Survey of Different Data Dependence Analysis Techniques

Monali Patil, Vandana Jagtap

Computer Department MAEER's MIT Pune, India

Abstract**—***Dependency analysis is a technique to detect dependencies between tasks that prevent these tasks from running in parallel. It is an important aspect of parallel programming tools. Dependency analysis techniques are used to determine how much of the code is parallelizable.*
Literature shows that number of data dependence test has been proposed for parallelizing loops in case of arrays with linear subscripts, however less work has been done for arrays with nonlinear subscripts. GCD test, Banerjee method, Omega test, I-test dependence decision algorithms are used for one-dimensional arrays under constant or variable bounds. However, these approaches perform well only for nested loop with linear array subscripts. The Quadratic programming (QP) test, polynomial variable interval (PVI) test, Range test are typical techniques for nonlinear subscripts. The paper presents survey of these different data dependence analysis tests.

Keywords— Dependence analysis, Parallelization, Data dependence test, Nonlinear subscript, Variable bound.

I. INTRODUCTION

The area of dependency analysis has served as grounds for fruitful research as well as practical implementation. Program dependency analysis is a technique to analyze programs and determine potential flow of data between program statements. It represents required arrangement of statements that need to be preserved in a program. Data dependence occurs when two statements in a program are accessing the same shared memory location one of them performs write operation on the shared memory.
Data dependency is one of the major hindrances in the area of high performance computing which aims at delivering higher performance for solving large problems. While implementing an algorithm presence of data dependence requires the statements of an algorithm to access shared memory location. Further dependence implies further access time or more inter process communication and increases complexity of an algorithm. This accordingly degrades the real-time performance of system. Parallelization refers to converting the sequential code to multi-threaded code to utilize available computing power. Program parallelization has become mainstream research topic due to the invention of multi core processors. Although multi-core processors could provide high processing speed, but legacy applications could not utilize it as most of them were written in serial fashion. Thus, focus of research shifted to development of tools that could detect part of a code that can be performed concurrently in a program. The approach to parallelism is based on the learning of data dependencies as dependence prevents two computations from executing in parallel. In general, fewer dependencies, greater the parallelism [1]. Therefore, it is essential to study and analyse dependencies in a program.
In section II we discuss the basic data dependence problem and how dependence tests solve this problem. In section III we present survey of available dependence techniques which includes GCD, Banerjee, Omega, I, Range, PVI and Quadratic test. Section IV demonstrates the review of these tests in tabular format. Finally, in section V, we present our conclusion.

II. DATA DEPENDENCE PROBLEM

Two statements are said to be data dependent if; one statement follows the other, they share a memory location and one of them writes to it. Based on the order in which these statements access the shared location classifies dependency into following classes:

i. True Dependence (read-after-write): statement 1 writes to the memory location while statement 2 reads from it.
ii. Anti Dependence (write-after-read): statement 1 reads from the memory location while statement 2 writes to that location.
iii. Output Dependence (write-after-write): both statement 1 and 2 writes to the same memory location.

These memory accesses in sequential languages are performed either with scalar variables, array variables or pointer references [2]. However dependence problem for scalar variables can be solved with techniques such as induction variable substitution, constant propagation etc, and dependence problem for pointers constitutes an altogether separate research area as they deal with dynamic memory accesses which are difficult to handle. So this study particularly focuses on data dependence problem for array references inside loop nests.

Inside loop, each statement can be executed several times. Loop independent dependence is when dependence relation between two statement instances is executed in the same iteration of a loop.

Example of loop independent dependence,

```
for(int j=0; j<100; j++) {
    a[j]=a[j-1]
}
```

Loop carried dependence occurs when data dependence relation between two statement instances is executed in two different iterations i.e. source and sink occurs on different iterations.

Example of loop carried dependence,

```
 for(int j=0; j<100; j++) {
    a[j]=a[j-1]
}
```

To convert sequential loop into parallel loop, it should not contain any loop carried dependences.

The concept of dependence is defined with information about the respective iterations in which the dependent instances occur [2]. This information is expressed in terms of Distance vector and Direction vector.

i. Distance Vector -If two statements share n common loops then distance vector computes subtraction of vectors representing these common loops for the two statements. Most loop transformation including parallelization and vectorization requires only sign of the elements in the distance vector. The sign of distance vector denotes the direction of dependence in a program.

