
      js = 258*i2k*jj + 129*j1 + mm +1
      js2 = js + 129*i2k
      h = data(js) - data(js2)
      data(js) = data(js) + data(js2)
      data(js2) = h*exj
    END DO
  END DO
END DO
```

Table 2 Testing results of TRFD and OCEAN programs with different nonlinear tests

Dependence tests	TRFD	OCEAN	Time complexity
Quadratic test	Maybe	Maybe	Low
QP test	No	No	High
PVI test	No	Maybe	Low
Range test	No	No	Low
Our work	No	No	Low

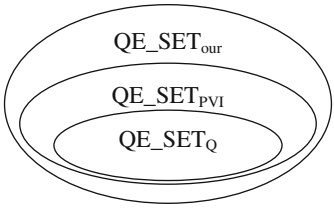

Fig. 3 Relationship between different quadratic equation sets of three nonlinear tests

QE_SET$_{our}$: quadratic equation set our method can analyze
QE_SET$_{PVI}$: quadratic equation set PVI test can analyze
QE_SET$_{Q}$: quadratic equation set Quadratic test can analyze

there are two loop indexes here. It demonstrates that our work can analyze nonlinear dependence testing problems in practice efficiently. As for OCEAN, the Quadratic test and PVI test fail to detect whether it is dependent, so they both return maybe. The subscripts in Fig. 2 do not match the form of Eq. (2), so the Quadratic test fails. Besides, there exist mixed polynomials, thereby resulting in the PVI test's failure. According to the analysis above, we construct equation sets that different nonlinear dependence tests can analyze and summarize them in Fig. 3. It illustrates that our method is able to handle more general quadratic dependence problems than the Quadratic test and PVI test.

On the contrary, our algorithm proves that the program is independent, as the QP test and Range test do. However, the QP test needs to calculate the principal minor of coefficient matrix $\boldsymbol{H}$, and solve the quadratic programming with a complicated process. As a result, it consumes much more time than the Range test and our method when analyzing OCEAN. Figure 4 shows the time comparison between these three methods. It has been normalized to the time consumed by our technique. The measurement unit of the execution time of each dependence testing technique is millisecond. It illustrates that compared with the QP test, our method has a much lower time complexity than that of the QP test.

7.2 Compared with existing nonlinear dependence tests

We can also see from Table 2 that our method performs similarly as the Range test in terms of TRFD and OCEAN. The difference between our method and the Range test can be explained as follows. Compared with the other nonlinear dependence testing methods mentioned in this paper, the Range test is definitely different. All the above-

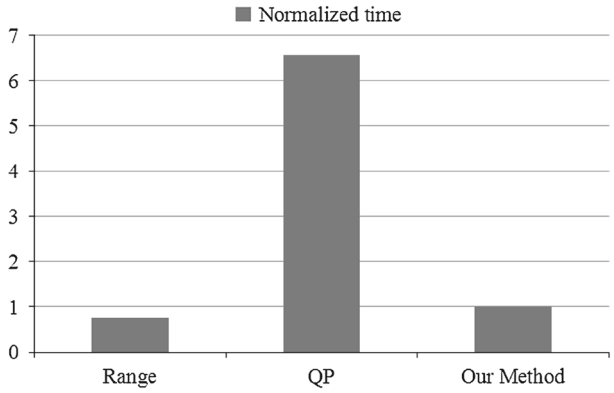

Fig. 4 Normalized time comparison between three nonlinear tests

mentioned methods determine the dependence by searching or determining whether there exist integer-valued solutions, but the Range test considers the two subscript expressions h and g of a dependence equation individually. If they are both monotonous and the maximum value of h $h_{\max}$ is less than the minimum value of g $g_{\min}$, then the analysis result will report that there is no dependence. It has no constraint on the form of the subscript expressions, but relies on their monotony. Unlike the Range test, our method determines dependences by solving the system of Eq. (1).

