Proc. Natl. Sci. Counc. ROC(A)
Vol. 24, No. 5, 2000. pp. 362-372

The K Test: an Exact and Efficient Knowledge-based Data Dependence Testing Method for Parallelizing Compilers

CHAO-TUNG YANG[*], SHIAN-SHYONG TSENG[**], AND WEN-CHUNG SHIH[***]

[*]*Ground System Section*
National Space Program Office
Hsinchu, Taiwan, R.O.C.
[**]*Department of Computer and Information Science*
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.
[***]*Microelectronics and Information System Research Center*
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

(Received June 7, 1999; Accepted February 3, 2000)

ABSTRACT

Many different classes of multiprocessors have been designed and implemented in industry and academia. Therefore, it has become an important issue to develop parallelizing compiling techniques that can exploit the potential power of multiprocessors. In this paper, we concentrate on the fundamental phase, data dependence analysis, in parallelizing compilers. We propose a new approach that integrates existing tests and makes good use of their advantages. This approach chooses an appropriate test using knowledge-based techniques, and then applies the resulting test to detect data dependences on loops. A rule-based system, called the K test, is developed using repertory grid analysis to construct the knowledge base. Simulation results show that the K test gives exact solutions in most of practical and contrived cases; furthermore, for system maintenance and extendibility, our approach is obviously superior to others. Therefore, we are trying to extend the knowledge-based approach to the whole field of parallelizing compiling.

Key Words: parallelizing compilers, data dependence testing, loop parallelization, knowledge-based, repertory grid analysis

I. Introduction

In the past decade, multiprocessors have been used to create a major new class of parallel and widely applicable machines. To achieve high speedup on such systems, it is plausible to decomposed tasks into several subtasks, which can be executed on different processors in parallel. Parallelizing compilers analyze sequential programs (Banerjee *et al*., 1993; Polychronopoulos, 1988; Zima and Chapman, 1990; Yang *et al*., 1994; Hsiao *et al*., 1994; Wolfe, 1996) to detect hidden parallelism and use this information for automatic restructuring of sequential programs into parallel subtasks on multiprocessors using scheduling algorithms (Tzen and Ni, 1993).

However, parallelizing compilers have not been well developed. Therefore, it has become an important issue to develop parallelizing compiling techniques that can exploit the potential power of multiprocessors. In particular, loops are a rich source of parallelism and can be used to achieve considerable improvement in efficiency on multiprocessors (Polychronopoulos, 1988). Therefore, we have investigated the possibility of solving the problem of data dependence testing on loops.

In brief, the *data dependence testing* problem is that of determining whether two references to the same array within a nest of loops may reference the same element of that array (Li *et al*., 1990; Goff *et al*., 1991; Maydan *et al*., 1991; Kong *et al*., 1991; Shen *et al*., 1990; Pugh, 1992). Traditionally, this problem has been formulated as *integer programming*, and the best integer programming algorithms are $O(n^{O(n)})$, where n is the number of loop indices (Schrijver, 1986). Obviously, these algorithms are too expensive to use. For this reason, a faster, but not necessarily exact, algorithm might be more desirable in some situations.

In this paper, we propose a *knowledge-based* approach to data dependence testing. First, a library of testing algorithms is constructed. The principle behind construction is that each algorithm has a different flavor of input cases. Next, a knowledge base has to be established (Shen *et al*., 1990), where the knowledge concerns how to choose an appropriate algorithm according to the features of the input.

Conceptually, knowledge-based systems are too ex-

pensive to use in data dependence testing. We present evidence that suggests this argument is wrong. First, researchers have found that once very simple subscripts are filtered out and tested inexpensively, only a few subscripts remain (< 3%) (Goff *et al*., 1991; Maydan *et al*., 1991). These subscripts can be easily taken care of using more complex and expensive tests without affecting the overall performance. As a result, current research in this area is no longer concentrating on efficiency, but on precision. Second, we have developed a rule-based system, called the *K test*, to implement our new approach (Shih *et al*., 1994). Experimental results reveal that the K test does not take too much time in comparison with the whole compiling process.

Furthermore, as for system maintenance and extendibility, our approach is obviously superior to others. If a new testing algorithm or testing technique is proposed, then we can integrate it into the K test easily by adding a knowledge base and rules. However, the power of the K test for detecting the parallelism on loops can be improved. Therefore, we are trying to extend the knowledge-based approach to the whole field of parallelizing compiling.

This paper is organized as follows. First, we introduce the historical background on this topic. In Section II, the data dependence testing problem is defined, and preliminary knowledge is presented. Next, our new approach is proposed in Section III, and experimental results are examined in Section IV. Section V gives some discussion on the K test. Finally, Section VI draws a conclusion and indicates future work.

II. Background

This section introduces the concepts of data dependence testing and knowledge-based systems used in this paper. A brief overview of data dependence analysis and testing is given in Section II.1. Then, we describe the knowledge-based system by applying repertory grid analysis to construct rules in Section II.2. Finally, we review the existing data dependence testing algorithms in Section II.3.

