Coping with Data Dependencies of Multi-Dimensional Array References

Lin Qiao, Weitong Huang, Zhizhong Tang

Department of Computer Science and Technology, Tsinghua University,
Beijing, 100084, PR China
{qiaolin, hwt}@cic.tsinghua.edu.cn; tzz-dcs@tsinghua.edu.cn

Abstract. This paper presents a new static data dependence analysis approach, Dependence Difference Inequality Test, which can deal with coupled subscripts for multi-dimensional array references for software pipelining techniques for nested loops. The Dependence Difference Inequality Test (DDIT) replaces direction vectors with dependence difference inequalities as constraints to variables in a linear system. The method presented in this paper extends the applicable range of the Generalized Lambda Test and seems to be a practical scheme to analyze data dependence. Experimental results show that the number of data independences checked by the DDIT algorithm is slightly smaller than that manually. It is also shown that our method is better than other traditional data dependence analysis methods without increasing time cost: it increases the success rate of the Generalized Lambda Test by approximately 14.19%.

1 Introduction

Data dependence analysis plays an important role in automatic detection of implicit parallelism in programs written in conventional sequential languages. Dependence analysis techniques estimate, at compile-time, the run-time interactions between different operations or between different instances of the same operation [1]. It is at the core of data dependence analysis strategies to estimate data dependence between two operations in which multi-dimensional array references are involved [2].

Mathematically the problem can be reduced to that of checking whether or not a system of m linear equations with $2n$ unknown variables has a simultaneous integer solution, which satisfies the constraints for each variable in the system. It has been proved that a loop can be software-pipelined with any value of initiation interval if the dependence difference inequalities do not satisfy simultaneously [3]. That is to say, dependence difference inequalities can act as additional constraints to each variable in the system of m linear equations on their own or with other constraints, such as direction vectors.

The paper is the ongoing work of [3]. We focuses on applying dependence difference inequalities for analyzing data dependence. The algorithm the paper presents, called Dependence Difference Inequality Test (DDIT), can handle coupled subscripts for multi-dimensional array references statically. Experimental results shows that our method is better than the Generalized Lambda Test.

This paper is organized as follows. Section 2 first introduces the Lambda Test and the Generalized Lambda Test, and then Section 3 discusses the Dependence Difference Inequality Test. Section 4 gives experimental results. The Last section draws a conclusion.

2 The Lambda Test and the Generalized Lambda Test

This section introduces the Lambda Test and the Generalized Lambda Test that are the cornerstone of the DDIT algorithm.

Geometrically, a linear equation in the system defines a hyperplane π in $\mathbf{R}^{2n}$ spaces. The intersection S of m hyperplanes corresponds to the common solutions to all linear equations in the system [2] [4]. It is obvious that there exists no data dependence if S is empty. The bounds introduced by the Lambda Test or by the Generalized Lambda Test, with any given direction vectors, define a bounded convex set V in $\mathbf{R}^{2n}$. If any of hyperplanes in the system does not intersect V, it is clear that S can not intersect V. However, even if every hyperplane intersects V, it is still possible that S and V are disjoint. And if S and V are disjoint, there exists a hyperplane which contains S and is disjoint from V. Furthermore, the hyperplane is a linear combination of hyperplanes in the system. On the other hand, if S intersect V, there is no such a linear combination [2].

In summary, the Lambda Test or the Generalized Lambda Test first applies the Banerjee Inequalities to test each hyperplane in the system, and then checks these hyperplanes simultaneously if every hyperplane intersects V. These two methods are efficient and precise to analyze the system beneath V. In fact, they are equivalent to a multi-dimensional version of the Banerjee Inequality because they can determine simultaneous constrained real-valued solutions. The tests form linear combinations of coupled references that eliminate one or more instances of index variables when direction vectors are not considered. On the other hand, once direction vectors are considered, they can generate new linear combinations that use a pair of relative index variables. Simultaneous constrained real-valued solutions exist if and only if the Banerjee Inequalities find solutions in all linear combinations generated [2].