It is manifest that the QP test has a higher time complexity, but it is an accurate technique when it does not return maybe. As we mentioned above, the reason is the QP test that always seeks the integer solutions when it proves a dependence. Nevertheless, current parallelizing compilers usually analyze a dependence with a time saving but inefficient test first, and then use an exact but time-consuming test. The latter serves as a backup test when the former fails. This strategy tends to decrease the analysis time as well as maintain the accuracy of dependence testing. Take the Open64 compiler as an example. Its testing suite is composed of two main tests, which are GCD test [9] and Omega test, respectively. When analyzing a dependence, the compiler always detects it with the GCD test first and then turns to the Omega test when the GCD test fails. The Open64 compiler also uses some simple testing strategies for the same simplified cases, but it is unnecessary to discuss them here.

Therefore, we use the QP test as a backup accurate test. Since the Range test analyzes the two subscript expressions h and g of a dependence equation separately, its analysis results cannot be utilized by other backup tests such as the QP test. If we combine the Range and QP test into a test suite, the QP test has to construct dependence equations when it is invoked. On the contrary, the compiler is allowed to invoke the QP algorithm directly without constructing dependence equations if it uses our method instead of the Range test. Consequently, we compare our method with the Quadratic, PVI and Range tests in this subsection.

To evaluate their performance on quadratic subscripts, we select various nonlinear programs from five numerical packages [10–12]. We find 303 pairs of one-dimensional (1D) cases with constant bounds and quadratic equations, and 168 pairs of multidi-

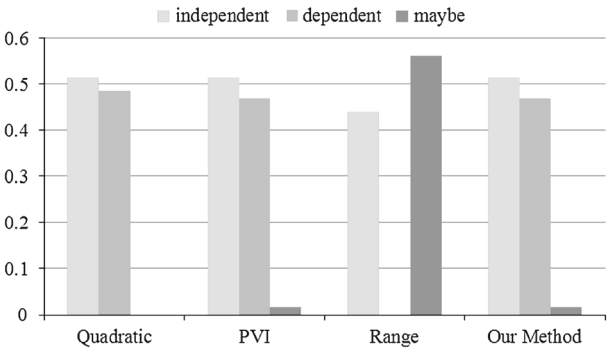

Fig. 5 Testing results of 1D cases

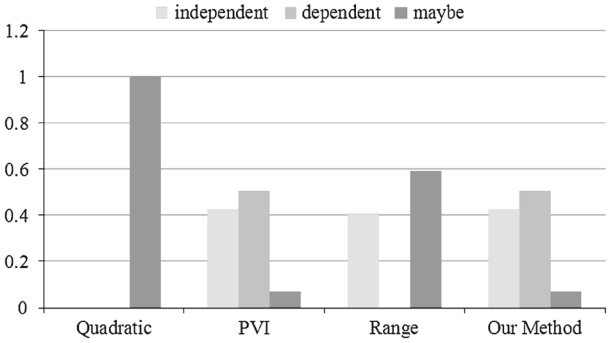

Fig. 6 Testing results of multidimensional cases without mixed polynomials

mensional cases. Here, a 1D case refers to a situation that only one loop index is allowed in an array subscript, while a multidimensional case implies that more than one loop indexes may appear. Among these multidimensional cases, 67 pairs contain mixed polynomials. We categorize these cases into three categories, which are 1D, multidimensional without mixed polynomials and multidimensional with mixed polynomials, respectively. The evaluation results are shown in Figs. 5, 6 and 7, which are normalized to the total numbers of 1D case, multidimensional case with mixed polynomials and multidimensional case without mixed polynomials, respectively.