1. Data Dependence Analysis

Data dependence[1] is said to exist between two statements S_1 and S_2 if there is an execution path from S_1 to S_2, if both statements access the same memory location and if at least one of the two statements writes to the memory location (Pugh, 1992). There are three types of data dependence: *True* (*flow*) *dependence* occurs when S_1 writes to a memory location that S_2 later reads. *Anti-dependence* occurs when S_1 reads a memory location that S_2 later writes to. *Output dependence* occurs when S_1 writes to a memory location that S_2 later writes to.

```
    DO I1 = L1, U1
      DO I2 = L2, U2
        …
          DO In = Ln, Un
S1:         A(f1(I1, I2, …, In), …, fm(I1, I2, …, In)) = …
S2:         … = A(g1(I1, I2, …, In), …, gm(I1, I2, …, In))
          ENDDO In
        …
      ENDDO I2
    ENDDO I1
```

Fig. 1. A model of a nested loop.

The nested loop is based on the assumption that the increment step is normalized to one. Suppose that we want to test whether or not there exists dependence from statement S_1 to S_2 in the nested loop model as shown in Fig. 1. *Data dependence testing* is the method used to determine whether dependences exist between two subscript references to the same array in a nested loop. Let A = (α_1, α_2, α_3, …, α_v) and B = (β_1, β_2, β_3, …, β_v) be the integer vectors of n integer indices within the ranges of the upper and lower bounds of the n loops in the loop model. There is a dependence from S_1 to S_2 if and only if there exist A and B, such that A is lexicographically less than or equal to B such that $f(A) = g(B)$, where f and g are functions from Z^n to Z. Otherwise, the two array reference patterns are independent.

In this case, we say that the system of equations is *integer solvable* with the loop-bounds constraints. Otherwise, the two array reference patterns are said to be *independent*. For example, consider the program segment shown in Fig. 2. By means of data dependence analysis, we find that the statement S_2 exhibits true dependence on the statement S_1, and that the statement S_2 exhibits anti-dependence on S_3.

The data dependence testing problem is equivalent to integer programming if all f and g are linear functions. Because the integer programming algorithm is *NP-Complete*, we do not think it is possible to develop a practical test that can related to be applied to every conceivable case. Based on preliminary observation, we think that

```
      DO I = 2, N – 2
S1:       A[I] = B[I]
S2:       C[I] = A[I – 1]
S3:       D[I] = C[I + 2]
      ENDDO
```

Fig. 2. A program segment.

[1]Data dependence is normally defined with respect to the set of variables which are used and modified by a statement, denoted by the In/Out sets.

concepts knowledge-based systems should be useful in parallelizing compiling because the compiling process is not deterministic and because much domain knowledge may be needed to generate efficient machine codes.

2. Knowledge-based Systems

Knowledge-based systems are systems that depend on a vast base of knowledge to perform difficult tasks (Turban, 1992). A simplified knowledge-based system is shown in Fig. 3. The knowledge is saved in a knowledge base separate from the inference component. This makes it convenient to append new knowledge or update existing knowledge without recompiling the inferring programs.

The inference engine is the interpreter of the knowledge stored in the knowledge base. It examines the contents of the knowledge base and the data accumulated about the current problem, and derives additional data and conclusions. The inference engine attempts to find connections between *facts* and *conclusions*. The *rule-based* approach is one of the methods most commonly used in knowledge-based systems. This type of system uses knowledge encoded in the form of production rules, i.e., `If` ... `Then` ... `rules`.

The primary difficulty in building a knowledge base is acquiring the required knowledge. A number of methods have been developed to ease knowledge acquisition; among them, the primary technique is called *repertory grid analysis* (RGA) (Hwang and Tseng, 1990). An example of applying RGA is shown in Table 1, where 'X' means that the attribute has no relationship with the object (i.e., the attribute is a 'don't care' condition). According to the table, five rules can be generated. For example, the first column includes [A_1, Obj_1] = {9, 10, 12}, [A_2, Obj_1] = Yes and [A_3, Obj_1] = X; hence, the resulting rule $RULE_1$: If $A_1 \in$ {9, 10, 12} and A_2 = Yes Then GOAL = Obj_1 is generated. RGA is easy to use, but it suffers from the problem of *missing embedded meanings* (Hwang and Tseng, 1990). For example, when a doctor says the symptoms of a cold are headache, coughing and sneezing, he means that people who catch colds may exhibit these symptoms. However, in RGA, a person is not considered to have a cold unless all of the symptoms are present. To overcome this limitation, an *attribute ordering table* (AOT) is employed to elicit embedded meanings by recording the importance of each attribute to each object (Hwang and Tseng, 1990).

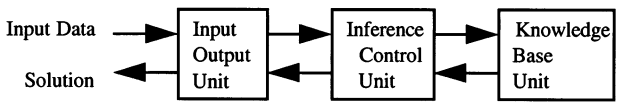

Fig. 3. Components of a simplified knowledge-based system.

Table 1. An Example of Applying RGA

	Obj_1	Obj_2	Obj_3	Obj_4	Obj_5
A_1	{9, 10, 12}	20	13	17	3
A_2	YES	NO	YES	YES	NO
A_3	X	X	4	2	6

The value of each AOT[attribute, object] entry may be labeled 'X', 'D' or an integer number. 'X' means that the attribute has no relationship to the object. 'D' means that the attribute dominates the object; i.e., if the attribute is not equal to the entry value, it is impossible to infer the object. Entries that are not labeled 'X' or 'D' have integers that represent the relative degree of importance of the object relative to other attributes. A larger integer number implies that the attribute is more important to the object. We use repertory grid analysis and an attribute ordering table to construct our knowledge base.

3. A Review of Existing Data Dependence Tests

The data dependence tests can be classified into two classes: single-dimensional tests and multi-dimensional tests.