3 Dependence Difference Inequality Test

This section takes account of m linear equations, $m \geq 2$, and gives a detailed description of the Dependence Difference Inequality Test.

Without losing generality, all m linear equations are assumed to be connected; otherwise they be partitioned into smaller systems. Furthermore, it is hypothesized that there are no redundant equations. An arbitrary linear combination of m linear equations can be written as $\sum_{i=1}^{m} \lambda_i F_i = 0$ where $F_i = a_{i,0} + \boldsymbol{a}_i\boldsymbol{x}$, $\boldsymbol{a}_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,2n})$ for $1 \leq i \leq m$, and $\boldsymbol{x} = (x_1, x_2, \ldots, x_{2n})^{\mathrm{T}}$. The domain of $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ is the whole $\mathbf{R}^m$ space. Let $F_\lambda = -\boldsymbol{\lambda b} + (\boldsymbol{\lambda a}_1', \boldsymbol{\lambda a}_2', \ldots, \boldsymbol{\lambda a}_{2n}')\boldsymbol{x}$ where $\boldsymbol{b} = (-a_{1,0}, -a_{2,0}, \ldots, -a_{m,0})^{\mathrm{T}}$

and $\boldsymbol{a}_k' = (a_{1,k}, a_{2,k}, \ldots, a_{m,k})^{\mathrm{T}}$ for $1 \le k \le 2n$. It needs to be determined whether or not $F_\lambda = 0$ intersects V in $\mathbf{R}^{2n}$ space for arbitrary $\boldsymbol{\lambda}$.

Definition 1. The coefficient of each variable in F_λ is a linear function of $\boldsymbol{\lambda}$ in $\mathbf{R}^m$ which is $\psi_k = \boldsymbol{\lambda}\boldsymbol{a}_k'$ for $1 \le k \le 2n$. The equation $\psi_k = 0$ is termed a ψ-equation, which corresponds to a hyperplane in $\mathbf{R}^m$, called a ψ-plane. Let $\phi_k = \psi_{2k-1} + \psi_{2k}$ for $1 \le k \le n$. The equation $\phi_k = 0$ is called a ϕ-equation, which still corresponds to a hyperplane in $\mathbf{R}^m$, called a ϕ-plane.

In general, each ψ-plane or ϕ-plane divides $\mathbf{R}^m$ into two closed half-spaces, i.e., $\psi_k^+ = \{\boldsymbol{\lambda} | \psi_k \ge 0\}$, $\psi_k^- = \{\boldsymbol{\lambda} | \psi_k \le 0\}$, $\phi_k^+ = \{\boldsymbol{\lambda} | \phi_k \ge 0\}$, and $\phi_k^- = \{\boldsymbol{\lambda} | \phi_k \le 0\}$. These ψ-planes and ϕ-planes divide the whole space into some regions, denoted by $(\cap_{1 \le k \le 2n} \psi_k^*) \cap (\cap_{1 \le k \le n} \phi_k^*)$ for $\psi_k^* \in \{\psi_k^+, \psi_k^-\}$ and $\phi_k^* \in \{\phi_k^+, \phi_k^-\}$. It is obvious that each region is a cone in $\mathbf{R}^m$ space. Furthermore, every region has several hyperlines as the frames of their boundaries. Note that such a hyperline can be determined by $m - 1$ hyperplanes in $\mathbf{R}^m$ space uniquely, there are at most $\binom{3n}{m-1}$ hyperlines. As a special case, there are at most $3n$ lines in $\mathbf{R}^2$ space if $m = 2$.

Definition 2. A hyperline determined by arbitrary $m - 1$ ψ-planes and/or ϕ-planes in $\mathbf{R}^m$ space is termed a λ-line which corresponds to a λ-equation, that is,

$$\begin{cases} \xi_1 = 0 \\ \xi_2 = 0 \\ \cdots \\ \xi_{m-1} = 0 \end{cases}, \tag{1}$$

where $\xi_i \in \{\psi_1, \psi_2, \ldots, \psi_{2n}\} \cup \{\phi_1, \phi_2, \ldots, \phi_n\}$ and $\xi_i \ne \xi_j$ for $1 \le i, j \le m - 1$, $i \ne j$.