As Fig. 5 shows, the Quadratic test does not return maybe for one-dimensional cases. In other words, for a 1D case, the Quadratic test is exact. Our method performs the same as the PVI test, since mixed polynomials will not appear in this case. Compared with the Quadratic test, our method returns some maybes in this case, since our technique as well as the PVI test need to check accuracy conditions. However, as Figs. 6 and 7 show, the Quadratic test always fails for multidimensional cases, as the dependence equations do not match the form of Eq. (2) in this case. When the tested subscripts contain no mixed polynomials, as the results shown in Fig. 6, the performance of our algorithm is still the same as that of the PVI test. However, the PVI test loses its efficiency when mixed polynomials appear, which is illustrated in Fig. 7, while our

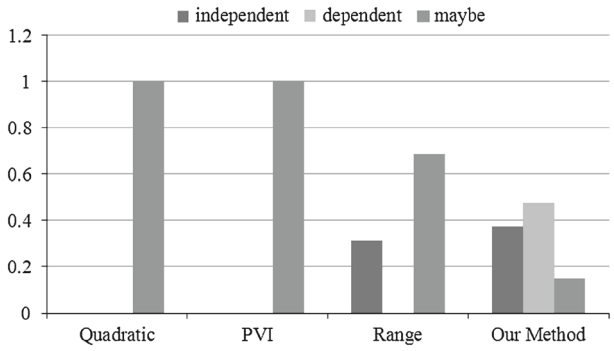

Fig. 7 Testing results of multidimensional cases with mixed polynomials

technique still works in this case, which means our technique can confirm whether there are integer solutions to the equation but will not return a conservative result.

From the evaluation results, we can see that the Range test can disprove dependences in all cases, and its performance is comparable with our method. Nonetheless, it always returns maybe when it cannot prove that there is no dependence, since it only confirms that there are solutions but cannot tell whether an integer solution exists. Our method is able to fix this problem when it is applicable, although it will not find these integer solutions.

As a consequence of the above discussions, we conclude that our algorithm can perform as well as the other three tests in one-dimensional and mixed polynomials free cases. When mixed polynomials appear, the Quadratic test and PVI test fail but our method still works.

7.3 Working together with QP test

As we described in previous subsections, the QP test is an exact dependence testing technique, but its time complexity is a boring problem. To reduce the time spent in quadratic dependence analysis, we implement our algorithm in a parallelizing compiler, and combine it with the QP test to constitute a sophistic dependence testing suite. As a result, not only can the time cost in dependence testing process be reduced, but more pseudo dependences can be eliminated, thereby exploiting more parallelism in applications. The reasons why we choose our method and the QP test have been explained above.

We implement our algorithm as well as that of the QP test on top of the Open64-5.0 compiler. During the quadratic dependence analysis process, the compiler captures necessary information about tested array subscripts, and constructs the set of quadratic dependence equations. After delivering this system of equations to the dependence testing procedure, the compiler first invokes the algorithm described in this work. If it disproves the dependences, then the process will not enter the QP test and returns the testing results. Otherwise, the compiler transfers the system of equations to the QP test. QP test will find out the integer solutions provided it proves the existence of

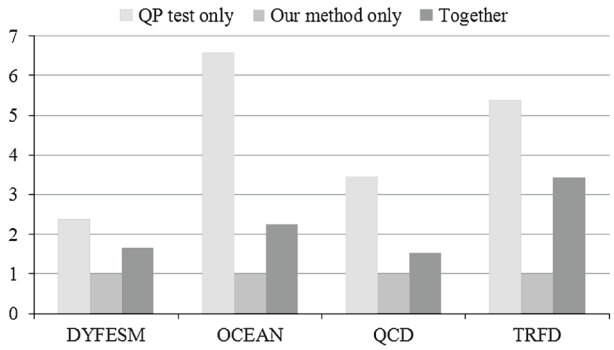

Fig. 8 Comparison of time costs between different methods

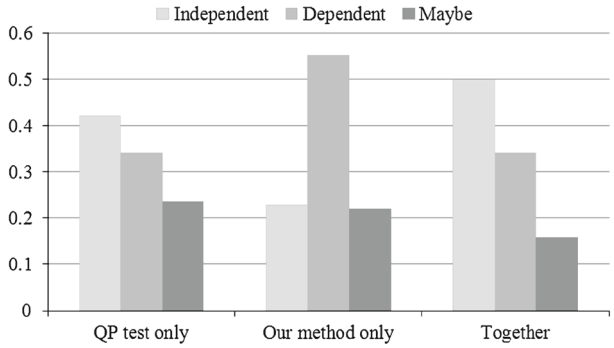

Fig. 9 Comparison of testing results between different methods

dependences, furthermore determine other dependence information, e.g., dependence directions, dependence distances, etc.