A. Single-dimensional Tests

(1) GCD Test (Zima and Chapman, 1990): The GCD test is based on a theorem of number theory which says that Eq. (1) has an integer solution if and only if $gcd(a_1, a_2, ..., a_n)$ is a divisor of a_0:

$$a_1I_1 + a_2I_2 + \ldots + a_nI_n = a_0. \quad \textbf{(1)}$$

The GCD test is an integer test that ignores loop bounds.

(2) Banerjee Test (Zima and Chapman, 1990): The Banerjee test first derives a Diophantine equation, but the test treats the equation as a real-valued equation. Based on the intermediate value theorem, the equation is solvable if and only if the minimum of the left-hand side is not greater than zero and the maximum not smaller than zero.

(3) I Test (Kong *et al*., 1991): The I test is a combination of the GCD and Banerjee tests. It checks for the existence of integer solutions and considers limits.

B. Multi-dimensional Tests

(1) Extended-GCD Test: The GCD test was extended by Knuth to find a general integer solution to a set of linear equations. Banerjee described a matrix

form of the algorithm.

(2) Lambda Test (Li *et al*., 1990): The Lambda test is used for an efficient and accurate data dependence analysis. It extends the numerical methods to allow testing of all dimensions to be tested simultaneously.

(3) Power Test (Wolfe and Tseng, 1992): The Power test combines the Extended-GCD test, constraint tightening and the Fourier-Motzkin method to eliminate variables in a system of inequalities. However, it may be too expensive for use as a general purpose test in a compiler.

(4) Omega Test (Pugh, 1992): The Omega test determines whether there is an integer solution to an arbitrary set of linear equalities. Convention holds that integer programming techniques are far too expensive for use in dependence analysis, except as a method of last resort in situations that cannot be decided by simpler, special-case tests. Pugh suggested that the above argument is wrong. Consequently, their research implies that for linear cases, utilizing integer programming techniques to analyze data dependences is exact and not expensive.

C. Classification Approach

Reviewing previous work, we find that two papers are similar to this paper in some respects. In both studies, the authors collected a small set of test algorithms and tried to use them to solve the problem both efficiently and exactly in practical situations. However, our work is essentially different from theirs.

(1) Practical Test (Goff *et al*., 1991): The Practical test is based on classifying pairs of subscripted variable references. The major difference between the Practical test and other tests stems from the fact that the Practical test is essentially designed for practical input cases, and that its strategy is fixed. However, our approach is not limited to certain of input cases.

(2) MHL Test (Maydan *et al*., 1991): These authors showed that in practice, data dependence can be computed exactly and efficiently. The major difference between the MHL test and our approach is that the MHL test is a cascaded method; that is, the Extended-GCD test is tried first. If it fails, a next test is applied, and so on. However, our approach uses only one test after a conclusion is drawn.

III. A New Approach to Data Dependence Testing

As mentioned above, every existing test has its own advantages. In fact, if we always apply an appropriate test to a given case, the case can be correctly solved, and the execution time will not be too long. Based on this concept, we propose a knowledge-based approach to data dependence testing in this section. Section III.1 describes the organization of knowledge-based approach. Then, we introduce the K test, a rule-based system, in Section III.2. The algorithm of the K test and two examples are given in Section III.3. Finally, Section III.4, we describe the model of our Fortran parallelizing compiler, where the K test is used to detect the parallelism in.

1. Organization of the Knowledge-based Approach

A knowledge-based system is composed of two parts: the *development environment* and the *runtime environment* (Turban, 1992). The former is used to build the knowledge base, while the latter is used to solve the problem. In our paper, the *development environment* is not discussed. The *runtime environment* contains three components as shown in Fig. 4, which are briefly described as follows.

(1) Knowledge Base: This component contains knowledge required to solve the problem of determining an appropriate test to be applied. The knowledge can be organized in many different schemes and can be encoded into many different forms. Therefore, there exist many ways to build the knowledge base.

(2) Inference Component: This component is essentially a computer program that provides a method for reasoning about information in the knowledge base along with the input, and for forming conclusions.

(3) Testing Algorithm Library: The library collects several representative tests either proposed by others or designed by us. The question of how the tests are chosen is determined in the *development environment*, so here we assume that it has been built.

The dependence testing process can be described as follows. First, the input, a set of equations, is fed into the inference component. Then, the inference component reasons about knowledge and draws a conclusion, a test. Finally, the resulting test is applied, and the answer is gen-

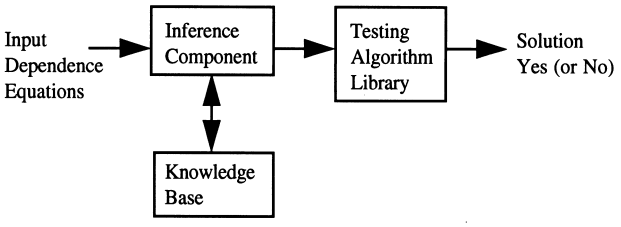

Fig. 4. Components of our approach.

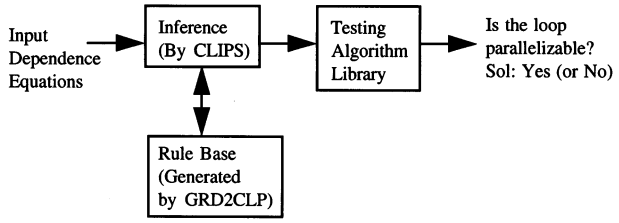

Fig. 5. Components of the K test.

erated. It should be noted that the knowledge base and the testing algorithm library shown in Fig. 4 are flexible; that is, they are not fixed. You can modify these two components so long as the efficiency and precision of the system are retained.