Mathematically Eq. (1) can be represented as

$$\boldsymbol{\xi} = \boldsymbol{U}\boldsymbol{\lambda}^{\mathrm{T}} = \mathbf{0}, \tag{2}$$

where $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_{m-1})^{\mathrm{T}}$, $\boldsymbol{U} = \begin{pmatrix} u_{1,1} & \cdots & u_{1,m} \\ \vdots & \ddots & \vdots \\ u_{m-1,1} & \cdots & u_{m-1,m} \end{pmatrix}$, and $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)$. On the other hand, it can also be of the form

$$\frac{\lambda_1}{v_1} = \frac{\lambda_2}{v_2} = \ldots = \frac{\lambda_m}{v_m} \tag{3}$$

since it passes through the origin of the coordinates, where $\boldsymbol{v} = (v_1, v_2, \ldots, v_m)$ denotes the direction vector of the λ-line. Let $\boldsymbol{u}_i = (u_{i,1}, u_{i,2}, \ldots, u_{i,m})$ be the normal vector of the hyperplane determined by the equation $\xi_i = 0$ for $1 \le i \le m - 1$. We have $\boldsymbol{v}$ is orthogonal with every $\boldsymbol{u}_i$ for $1 \le i \le m - 1$ because the λ-line belongs to every hyperplane, i.e.,

$$\boldsymbol{v} = \boldsymbol{u}_1 \times \boldsymbol{u}_2 \times \ldots \times \boldsymbol{u}_{m-1}. \tag{4}$$

Thus, we have

$$\boldsymbol{v}\boldsymbol{\lambda}^{\mathrm{T}} = \begin{vmatrix} \boldsymbol{\lambda} \\ \boldsymbol{u}_1 \\ \cdots \\ \boldsymbol{u}_{m-1} \end{vmatrix} = \begin{vmatrix} \lambda_1 & \lambda_2 & \ldots & \lambda_m \\ u_{1,1} & u_{1,2} & \ldots & u_{1,m} \\ \ldots & \ldots & \ldots & \ldots \\ u_{m-1,1} & u_{m-1,2} & \ldots & u_{m-1,m} \end{vmatrix} = \sum_{j=1}^{m} v_j \lambda_j \ , \tag{5}$$

where

$$v_j = (-1)^{j+1} \begin{vmatrix} u_{1,1} & \ldots & u_{1,j-1} & u_{1,j+1} & \ldots & u_{1,m} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ u_{m-1,1} & \ldots & u_{m-1,j-1} & u_{m-1,j+1} & \ldots & u_{m-1,m} \end{vmatrix} . \tag{6}$$

Definition 3. Given an equation of the form $\boldsymbol{v}\boldsymbol{\lambda}^{\mathrm{T}} = 0$ where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)$, $\boldsymbol{v} = (v_1, v_2, \ldots, v_m)$, and $v_1, v_2, \ldots, v_m$ are not zero simultaneously, a canonical solution of the equation is defined as

$$\begin{cases} \lambda_i = 1, \lambda_j = 0, & \text{if } \exists v_i = 0 \text{ and } \forall v_j \neq 0 \text{ for } 1 \leq i, j \leq m \\ \boldsymbol{\lambda} = (v_2, -v_1, 0, \ldots, 0) & \text{if } \forall v_i \neq 0 \text{ for } 1 \leq i \leq m \end{cases} . \tag{7}$$

Definition 4. The set Λ is denoted to be the set of all canonical solutions to ψ-equations and ϕ-equations. The hyperplane in $\mathbf{R}^{2n}$ corresponding to $F_\lambda = 0$, where $\boldsymbol{\lambda}$ is a canonical solution in the Λ set, is called a λ-plane.