The experiment is divided into two phases. In the first phase, we detect nonlinear dependences with each test individually, and record the time costs of dependence analysis. Then, we test the same programs with two methods together, and write down the time spent in dependence testing procedure. The tested programs are extracted from the Perfect benchmark suites, which are DYFESM, OCEAN, QCD and TRFD, respectively. In these programs, numerous quadratic subscripts and other kinds of nonlinear subscripts which can be transformed into quadratic form are found. So, discussion on these programs can demonstrate the efficiency of nonlinear dependence tests. In the second phase, we record the dependence testing results when using different tests, so as to compare their accuracies. The time cost and testing performance comparisons are shown in Figs. 8 and 9, which are normalized to the time consumed by our method and the total number of tested cases, respectively.

From Fig. 8, we can see that for the tested programs in the experiment, the time cost of the QP test far outstrips those of our method and two tests together. When the compiler only uses our method, its time complexity is lower than that when the compiler uses two methods together. Figure 9 shows that compared with sole QP test, combining two tests can disprove more dependences. The reason is as follows.

When testing a quadratic dependence with QP test only, the compiler can analyze the problem only when the coefficient matrix of the problem is positive semi-definite. But if the hypothesis cannot be satisfied, it has to give a conservative result. In this case, our algorithm may still disprove dependences. Compared with using each method separately, combining two algorithms together can find more independents, while decreasing the number of maybes. That is to say, the accuracy is enhanced.

8 Theoretical analysis

The previous section verifies the technique's efficiency in an experimental way, but it can only illustrate that the methods are efficient to the data and the programs tested in the experiments. It cannot be used for a general purpose. Compared with the experimental evaluation method, the theoretical evaluation approach is more persuasive and general. Hence, we perform a theoretical analysis in this section. Since all quadratic dependence testing methods use a subscript-by-subscript approach, the efficiency of testing a single dependence equation is equal to that of testing the whole system of Eq. (1). We therefore only analyze the efficiency of testing a single dependence equation.

As described in Sect. 2, the Quadratic test and QP test are two specific methods for quadratic subscripts. The former can only deal with dependence equations written in the form of Eq. (2), while the latter must be on the basis of a positive semi-definite matrix. When the number of variables of a dependence equation is large than 2, saying that the tested subscripts include more than one loop index, the Quadratic test will lose its efficiency and give a conservative result. However, the QP test is still able to give an exact result when its hypothesis is satisfied. Hence, the QP test is more efficient than the Quadratic test in this case.

Now consider the other case. When a dependence equation is written in the form of Eq. (2), it is manifest that the coefficient matrix $\boldsymbol{H}$ of the quadratic polynomials is certainly positive semi-definite (It has been proved that the QP test has the same process with the standard form in the case of $a \leq 0$, provided $e(\boldsymbol{i}) = -f(\boldsymbol{i})$). Besides, after the extension algorithm [6], the QP test can also give as exact results as those of the Quadratic test. The QP test is therefore as accurate as the Quadratic test in this case. From the above analysis, we say that the QP test is more efficient than the Quadratic test for a general dependence equation (9).

Unlike the Quadratic test and QP test, the PVI test is not a specific dependence technique for quadratic subscripts, but a general nonlinear dependence testing approach. The PVI test can deal with the Eq. (2) for certain, and it has been proved above that the QP test is more efficient than Quadratic test. So, we compare the QP test and PVI test. As shown in Table 1, the QP test can determine a dependence equation with mixed polynomials, but its limitation lies in the positive semi-definite matrix. On the contrary, the PVI test has no restriction on the coefficient matrix, but it requires no mixed polynomials. As a result, when a mixed polynomial appears in the dependence equation, the QP test is more efficient than the PVI test.