2. The Anatomy of the K Test

An implementation, called the *K test*, is proposed to demonstrate the effectiveness of the new approach. The K test is a rule-based system. The primary reason we choose a rule-based system is that this type of system is easy to understand; in addition, rule-based inference tools are widely available, which simplifies the implementation task.

The organization of the K test is shown in Fig. 5, which is similar to that in Fig. 4 except that the three components are replaced by actual software. We describe them briefly in the following.

(1) Knowledge Base: In our implementation, the knowledge base is constructed as a rule base; i.e., the knowledge is expressed in the form of production rules. These rules can be coded by hand or generated by a translator. In our system, the latter method is used. A translator, GRD2CLP, translates the repertory grid and attribute ordering table into CLIPS production rules. This approach has great flexibility as we can add new scheduling algorithms to the repertory grid and attribute ordering table, and then use GRD2CLP to convert the tables into CLIPS rules.

(2) Inference Component: This component is essentially a computer program that provides a method for reasoning about information in the knowledge base along with the input and for forming conclusions. An expert system shell, CLIPS, developed by NCSA with full source codes available, is used for the purpose of inference in our knowledge-based system shell. CLIPS, a forward-reasoning rule-based tool, is very efficient and does not increase the execution time cost of our system significantly. The basic elements of CLIPS are:

(i) Fact list – Facts are made up of fields that can be words, strings, or numbers. The first field of a fact is normally used to indicate the type of information stored in the fact. The following are examples of facts:

```
(single-field),

(two fields),

(speed 100 kmph).
```

There are commands for initiating, adding, removing and displaying facts.

(ii) Rule base – A rule is divided into an LHS and an RHS. The LHS of a rule can be thought of as the `If` portion, and the RHS can be thought of as the `Then` portion of the rule. Commands are available for displaying the rule list and the text of individual rules.

(iii) Inference engine – Rules that have patterns which are satisfied using facts produce an *activation*, which is placed in the *agenda*. CLIPS attempts to match the patterns of rules against facts in the fact list. If all the patterns of a rule match some facts, the rule is activated and put in the agenda, the set of activated rules.

We will illustrate the inference in CLIPS with an example.

Example 1.
Fact: `(sunny)`
Rule: 1. `If (rainy) Then (stay home)`
Rule: 2. `If (sunny) Then (go shopping).`
Inference: Rule 2 is matched because of the fact; thus, a result is obtained, and we go shopping.

(3) Testing Algorithm Library: We include four tests in the library. They are the GCD test, Banerjee test, I test and Power test. They are all existing tests. One of the four tests is chosen based on experience; therefore, it may not be the best choice. However, simulation results show that this approach is satisfactory to some extent.

The repertory grid of the K test is shown in Table 2, which contains four attributes and four objects that are

Table 2. The Repertory Grid of the K Test

	GCD	Banerjee	I	Power
Unity_Coef	1	5	1	1
Bound_Known	1	5	1	5
Multi_Dim	1	1	1	5
Few_Var	5	5	1	1

Table 3. The AOT of the K Test

	GCD	Banerjee	I	Power
Unity_Coef	1	2	1	1
Bound_Known	2	D	1	2
Multi_Dim	1	1	2	2
Few_Var	1	1	2	2

four existing data dependence tests. The four attributes are described below:

(1) Unity_Coef: whether the coefficients of variables are 1, 0, –1, or not. If they are, we set this value to be 5, otherwise 1.
(2) Bound_Known: whether the loop bounds are known or not. If the loop bound is known, we set this value to be 5, otherwise 1.
(3) Multi_Dim: whether the array references are multi-dimensional or not. We set this value to be the number of dimensions of the array reference.
(4) Few_Ver: whether the number of variables in the equation is small or not. We set this value to be the number of variables in the loop.

Four rules can be generated from the repertory grid, one rule per column, as follows:

```
(1) (defrule rule_01
    (Unity_Coef 1)
    (Bound_Known 1)
    (Multi_Dim known 1)
    (Few_Var known 5)
    ))
    =>
    (assert (goal GCD)))
(2) (defrule rule_02
    (Unity_Coef 5)
    (Bound_Known 5)
    (Multi_Dim known 1)
    (Few_Var known 5)
    ))
    =>
    (assert (goal Banerjee)))
(3) (defrule rule_03
    (Unity_Coef 1)
    (Bound_Known 1)
    (Multi_Dim known 1)
    (Few_Var known 1)
    ))
    =>
    (assert (goal I)))
(4) (defrule rule_04
    (Unity_Coef 1)
    (Bound_Known 5)
    (Multi_Dim known 5)
    (Few_Var known 1)
    ))
    =>
    (assert (goal Power)))
```

In order to elicit the embedded meanings from Table 2, we construct the AOT. The AOT of the K test is shown in Table 3. The process is described below in dialog form.