Distance vector δ=
I(vector representing - I'(vector representing
Common
loops for common loops for
Statement 1)
statement 2)

ii. Direction Vector- When testing for data dependence, direction vector for all dependent instances are calculated. A vector of the form ($d_1, d_2, d_3, \ldots\ldots, d_n$) where d_k Є (<, >, =, *), 1<k<n, is termed as a direction vector. A data dependence exists between statements with direction vector d_k such that,

$$d_k = \begin{cases} < & \text{if } I_k < I'_k \\ = & \text{if } I_k = I'_k \\ > & \text{if } I_k > I'_k \end{cases}$$

Basic dependence problem is to determine if two indexed elements of an array would represent the same memory location under certain given conditions. When a loop has no dependence then it can be executed in parallel otherwise the loop must be run in sequential order. To solve the problem of data dependence inside loops means to solve a system of linear Diophantine equations subject to a set of constraints which may take different forms based on the program and characteristics of dependence under consideration. If there does not exist any solution to these equations then there is no dependence otherwise dependence exists between statements in a loop. However, there exist two major types of dependence tests i.e. exact test and inexact (approximate) tests. In inexact test, there is no dependence when an integer solution to the system of equations is not available but they assume dependence when the outcome is not known.

III. DATA DEPENDENCE TESTS

It has been more than 40 years since the first data dependence test was proposed [3]. Data dependence techniques for linear subscripts were well developed. Conventional approaches perform well only for nested loop with linear array subscripts and in each test there is tradeoff between accuracy and efficiency. Compilers use dependence testing suite for analyzing dependencies in a program. Current parallelizing compilers generally analyze dependence with a time saving but inefficient test first and then use an exact but time-consuming test. The exact test works as a backup test when the inexact test fails. This leads to decrease in analysis time and also maintains accuracy of dependence testing. Example, In Open64 compiler, testing suite is composed of GCD and Omega tests.

i. The GCD Test

The GCD test is arguably the most basic of the dependence tests and is often incorporated into other dependence tests as an initial screen for dependence [4]. GCD test aims at finding integer solution to the equation.

The GCD test simply checks for the integer divisibility of the linear equation i.e. a linear equation has an integer solution if and only if the greatest common divisor of the coefficients on the left-hand-side (LHS) of equation uniformly divides the constant term on the right-hand-side (RHS) of the equation. If this condition does not hold then there is no solution possible to the equation and thus no dependence exists.

However, if condition does apply, then it is not necessary that dependence exists, here; the test returns a maybe answer. This implies that GCD test is an inexact test. As it is a basic test for dependency analysis thus has many limitations. It is incapable of proving dependencies, can only disprove them. The GCD test is necessary but not sufficient condition for data dependence.

ii. The Banerjee Test

The Banerjee test is based on the intermediate value theorem [4]. The test computes minimum and maximum values an expression on left-hand-side of linear equation can achieve, under constraints on variables involved in the expression. Once the minimum and maximum values of an expression are known then the test checks whether constant term on the right-hand-side lies between these extreme values. If the constant does not lie within the range, then no dependence exists. If it does, then real solution to the linear equation exists. However, it is not necessary that dependence exists since there may not be, in fact, an integer solution to the equation. This inability of Banerjee test to differentiate between real and integer solutions makes it an inexact test.

Banerjee test provides the direction vector information for the dependent instances. The GCD test and Banerjee test are very simple and efficient at disproving dependencies so they are widely used in compilers.

iii. The I-test

I-test is a conventional dependence analysis technique designed for linear subscripts. The I-test is based on and enhances the Banerjee test [5]. It assumes all the advantages of GCD test and Banerjee test and was proposed considering problems brought by these two tests. Banerjee test is unable to differentiate between real and integer solution to the linear equation, while I-test can positively prove or disprove existence of integer solution in most cases.

The I-test uses the concept of integer interval equation, $a_1x_1 + a_2x_2 +.....+ a_nx_n = [L, U]$. All the ordinary linear equations are denoted with integer interval equation with constant term on the right-hand-side between L and U bounds. The above interval equation has an integer solution if and only if at least one of the equations in the set has an integer solution, subject to constraints i.e. there exists a solution if the value realized by the left-hand-side expression is between L and U. The I-test is an exact test but fails to solve problem with non-linear subscripts.

iv. The Omega Test

The Omega test is based on the Fourier-Motzkin variable elimination (FMVE) algorithm and its extensions. FMVE is a mathematical algorithm for eliminating variables from a system of inequalities.