Now consider the case without mixed polynomials. In this case, it has been proved in [5] that only when each $h_{kk} \geq 0$ $(1 \leq k \leq n)$ is satisfied, the coefficient matrix $\boldsymbol{H}$

can be positive semi-definite. The PVI test can determine whether there is an integer-valued solution to the dependence equation, although it will not search such a solution. So, we think the PVI test is more efficient than the QP test in this case. On the basis of the above discussion, we cannot say that the QP test is certainly more efficient than the PVI test or the contrary for a general dependence equation (2). It depends on whether there are mixed polynomials in the equation.

Next, let us compare our proposed method and PVI test. The common ground of these two techniques is that they are both based on the interval equation theory. They both first eliminate nonlinear variables via this theory and determine whether the equation is integer solvable. The PVI test cannot deal with mixed polynomials as described above, but our method can transform any equation into canonical form and gives an exact result. Hence, we think our method is more efficient than the PVI test in this case. When there is no mixed polynomial in a dependence equation, the PVI test needs to continue eliminating linear variables, while our method only needs to eliminate the introduced or trapezoidal linear variables and call the I test algorithm, because the equation has been a canonical form in this case. The I test will not check any accuracy conditions. So, our method is also more efficient in this case. Therefore, the proposed method in this paper is more efficient than the PVI test for the general dependence equation (9).

Now compare our method with the QP test. When the coefficient matrix $\boldsymbol{H}$ is not positive semi-definite, our algorithm can still give an exact answer but the QP test is inefficient. On the contrary, when the coefficient matrix $\boldsymbol{H}$ is positive semi-definite, the QP test can determine whether there is an integer-valued solution to the dependence equation. Our test needs to check accuracy conditions. However, as the experimental results show, QP test is much more time-consuming than our algorithm, although it is likely to be able to disprove more dependences than ours.

The difference between our work and the Range test is illustrated in Sect. 7.2, so we do not repeat it here.

Of course, our technique has its limitations. Since we focus on quadratic cases, it can only analyze quadratic subscripts compared with the PVI test and Range test. The QP test always seeks the integer solutions when it proves the dependence equations, while our technique only determines whether it has such solutions. Compared with the Quadratic test, our drawback is that we need to check accuracy conditions in 1D case. To summarize, we list both the advantages and disadvantages of these nonlinear dependence testing techniques in Table 3.

9 Related work

It has been more than 40 years since the first data dependence testing method was proposed. Earlier testing techniques determine the dependence by constructing and solving the Diophantine [13] equation of the subscript expressions. When testing multidimensional arrays with linear subscripts, a subscript position is said to be separable if its indexes do not occur in the other subscripts [14,15]. If they contain the same index, they are coupled [16]. The research community uses a subscript-by-subscript method to analyze separable subscripts.

Table 3 Comparisons between our technique with other nonlinear dependence tests

	Advantages	Disadvantages
Compared with Quadratic test	Our technique can deal with multidimensional cases	Our technique has to check accuracy conditions in 1D case
Compared with QP test	Our technique has a much lower time complexity	Our technique cannot find out the integer solutions for the equations
Compared with PVI test	Our technique can determine the quadratic equations with mixed polynomials	Our technique will lose its efficiency when confronted with a dependence equation with larger integer power variables
Compared with Range test	Our technique can be combined with other nonlinear tests easily	Our technique will lose its efficiency when confronted with a dependence equation with larger integer power variables

The GCD test [9] is a dependence testing technique using a fundamental theorem about the Diophantine equation. If the greatest common divisor of all the coefficients of loop induction variables does not divide the constant term of the equation, the dependence equation has no integer-valued solution and "no dependence" is reported. The Banerjee inequality [17] is used to determine whether the equation has real solutions in the region of interest. It first calculates the extreme values of the left-hand expression to get a real interval, and checks whether the constant on the right-hand side is in this interval. There is no dependence if this condition cannot be satisfied. Neither the GCD test's nor Banerjee Inequality's is the region of interest. The former does not consider the bound constraint, while the latter does not require an integer-valued solution. However, they indicate two specific directions for the subsequent methods. Considering the problems brought by these two methods, Kong et al. [7] proposed the I test, which combines the advantages of these two methods. Its main idea is also to eliminate variables from the dependence equation with the interval equation theory. One limitation of the I test is that the constraint of each variable has to be integer.