Algorithm: K test
Input:
(a_0^1, a_1^1, …, a_n^1, M_1^1, N_1^1, …, M_n^1, N_n^1,
…
a_0^m, a_1^m, …, a_n^m, M_1^m, N_1^m, …, M_n^m, N_n^m,
Unity_Coef, Bound_Known, Multi_Dim, Few_Var)
Output:
True: the input is integer solvable.
or False: the input is not integer solvable.
or Maybe: the input may be integer solvable.
Phase 1: calling CLIPS to draw a conclusion, that is, the most suitable dependence test.
Phase 2: calling the corresponding testing algorithm to check for data dependence.

Fig. 6. The K Test algorithm.

Q: If Bound_Known is not equal to 5, is it possible for the Banerjee test to be applied?
A: No.

The answer means that Bound_Known dominates the Banerjee test; hence, AOT[Bound_Known, Banerjee] = ‘D’.

Q: If Unity_Coef is not equal to 5, is it possible for the Banerjee test to be applied?
A: Yes.
Q: If Multi_Dim is not equal to 1, is it possible for the Banerjee test to be applied?
A: Yes.
Q: If Few_Var is not equal to 5, is it possible for the Banerjee test to be applied?
A: Yes.

The last three answers indicate that Unity_Coef, Multi_Dim and Few_Var do not dominate the Banerjee test, respectively, and that further questions have to be asked.

Q: Please rank the three attributes, Unity_Coef, Multi_Dim and Few_Var, in their order of importance with respect to the Banerjee test.
A: Unity_Coef is more important than Multi_Dim, and Multi_Dim is as important as Few_Var.

The answer indicates that Unity_Coef is more important than Multi_Dim, and that Multi _ Dim is as important as Few_Var. Therefore, AOT[Unity_Coef, Banerjee] is set to 2, AOT[Multi_Dim, Banerjee] is set to 1, and AOT[Few_Var, Banerjee] is set to 1. With AOT, 52 addi-

```
      DO  I = M, N
S1:         A[I + 1] = …
S2:         … = A[I]
      ENDDO
```

Fig. 7. The program segment of Example 1.

tional embedded rules can be generated; 15 rules from column one, 7 from column two, 15 from column three and 15 from the last column.

3. The Algorithm of the K Test

We will now summarize our discussion of the K test in an algorithm shown in Fig. 6. The algorithm consists of two phases. We will illustrate the K test with two examples.

Example 2. The program segment is shown in Fig. 7. For data dependences to exist between S_1 and S_2 due to the two references to A, the subscript of A referenced in S_1 should be equal to that in S_2. Hence, we can derive the following equations:

$$i_1 + 1 = i_2$$

$$i_1 - i_2 = -1$$

After the implicit equation is generated from the loop, and the attributes are set as follows:

Unity_Coef = 5: The coefficients are 1's or –1's.
Bound_Known = 1: The loop bounds are unknown values.
Multi_Dim = 1: The array reference is one-dimensional.
Few_Var = 5: The number of variables is two, thus Few.

During Phase 1, the CLIPS use those attributes and determine that the GCD test is the most suitable test. After Phase 2, the GCD test is invoked, and the result shows that the array references are data dependent. Therefore, they could not be parallelized.

Example 3. The program segment is shown in Fig. 8. For data dependences to exist between S_1 and S_2 due to the two references to A, the subscript of A referenced in S_1 should be equal to that in S_2. Hence, we can derive the following equations:

$$i_1 + 100 = i_2,$$

$$i_1 - i_2 = -100$$

After the implicit equation is generated from the loop, the attributes are set as follows:

```
      DO  I = 1, 100
S1:         A[I + 100] = …
S2:         … = A[I]
      ENDDO
```

Fig. 8. The program segment of Example 2.

Unity_Coef = 5: The coefficients are 1's or –1's.
Bound_Known = 5: The loop bounds are known values.
Multi_Dim = 1: The array reference is one-dimensional.
Few_Var = 5: The number of variables is two, thus Few.

During Phase 1, the CLIPS use these attributes and determine that the Banerjee test is the most suitable test. After Phase 2, the Banerjee test is invoked, and the result shows that the array references are data dependent. Therefore, they can be parallelized.

4. The Model of Our Parallelizing Compiler Using the K Test

A model of a Fortran parallelizing compiler intended to produce parallel object codes rather than being just a source-to-source translator (Hsiao *et al.*, 1994) is shown in Fig. 9. We describe this model below:

(1) Firstly, the parallelism detector treats the data dependence relations using the K test and then restructures a sequential Fortran source program into a parallel form; i.e., if a loop can be parallelized, then the parallelism detector converts it into DOALL loop. In the previous version of our compiler, we use, Parafrase-2 (p2fpp) to treat the data dependence relations (Polychronopoulos, 1988).

(2) Secondly, because there is no Fortran compiler on OSF/1 and because multithreading only supports C programming, a Fortran-to-C (f2c) (Feldman *et al.*, 1992) converter is used to convert the Fortran program, output by the K test, into its C equivalent. However, the output of the K test contains an extension (i.e., a DOALL statement) that is not in the standard Fortran 77 and, hence, is not recognized by f2c. Therefore, an additional module f2c_p (f2c preprocessor) is added to tackle this problem.