Omega test takes set of constraints as an input. These constraints can be in the form of equality or inequality conditions obtained from the subscript expression, iteration index bounds or if-statements. Then Omega test performs series of variable elimination operations to minimize the problem into smaller equivalent problems which can be solved recursively. First, all the input constraints are normalized by dividing coefficients by Greatest common divisor of each constraint. In the next step the test eliminates all equations such as by replacing two inequalities by an equation. This process continues repeatedly until a coefficient having absolute value of one is found and the system can be reduced by one equation. The Omega test then performs a Fourier-Motzkin projection of the problem constraints which is a very expensive step [6]. This Fourier-Motzkin projection is performed on dimension of variable that is eliminating.

In data dependence problem, the Omega test can be applied to a lot of cases. The Omega test can be used for array references with coupled subscripts which introduce equations in the dependence system that share common variables. The Omega test can handle symbolic variable, that appear in subscript expression and which do not have numerical value. Data dependence in this case is tested by determining values for loop index variables and symbolic variable satisfying constraints of the problem. The Omega test can also handle triangular, trapezoidal bounds and nested if-statements. This makes Omega test a very powerful test but the cost of Omega test is high. The test has very costly initialization and needs a lot of memory to hold all these sub-problems.

v. The Range Test

Blume and Eigenmann proposed the Range test. The Range test is mainly used to check whether a dependence relation exists for nonlinear expressions [7]. The expression that cannot be written in $a_1x_1 + a_2x_2 +.....+ a_nx_n+a_0$ format are called as nonlinear expressions where a_0, a_k are integer constants and x_k is loop index variable for $1<k<n$.

The Range test is based on the extreme value computation just like Banerjee test. It works as, for a given iteration of a loop, the accessed array subscript range is considered a symbolic expression; and if this range does not overlap with the range accessed in the next iteration, then no dependence exists between the iterations of a loop. The Range test is a subscript-by-subscript test and does not consider the rest of the array subscripts while testing for data dependence.

vi. The PVI Test

The PVI polynomial variable interval is a dependence test for nonlinear subscripts. It can accurately handle couple subscript polynomial expressions and trapezoidal bounds [8].

The PVT test extends the I-test which fails to solve the dependence problem for nonlinear subscripts. It is based on interval solution theories for polynomial equation and works by repeatedly eliminating variables from the polynomial interval equation. The PVI test consists of three steps, the first step is to rewrite the Diophantine equations into the form of polynomial interval equation subjected to inequalities. In the next step, it repeatedly eliminates variables which do not appear in any constraints in the equation. In the last step, it checks whether dependence exists. The 2nd step results in an integer interval equation with zero on the left-hand-side and integer interval on the right-hand-side. If zero belongs to the integer interval on the right-hand-side then there exists solution to the system of equations and thus dependence exists.

vii. The Quadratic test

The quadratic test is based on subscript-by subscript idea which solves one equation at a time other than solving the whole system.

Quadratic test is defined for cases when the dependence equation is quadratic and only one loop index appears in the subscript [9]. In case of loop carried dependence, there will be two variables x and z in the equation which are different instances of the same loop index. The algorithm iterates one instance variable with other in order to narrow their value intervals. When either of their intervals becomes null, then iteration process stops and thus no dependence exists. Otherwise their intervals are reduced up to one point and thereby resulting into the solution for the equation. For quadratic test dependence equation need to be in the following form,

$$Ax^2 + bz = C$$

Where x and z are different instances of same index.

The drawback of quadratic test is that it allows only one loop iteration variable. To overcome the limitations of quadratic test, quadratic programming (QP) test was proposed [3]. Quadratic programming test treats the left-hand-side expression of the dependence equation as objective function and computes its extreme values. If the minimum value is greater than zero, then there is no dependence. When it is not greater than zero then the algorithm looks for a solution making the objective function equal to zero by branch and bound method. If this is not an integer solution, there is no dependence.