All these methods are subscript-by-subscript techniques. If the subscripts are coupled, the system of Eq. (1) may also be unsolvable even if each equation has an integer-valued solution. Hence, the research community developed dependence testing techniques for coupled subscripts. Knuth [18] extended the GCD test algorithm and proposed a general algorithm to solve a set of linear equations with integer coefficients. Just like the GCD test, this method does not take the bound constraint into account. The λ test [16,19] is the first technique to consider all the subscripts of coupled subscripts together. The algorithm, which combines all equations in one with some λs, can be viewed as the multidimensional form of the Banerjee Inequality. Since it introduces some new variables, the algorithm can determine the equation only when all λs can be eliminated. Focusing on the problem generated by the general GCD test, the Power test [20] was presented. When the general GCD test returns solutions to a dependence

equation, the Power test will determine whether these solutions are in the region of interest with the Fourier–Motzkin elimination (FME) [21]. The Omega test [8] is a similar approach. It also tries to address an integer programming problem by FME. As the integer programming is an NP-hard problem as everyone knows, techniques like this always trend to have a high time complexity. The Delta test [22] is an exact and efficient testing method. Its main idea is to propagate the constraints produced by some single index variable (SIV) subscripts to others in the same group without losing accuracy. It has been proved that the Delta test is just a restricted form of the λ test [1].

The above techniques can determine the dependence of linear subscripts well for most cases. So, some developers began to evaluate these tests to construct a complete dependence test suite in early 1990s. Shen et al. [23] evaluated some dependence tests and analyzed their efficiencies with an experimental method. They performed a preliminary empirical study on some numerical packages, including the Linpack and Eispack. Petersen and Padua [24] also carried out an experimental evaluation of a proposed sequence of dependence tests based on some packages, such as the Linpack, Eispack and Perfect Benchmarks. Besides evaluating the accuracy, Psarris and Kiriakopoulos [25] also showed tradeoffs between accuracy and efficiency and the time costs of three representative techniques. All these methods tried to give experimental or empirical results so as to enhance the ability of compilers. Directed by this issue, some newer proposed data dependence tests combined a suite of testing methods for different cases. Golf et al. [22] divided the subscripts into three categories, which are zero index variable (ZIV), SIV and multiple index variable (MIV), respectively. They also presented a set of testing methods to deal with these different cases. Maydan et al. [26] also proposed a subscript pattern-based dependence test suite. They adapted the FME as an expensive back up test when all the primary methods failed. All the above tests are designed based on an assumption that the subscripts are linear affine expressions of the loop index variables, while the algorithm proposed in this paper, as well as the Quadratic test [3], QP test [5,6], PVI test [4] and Range test [2] are nonlinear dependence tests.

In addition to these, there are some other specific data dependence analysis techniques. Hommel [27] proposed a general data dependence test for dynamic, pointer-based data structures based on a logic proof idea. It can also be viewed as a nonlinear test. Paek et al. [28,29] categorized the nonlinear subscripts, and proposed an array access pattern LMAD. A subscript is represented as a general expression, and an ART test [30,31] was described to address the dependence problem based on it. Engelen et al. [32] presented a unified approach for nonlinear dependence testing. They proposed Nonlinear GCD test, Nonlinear Value Range test and Nonlinear EVT (Extreme Value test) for different cases, respectively.

The literature [33] proposed a knowledge-based learning system K test. It integrates existing tests and makes good use of their advantages. An single instruction multiple data (SIMD) faced dependence test, D test, was proposed in [34]. It extended the Banerjee Inequality and proved some theorems for the case where the dependence distance is greater than or equal to the number of data processed in the SIMD register.

With the development of speculative parallelizing compilers, the research community began to focus on the study on data dependence profiling, which provides

runtime dependence information in speculative optimizations. Chen et al. [35] proposed a hash-based rather than pair-wise data dependence profiling technique, so as to perform the compiler-based instrumentation. To take full advantages of multi-core architectures, shadow profiling [36] was introduced to perform sampling on long traces of instrumented codes. Other representative prior data dependence profilers can mainly categorized into pair-wise [37,38], which tries to track all pairs of dependences that occur at runtime, and software signature-based [39] determining the dependence by grouping many pair-wise relationships into a single set operation.