(3) Thirdly, the component, single-to-multiple threads translator (s2m) (Hsiao *et al.*, 1994) takes the obtained parallel form as input, where the parallel for loops are translated into subtasks by replacing them

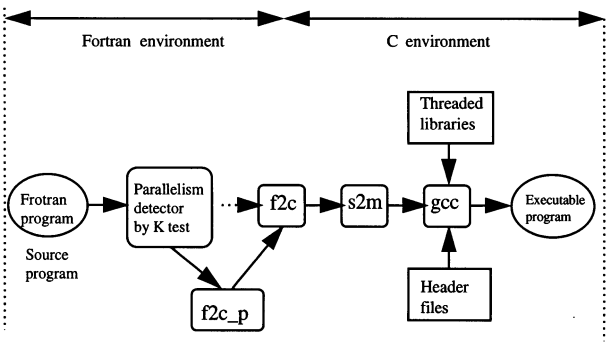

Fig. 9. A Fortran parallelizing compiler on OSF/1.

with multithreaded codes. The resulting multithreaded program is then compiled and linked with OSF/1's P Thread runtime library using the native C compiler, e.g., the GNU C compiler.

(4) Finally, the generated parallel object codes can be scheduled and executed in parallel on the multiprocessors under OSF/1 to achieve high performance.

Based on this model, we are implementing a Fortran parallelizing compiler to help programmers take advantage of multithreaded parallelism on OSF/1 multithreading OS.

IV. Experimental Results

In this section, we present experimental results which demonstrate that the K test is not expensive in terms of actual execution time, and that it gives correct answers in most cases. Section IV.1 describes the system environments in which the K test can be run. In Section IV.2, we use the practical data and contrived data to test the precision of the K test. Finally, we examine the execution time of the K test in Section IV.3.

1. System Environments

The experiment was performed on an HP Apollo 9000 Model 755 workstation with a 99-MHz PA-RISC processor. We had coded the GCD test, the Banerjee test, the I test, the Power test and the K test in the C programming language. Experiments conducted in previous works usually involved methods implemented in a prototype parallelizing compiler, such as Parafrase-2 (Polychronopoulos, 1988), Tiny (Pugh, 1992) etc. However, we could not afford such a large project. Consequently, we decided to construct a primary program that could read the input equations from a file, then call CLIPS, and finally invoke the tests. Hence, we examined all the input codes, selected the representative array subscripts, and encoded them into an input file using parallelism detector. For example, the simultaneous equations

$x - y = 1$

$-y + z = -1$

```
1: 2  3
2: 1  1  10
3: –1  1  10
4: 0  1  10
5: 1
6: 0  1  10
7: –1  1  10
8: 1  1  10
9: –1
```

Fig. 10. The input format of testing algorithms.

Table 4. Program Characteristics for Practical Data

	lines	subrs	array pair tested
EISPACK	11519	75	211
LINPACK	7427	51	106

where $1 \le x, y, z \le 10$ were encoded into the form shown in Fig. 10. The first line contains two numbers: the first number represents the number of equations, and the second refers to the number of variables. Hence, (2 3) means that there are 2 equations and 3 variables. Line 2 – line 5 represent the first equation, and line 6 – line 9 the second equation. The three numbers in line 2 refer to the coefficient, the lower bound and the upper bound, respectively.

2. Precision

Shen *et al*. (1990) found that many array references are not amenable to currently available data dependence tests. After examining LINPACK (Dongarra *et al*., 1979), EISPACK, and other numerical programs, we found that most of the array references in scientific code are very simple. However, examples appearing in related papers have been elaborately contrived to demonstrate the power of those methods. To find a compromise between off the two extremes, we divided the data into two classes. The first class, referred to as practical data, was selected from LINPACK and EISPACK; the second class, called contrived data, was collected from previous papers (Goff *et al*., 1991; Maydan *et al*., 1991; Pugh, 1992; Wolfe and Tseng, 1992).

A. Practical Data

We have performed experiments using two numerical packages, EISPACK and LINPACK. EISPACK is a collection of subroutines for computing the eigenvalues of matrices. LINPACK is a collection of Fortran subroutines that analyze and solve various systems of simultaneous linear algebraic equations. Because of their systemization and representatively, the packages have been widely adopted as benchmark programs (Goff *et al*., 1991; Li *et al*., 1990).

EISPACK has 75 subroutines, which contain about 70000 pairs of array references. LINPACK has 51 subroutines, which contain about 5000 pairs of array references (Goff *et al*., 1991). In our experiments, we considered only possible true dependent references. Table 4 lists the number of lines, subroutines and array pairs tested.

Table 5 shows the usage and success frequencies of the dependence tests for the two packages. The notations are similar to those in Goff *et al*. (1991). 'A' denotes the number of times the test was applied; 'S' denotes the

Table 5. Application/Success/Independence Frequencies for Practical Data

	EISPACK			LINPACK		
	A	S	I	A	S	I
GCD	206	204	0	83	83	0
Banerjee	0	0	0	0	0	0
I	206	204	0	83	83	0
Power	206	206	2	83	83	0
K	206	206	2	83	83	0

Table 7. Application/ Success/ Independence Frequencies for Contrived Data

	Contrived Data		
	A	S	I
GCD	14	5	5
Banerjee	12	7	7
I	14	12	12
Power	14	14	14
K	14	14	14

number of times the test succeeded in determining dependences; 'I' denotes the number of times the test proved the pair was data independent.

We know that practical array subscripts are usually simple, with unity coefficiencies and few index variables. Therefore, the GCD test seem to be sufficient for practical data, and the Power test and the K test were not significantly superior to the GCD test. Note that in LINPACK and EISPACK, the loop bounds are all parameters and are unknown values. This is why the Banerjee test could not be applied to the two packages.