IV. REVIEW OF DEPENDENCE TESTS

Paper	Year	Method Used	Advantages	Limitations
Dependence Analysis For Super-computing	1988	GCD Test	-Simple test -efficient at disproving dependencies.	-Inexact test -ignores loop limit and inequality constraints - does not provide direction vector information.
Automatic Program Parallelization	1993	Banerjee Test	-simple test -efficient at disproving dependencies -generates direction vector information.	-Inexact test -considers single subscript of a multi-dimensional array.
The Omega Test: a fast and practical integer Programming algorithm for dependence analysis	1992	Omega Test	-Exact test -fast and practical -generates distance and direction vector information.	-Expensive test.
The I test: an improved dependence test for automatic parallelization and vectorization.	1991	I-test	-inherits all the benefits of GCD test and Banerjee test -extends the range of applicability of Banerjee test.	-constraint of each variable has to be integer -cannot handle multi-dimensional array references involving coupled subscripts

Nonlinear and symbolic data dependence testing	1998	Range test	-applicable for nonlinear expressions.	-subscript by subscript test.
A general data dependence analysis for parallelizing compilers	2008	PVI test	-applicable for nonlinear subscripts	-lose efficiency when mixed polynomial exists in dependence equation.
An improved nonlinear data dependence test	2014	Quadra-tic progra mming test	-Exact test -works efficiently for mixed polynomials.	-time complexity high -coefficient matrix of quadratic terms should be positive semi-definite.

V. CONCLUSIONS

If done correctly, dependence analysis is of immense benefit. Data dependence information is essential to detecting loop iterations that can be executed in parallel on multiprocessor. Advances in data dependence analysis have improved dependence accuracy, but without much success in increasing program parallelization.
Literature shows that many dependence tests have been proposed. Each dependency test is designed for a specific type of data reference found in the loops. Conventional linear data dependence tests are not able to satisfy their demands due to presence of irregular and non-linear subscripts for some applications. To address this problem, some non-linear tests are proposed.

ACKNOWLEDGEMENT

We would like to take this opportunity to thank people who have been constantly supporting and encouraging us in matters, big or small. Firstly we would like to thank our Professors for their valuable guidance. Furthermore, we would like to thank anonymous referees for helping immensely to improve this paper.

REFERENCES

[1] M.A. Hossaina, U. Kabirb, M.O. Tokhi, “Impact of data dependencies in real-time high performance computing,” Elsevier Science B.V. 2002.
[2] Konstantinos Kyriakopoulos, and Kleanthis Psarris “ Data Dependence Analysis Techniques for Increased Accuracy and Extracted Parallelism,” International Journal of Parallel Programming, Vol. 32, No. 4, August 2004.
[3] Jie Zhao , Rongcai Zhao, Xi Chen, Bo Zhao ,“An improved nonlinear data dependence test,” Springer Science+Business Media New York 2014.
[4] U. Banerjee,” Dependence Analysis for Supercomputing,” Boston:Kluwer Academic, 1988.
[5] Xiangyun Kang, David Klappholz and Kleanthis Psarris,“The I test:an improved dependence test for automatic parallelization and vectorization,” IEEE transactions on parallel and distributed systems VOL. 2, NO. 3, July 1991.
[6] William Pugh ,“The Omega Test: a fast and practical integer programming algorithm for dependence analysis,” ACM, August 1992.
[7] Blume W., Eigenmann R.,”Nonlinear and symbolic data dependence testing,” Parallel and distributed systems IEEE, Dec 1998.
[8] Jing Zhou · Guosun Zeng, “A general data dependence analysis for parallelizing compilers,” Springer Science+Business Media, LLC 2008
[9] Jia-Hwa Wu , Chih-Ping Chu, “An exact data dependence testing method for quadratic expressions,” Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, June 2007.
[10] Kleanthis Psarris, Santosh Pande, “Classical Dependence Analysis Techniques:Sufficiently Accurate in Practice”, IEEE 28th Hawaii International Conference on System Sciences, 1995.
[11] Kleanthis Psarris, Konstantinos Kyriakopoulos “An Experimental Evaluation of Data Dependence Analysis Techniques,” IEEE transactions on parallel and distributed systems, VOL. 15, NO. 3, MARCH 2004.
[12] Jia−Hwa Wu and Chih−Ping Chu, “The Quadratic Test: An Exact Data Dependence Test for Quadratic Expressions,” Department of Computer Science and Information Engineering National Cheng Kung University, Tainan, Taiwan 701, R.O.C.
[13] Paul M. Petersen and David A. Padua,“Static and Dynamic Evaluation of Data Dependence Analysis Techniques,” IEEE transactions on parallel and

distributed systems, VOL. 7, NO. 11, November 1996.
[14] Banerjee U, Eigenmann R, Nicolau A, Padua DA," Automatic program parallelization," IEEE 1993.