10 Conclusions

Conventional linear data dependence tests are not able to satisfy their demands due to the existence of irregular and nonlinear subscripts for the scientific and engineering applications. To address this problem, the research community proposed some nonlinear tests. The QP test and PVI test are two representative methods among these tests. However, both of them have their own limitations when applied in practice. This paper proposed an improved nonlinear data dependence test, which can deal with the cases where the PVI test fails, and it can reduce the time cost of dependence testing when combined with QP test. Experimental results show that compared with existing quadratic dependence analysis techniques, the method proposed in this paper either has a wide range of application or has a much lower time complexity. As we design this technique for quadratic subscripts, our next research plan is to extend the algorithm for more complicated nonlinear cases.

Acknowledgments We would like to acknowledge the anonymous referees for their invaluable comments and suggestions on this paper. Our work is supported by the HEGAOJI Major Project of China under Grant No. 2009ZX01036-001-001-2 and the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing No. 2013A11.

References

1. Allen R, Kennedy K (2001) Optimizing compilers for modern architectures: a dependence-based approach. Morgan Kaufmann Publisher, San Francisco
2. Blume W, Eigenmann R (1998) Nonlinear and symbolic data dependence testing. IEEE Trans Parallel Distrib Syst 9(12):1180–1194
3. Wu J-H, Chu C-P (2007) An exact data dependence test for quadratic expressions. Inf Sci 177(23):5316–5328
4. Zhou J, Zeng GH (2008) A general data dependence analysis for parallelizing compilers. J Supercomput 45(2):236–252
5. Zhao J, Zhao RC, Han L (2012) A nonlinear array subscripts dependence test. In: Proceedings of the 14th international conference on high performance computing and communication, pp 764–771, June 2012
6. Zhao J, Zhao RC, Han L, Xu JL (2013) QP test: a dependence test for quadratic array subscripts. IET Softw 7(5):271–282
7. Kong X, Klappholz D, Psarris K (1991) The I test: an improved dependence test for automatic parallelization and vectorization. IEEE Trans Parallel Distrib Syst 2(3):342–349
8. Pugh W (1991) The Omega test: a fast and practical integer programming algorithm for dependence analysis. In: Proceedings of the 1991 ACM/IEEE conference on supercomputing, pp 4–13, June 1991

9. Wolfe MJ (1995) High performance compilers for parallel computing. Addison-Wesley Press, Redwood City
10. Blume W, Eigenmann R (1992) Performance analysis of parallelizing compilers on the perfect benchmarks program. IEEE Trans Parallel Distrib Syst 3(6):643–656
11. Dongar J, Furtney M, Reinhardt S (1991) Parallel loops: a test suite for parallel compilers: description and example results. Parallel Comput 17(10–11):1247–1255
12. Smith BT, Boyle JM, Dongarra JJ, Garbow BS, Ikebe Y, Klema VC, Moler CB (1976) Matrix eigensystem routines-eispack guide, 1st edn. Springer, New York
13. Shen ZY, Li ZY, Yew PC (1989) An empirical on array subscripts and data dependencies. In: Proceedings of the international conference on parallel processing, pp 145–152, August 1989
14. Allen R (1983) Dependence analysis for subscripted variables and its application to program transformations. Ph.D. thesis. Department of Mathematical Sciences, Rice University
15. Callahan D (1986) Dependence testing in PFC: Weak separability. Supercomputer Software Newsletter 2, Department of Computer Science, Rice University, August 1986
16. Li ZY, Yew PC, Zhu CQ (1990) An efficient data dependence analysis for parallelizing compilers. IEEE Trans Parallel Distrib Syst 1(1):26–34
17. Banerjee U, Eigenmann R, Nicolau A, Padua DA (1993) Automatic program parallelization. Proc IEEE 81(2):211–243