Theorem 1 [5]. Suppose that a bounded convex set V is defined by the limit of Eq. (2) and Eq. (5) presented in [3]. Given a line in $\mathbf{R}^m$ corresponding to an equation $\boldsymbol{v}\boldsymbol{\lambda}^{\mathrm{T}} = 0$, if $F_\lambda = 0$ intersects V in $\mathbf{R}^m$ for any fixed point $\boldsymbol{\lambda} \neq \mathbf{0}$ in the line, such as its canonical solution, then for every $\boldsymbol{\lambda}$ in the line $F_\lambda = 0$ also intersects V.

Theorem 1 shows that there are at most $2n$ ψ-planes and n ϕ-planes, and there are no more than $\binom{3n}{m-1}$ λ-lines determined by these hyperplanes. Each of λ-line generates a canonical solution according to Definition 3, and each canonical solution forms a λ-plane in light of Definition 4. That is, there are at most $\binom{3n}{m-1}$ λ-planes to be tested, which is the same as the number of λ-planes checked by the Lambda Test and the Generalized Lambda Test.

4 Experimental Results

We test the DDIT through the NASA benchmark code. For the sake of clarity, we just draw out 100 loops at random, in which the number of operations varies from 17 to 354. As shown in Table 1, among these loops only 3 loops are above 4-level-nested, including two 5-level-nested loops and one 6-level-nested loops. In addition, when a data dependence analysis approach is applied, total 45154 pairs of array references with coupled subscripts have to be tested, 62.57% of which is 3-dimensional.

Table 1. Statistics of extracted loops

	Levels of nested loops					
	Total	1	2	3	4	Others
Number of loops	100	18	25	29	25	3
	Dimension of arrays					
	Total	1	2	3	4	5
Pairs of array references tested	45154	3178	5484	28252	6888	1352

In order to reduce the number of variables of linear systems and to validate our static data dependence analysis approach, we employ an innerprocedural constant propagation technique, manually, before invoking the DDIT algorithm. Except that, no other symbolic value propagation techniques or interprocedural dependence analysis techniques [6] are applied even if they are able to improve on the DDIT algorithm.

Table 2 reveals the success rate of the DDIT algorithm for multi-dimensional array references, by which we mean how often the DDIT detects a case where there is no data dependence. In the table *dependences proved* means they have been checked manually or by the DDIT algorithm; *dependences assumed* means they have been assumed to be dependent because of lacking of further detailed information, which can be found when nonlinear coupled subscripts are involved in pairs of array references or coupled subscripts can not be determined at compile-time at all; and *independences proved* means these data independences have been confirmed.

Table 2. The success rate of the DDIT for multi-dimensional array references

		Dependences proved		Dependences assumed		Independences proved	
		Number	Percent	Number	Percent	Number	Percent
2-dimensional	Manually	722	13.16	154	2.81	4608	84.03
	Checked by DDIT	786	14.33	235	4.29	4463	81.38
3-dimensional	Manually	5612	19.86	945	3.35	21695	76.79
	Checked by DDIT	6481	22.94	1060	3.75	20711	73.31
4-dimensional	Manually	984	14.29	336	4.88	5568	80.83
	Checked by DDIT	1011	14.68	360	5.23	5517	80.09
5-dimensional	Manually	90	6.66	135	9.98	1127	83.36
	Checked by DDIT	90	6.66	141	10.63	1121	82.91

Manual analysis results show that for the 2-dimensional array references 84.03% of them can be parallelized by the ILSP algorithm, for the 3-dimensional 76.79%, for the 4-dimensional 80.83%, and for the 5-dimensional 83.36%. When the DDIT is applied it is founded that 81.38% of pairs of 2-dimensional array references, 73.31% of pairs of the 3-dimensional, 80.09% of pairs of the 4-dimensional, and 82.91% of pairs of the 5-dimensional are independent. That is, the DDIT algorithm detects no data dependences for 31812 pairs of multi-dimensional array references, 75.79% of overall 41976 pairs. It is shown that the number of data independences checked by the DDIT algorithm is slightly smaller than that manually.