B. Contrived Data

We collected 14 loop segments from Maydan *et al*. (1991), Shen *et al*. (1990), Wolfe and Tseng (1992) and Zima and Chapman (1990). They were special examples constructed to demonstrate the power of certain tests, so they rarely appear in practical programs. However, for these examples, actually more powerful tests are required to find their parallelism. The statistics are shown in Table 6.

Table 7 is similar to Table 5 except that it is for contrived data. In this case, we clearly find that the GCD test and the Banerjee test are less exact than the Power test and the K test.

3. Execution Time

Table 8 lists the execution times of five tests for the array references shown in Fig. 11. Every entry in the table denotes the time required to analyze this array reference pair on our HP workstation. Although the experiments were performed only on a specific array pair, we think using other input data would lead to similar results.

The first row in Table 8 shows the times required by the four tests themselves, and the second row lists the execution times of the K test when it invokes the four tests. For comparison, we ran the Parafrase-2 restructurer on the same HP workstation (Yang *et al*., 1994) and measured the execution time for the same loop segment. Consequently, about 0.1 second was required to analyze the data dependences. This implies that the K test is not too expensive, and that it can be adopted in a parallelizing compiler.

V. Discussion

1. Treatment Using the K Test

In both our practical and contrived cases, the K test achieve high accuracy. Although the number of array samples that we collected was not large in comparison with previous works (Goff *et al*., 1991; Shen *et al*., 1990), these samples were fairly selected from EISPACK, LINPACK, and previous works (Goff *et al*., 1991; Maydan *et al*., 1991; Pugh, 1992; Zima and Chapman, 1990). For this reason, we can say that the K test can give correct answers in most cases, at least in terms of these sources.

The execution time for the K test was significantly longer than that for the GCD test, the Banerjee test and the I test. The overhead resulted from the cost of the inferential process and from using the expensive Power test. However, the time required by the K test was nearly the same as that for the Power test in the worst case. In fact, the amount of time used was relatively small compared

Table 6. Loop Characteristics for Contrived Data

	one-dim	two-dim	total
Zima	6	1	7
Maydan	3	1	4
Shen	2	0	2
Wolfe	0	1	1

```
      DO  I = 1, 10
        DO  J = 1, 10
S1:         A[I, J] = …
S2:         … = A[J, J – 1]
        ENDDO
      ENDDO
```

Fig. 11. The input exmple of simulations.

Table 8. The Execution Time of the K Test on the HP Workstation

	GCD	Banerjee	I	Power
without K	10^{-5} sec	7×10^{-6} sec	10^{-5} sec	0.06 sec
with K	0.02 sec	0.03 sec	0.02 sec	0.07 sec

with the time needed for the whole compiling process.

Although several authors (Maydan *et al*., 1991; Pugh, 1992) have claimed that their methods are both efficient and exact, their accuracy was confined within specific domains. For nonlinear and other general cases, these methods are definitely inapplicable. Furthermore, Shen *et al*. (1990) observed that 47% of the array references in scientific codes are not linear. For this reason, we think it is necessary to consider nonlinear cases for data dependence testing. In this study, knowledge-based approaches were not directly utilized to deal with nonlinear cases, but it is easy to extend the knowledge-based approach; for example, if a new test for nonlinear cases is presented (Blume and Eigenmann, 1994), we only need to include the new test in our library and modify the repertory grid. In the above process, no modification of program logic is needed. Furthermore, the rule base of the K test is constructed using GRD2CLP, which translates a repertory grid into production rules, so modification of the rule base is easier.

2. A New Trend

It is well known that the major source of parallelism is loops. A loop is called a DOALL loop if there is no data dependence among all the iterations, i.e., all the iterations of a loop can be executed in parallel. Parallel loop scheduling is a method that schedules a DOALL loop on multiple processors (Polychronopoulos, 1988; Tzen and Ni, 1993). In the past, load balance among processors and synchronization operation overhead have been two main issues for various scheduling algorithms. We explain these issues in the following:

(1) Load Balance: If processors are idle, the process is not taking full of advantage of the multiprocessors. Scheduling algorithms try to distribute the workload among the multiprocessors as evenly as possible.

(2) Synchronization Overhead: The overhead results from the simultaneous accesses by processors to a set of shared variables that contain the indices of iterations.

In a shared-memory multiprocessor system, scheduling decisions can be made either statically at compile time or dynamically at runtime. Static scheduling is usually applied to iteration uniformly distributed iterations among processors. However, it has the drawback of load imbalance when the loop style is not uniformly distributed, and its loop bound must be known at compile time. In contrast, dynamic scheduling is more suitable for load balancing; however, the runtime overhead must be taken into consideration.

Traditionally, the parallelizing compiler dispatches the loop by using only one scheduling algorithm, either static or dynamic. However, programs have different kinds of loops, including uniform workload, increasing workload, decreasing workload, and random workload, loops, and every scheduling algorithm can achieve good performance on a different loop style (Tzen and Ni, 1993). To reduce the overhead and enhance load balancing, the knowledge-based approach is feasible solution for parallel loop scheduling. An approach that integrates existing static and dynamic scheduling algorithms and makes good use of their advantages will be proposed in the future. We can use this approach to choose an appropriate scheduling algorithm base on some features that include the loop style, loops bound, system status, data locality, and synchronization overhead, and then apply the resulting algorithm to schedule the DOALL loop on processors. In addition, we plan to study whether the knowledge-based approach can be applied to guide different types of loop transformation to obtain parallelism in parallelizing compilers.