18. Knuth DE (1987) The art of computer programming. Seminumerical algorithms, vol 2, 3rd edn. Addison-Wesley, Reading
19. Li ZY, Yew PC, Zhu CQ (1989) Data dependence analysis on multi-dimensional array references. In: Proceedings of the 3rd international conference on supercomputing, pp 215–224, June 1989
20. Wolfe M, Tseng CW (1992) The Power test for data dependence. IEEE Trans Parallel Distrib Syst 3(5):591–601
21. Williams HP (1976) Fourier–Motzkin elimination extension to integer programming problems. J Comb Theory (A) 21(1):118–123
22. Goff G, Kennedy K, Tseng CW (1991) Practical dependence testing. In: Proceedings of the ACM SIGPLAN 1991 conference on programming language design and implementation, pp 15–29, June 1991
23. Shen ZY, Li ZY, Yew PC (1990) An empirical study of Fortran programs for parallelizing compilers. IEEE Trans Parallel Distrib Syst 11(3):356–364
24. Petersen P, Padua D (1996) Static and dynamic evaluation of data dependence analysis techniques. IEEE Trans Parallel Distrib Syst 7(11):1121–1132
25. Psarrisand K, Kyriakopoulos K (2004) An experimental evaluation of data dependence analysis techniques. IEEE Trans Parallel Distrib Syst 15(3):196–213
26. Maydan DE, Hennessy JL, Lam MS (1991) Efficient and exact data dependence analysis. In: Proceedings of the ACM SIGPLAN 1991 conference on programming language design and implementation, pp 1–14, June 1991
27. Hummel J, Hendren LJ, Nicolau A (1994) A general data dependence test for dynamic, pointer-based data structures. In: Proceedings of the ACM SIGPLAN 1994 conference on programming language design and implementation, pp 218–229, June 1994
28. Paek Y, Hoeflinger J, Padua DA (1998) Simplification of array access patterns for compiler optimizations. In: Proceedings of the 19th ACM SIGPLAN conference on programming language design and implementation, pp 60–71, June 1998
29. Paek Y, Hoeflinger J, Padua DA (2002) Efficient and precise array access analysis. ACM Trans Program Lang Syst 24(1):65–109
30. Hoeflinger J (2000) Interprocedural parallelization using memory classification analysis. Ph.D. thesis, University of Illinois at Urbana-Champaign, Department of Computer Science
31. Hoeflinger J, Paek Y (1999) The access region test. In: Proceedings of the workshop on languages and compilers for parallel computing, pp 271–285
32. van Engelen RA, Birch J, Shou Y, Gallivan KA (2004) A unified framework for nonlinear dependence testing and symbolic analysis. In: Proceedings of the international conference on supercomputing, pp 106–115
33. Yang CT, Tseng SS, Shin WC (2000) The K test: an exact and efficient knowledge-based data dependence testing method for parallelizing compilers. Proc Natl Sci Counc 24(5):362–372

34. Bulic P, Gustin V (2004) D-test: an extension to Banerjee test for a fast dependence analysis in a multimedia vectorizing compiler. In: Proceedings of the 18th international parallel and distributed processing symposium, pp 535–546
35. Chen T, Lin J, Dai XR, Hsu WC, Yew PC (2004) Data dependence profiling for speculative optimizations. In: Proceedings of the international conference on compiler construction, pp 57–72
36. Moseley T, Shye A, Reddi VJ, Grunwald D, Peri R (2007) Shadow profiling hiding instrumentation costs with parallelism. In: Proceedings of the IEEE/ACM international symposium on code generation and optimization, pp 198–208
37. Yu HT, Li ZY (2012) Fast loop-level data dependence profiling. In: Proceedings of the international conference on supercomputing, pp 37–46
38. Kim MJ, Kim H, Luk CK (2010) SD^3: A scalable approach to dynamic data-dependence profiling. In: Proceedings of the IEEE/ACM international symposium on microarchitecture, pp 535–546
39. Vanka R, Tuck J (2012) Efficient and accurate data dependence profiling using software signatures. In: Proceedings of the IEEE/ACM international symposium on code generation and optimization, pp 168–195