We use the Generalized Lambda Test to test the same dataset, as shown in Table 3. When the Generalized Lambda Test is applied it is founded that 78.52% of pairs of 2-dimensional array references, 57.60% of pairs of the 3-dimensional, 61.21% of pairs of the 4-dimensional, and 78.55% of pairs of the 5-dimensional are independent. That

is, the Generalized Lambda Test detects no data dependences for 25857 pairs of multi-dimensional array references, 61.60% of overall 41976 pairs. It can be concluded that the number of data independences checked by the DDIT algorithm is significantly greater than that checked by the Generalized Lambda Test: the increasing success rate was about 14.19%. It is satisfying.

Table 3. The success rate of the GLT for multi-dimensional array references

	Dependences proved		Dependences assumed		Independences proved	
	Number	Percent	Number	Percent	Number	Percent
2-dimensional array references	913	16.65	265	4.83	4306	78.52
3-dimensional array references	9543	33.78	2436	8.62	16273	57.60
4-dimensional array references	1824	26.48	848	12.31	4216	61.21
5-dimensional array references	109	8.06	181	13.39	1062	78.55

5 Conclusion

This paper has presented a new data dependence analysis approach, Dependence Difference Inequality Test, for our software pipelining algorithm ILSP. The DDIT extends the applicable range of the Generalized Lambda Test and is able to deal with linear coupled subscripts for multi-dimensional array references by employing dependence difference inequalities as constraints to variables in a linear system on their own or with others.

As the same as the Generalized Lambda Test, The DDIT only ascertains whether or not real-valued solutions exist because it is based on equality consistency checking. However, it is implemented for the ILSP algorithm in particular. Experimental results shows that compared to the Generated Lambda Test the DDIT increases the success rate without increasing time cost. Therefore, the DDIT seems to be a practical scheme to analyze data dependence for the ILSP algorithm.

On the other hand, a dynamic data dependence analysis approach is presented in [7], which can work with this method together to coping with data dependencies for software pipelining.

Acknowledgement

This work was partially supported by National Nature Science Foundation, grant number 60173010, of *P. R. China*.

References

1. Petersen, P. M., Padua, D. A.: Static and Dynamic Evaluation of Data Dependence Analysis Techniques. IEEE Transactions on Parallel and Distributed Systems 7 (1996) 1121–1132

2. Chang, W. L., Chu, C. P., Wu, J.: The Generalized Lambda Test: A Multi-Dimensional Version of Banerjee's Algorithm. International Journal of Parallel and Distributed Systems and Networks 2 (1999) 69–78
3. Qiao, L., Huang, W. T., Tang, Z. Z.: A Static Data Dependence Analysis Approach for Software Pipelining. In: Jin, H., Reed, D., Jiang, W. (eds.): Proceedings of IFIP International Conference on Network and Parallel Computing, Beijing, Lecture Notes in Computer Science. Springer-Verlag, Berlin Heidelberg New York (2005) accepted by NPC'05
4. Li, Z., Yew, Y. C., Zhu, C. Q.: An Efficient Data Dependence Analysis for Parallelizing Compilers. IEEE Transactions on Parallel and Distributed System 1 (1990) 26–34
5. Qiao, L.: On Data Dependencies in Software Pipelining. Doctorial Dissertation, Department of Computer Science, Tsinghua University, Beijing (2001)
6. Johnson, S. P., Cross, M., Everett, M. G.: Exploitation of Symbolic Information in Interprocedural Dependence Analysis. Parallel Computing 22 (1996) 197–226
7. Qiao, L., Huang, W. T., Tang, Z. Z.: A Dynamic Data Dependence Analysis Approach for Software Pipelining. In: Jin, H., Reed, D., Jiang, W. (eds.): Proceedings of IFIP International Conference on Network and Parallel Computing, Beijing, Lecture Notes in Computer Science. Springer-Verlag, Berlin Heidelberg New York (2005) accepted by NPC'05