VI. Conclusions and Further Work

Current research on data dependence testing is no longer concentrated on efficiency, but rather on precision. Since the actual time required to test data dependences is short in comparison with the execution time of the whole compiling process, researchers tend to adopt more complex algorithms in order to produce more results that are accurate and to treat more general input cases. Consequently, knowledge-based approaches have become feasible for data dependence testing, and provide a different means of achieving parallelizing compiling.

This paper has presented a new approach to data dependence testing based on knowledge-based methodology. To implement this new approach, a rule-based system called the K test has been developed. Experimental results show that the K test could give correct answers for most of the practical and contrived cases studied, and that the execution time was not high compared with that of a parallelizing compiler. In addition, our approach is obviously superior to others in terms of software maintenance. Our future research efforts will include formulating knowledge acquisition methodology for data dependence testing. In addition, we plan to determine whether knowledge-based approaches can be applied to guide different types of loop transformation to achieve parallelism and parallel loop scheduling in parallelizing compilers. We

believe that our research will provide more insight needed to develop a high performance parallelizing compilers.

Acknowledgment

This work was supported in part by the National Science Council of the Republic of China under grant NSC 84-2213-E-009-090.

References

Banerjee, U., R. Eigenmann, A. Nicolau, and D. A. Padua (1993) Automatic program parallelization. *Proc. IEEE*, **8**(12), 211-243.

Blume, W. and R. Eigenmann (1994) The range test: A dependence test for symbolic, non-linear expressions. *Supercomputing'94*, pp. 528-537, Washington D.C., U.S.A.

Dongarra, J., C. B. Moler, J. R. Bunch, and G. W. Stewart (1979) *LINPACK Users' Guide*. SIAM, Philadelphia, PA, U.S.A.

Feldman, S. I., D. M. Gay, M. W. Maimone, and N. L. Schryer (1992) *A FORTRAN-to-C Converter*. Computing Science Technical Report, No. 149, Bell Communication Research and Carnegie-Mellon University, Pittsburgh, PA, U.S.A.

Goff, G., K. Kennedy, and C. W. Tseng (1991) Practical dependence testing. *Proc. of the ACM SIGPLAN '91 Conf. on Programming Language Design and Implementation*, pp. 15-29, Toronto, Canada.

Hsiao, M. C., S. S. Tseng, C. T. Yang, and C. S. Chen (1994) Implementation of a portable parallelizing compiler with loop Prtition. *Proc. of the 1994 Int'l Conf. on Parallel and Distributed Systems*, pp. 333-338, Hsinchu, Taiwan, R.O.C.

Hwang, G. J. and S. S. Tseng (1990) EMCUD: a knowledge acquisition method which captures embedded meanings under uncertainty. *Int'l J. Man-Machine Studies*, **33**, 431-451.

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

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

Maydan, D. E., J. L. Hennessy, and M. S. Lam (1991) Efficient and exact data dependence analysis. *Proc. of the ACM SIGPLAN '91 Conf. on Programming Language Design and Implementation*, pp. 1-14, Toronto, Canada.

Polychronopoulos, C. D. (1988) P*arallel Programming and Compilers*. Kluwer Academic Publishers, Norwell, MA, U.S.A.

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

Schrijver, A. (1986) *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, NY, U.S.A.

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

Shih, W. C., C. T. Yang, and S. S. Tseng (1994) Knowledge-based data dependence testing on loops. *Proc. of the 1994 Int'l Computer Symposium*, pp. 961-966, Hsinchu, Taiwan, R.O.C.

Turban, E. (1992) *Expert Systems and Applied Artificial Intelligence*. Macmillan Publishing Co., New York, NY, U.S.A.

Tzen, T. H. and L. M. Ni (1993) Trapezoid self-scheduling: a practical scheduling scheme for parallel compilers. *IEEE Trans. Parallel Distrib. Syst.*, **4**(1), 87-98.

Wolfe, M. and C. W. Tseng (1992) The power test for data dependence. *IEEE Trans. Parallel Distrib. Syst.*, **3**(5), 591-601.

Wolfe, M. (1996) *High-Performance Compilers for Parallel Computing*. Addison-Wesley Publishing, New York, NY, U.S.A.

Yang, C. T., S. S. Tseng, and C. S. Chen (1994) The anatomy of parafrase-2. *Proc. Natl. Sci. Counc. ROC(A)*, **18**(5), 450-462.

Zima, H. P. and B. Chapman (1990) *Supercompilers for Parallel and Vector Computers*. Addison-Wesley Publishing and ACM Press, New York, NY, U.S.A.

一個基於知識庫之資料相依性測試法

楊朝棟[*]　曾憲雄[**]　時文中[***]

[*]行政院國家科學委員會太空計畫室籌備處地面系統組
[**]國立交通大學資訊科學系
[***]國立交通大學電子與資訊研究中心

摘　要

爲了有效率且精確地解決資料相依性問題，在本論文中我們發展一套基於知識庫的資料相依性測試方法，建立一個Rule-based系統叫K Test。K Test是利用表格擷取式的分析與屬性擇序表的方法來建構知識庫，並整合於PFPC平行編譯器中。這些知識庫系統會依照迴圈的特性加以分析，利用現有的資料相依性測試法，推理一個最適合的資料相依性測試法。模擬（含實際及人爲的資料）結果顯示K Test在大部分的實驗資料中，能推理出最合適的資料相依性測試法，以解決陣列的參考。