Włodzimierz BIELECKI, Tomasz KLIMEK, Maciej PIETRASIK
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY, WYDZIAŁ INFORMATYKI

An experimental study on recognizing classes of dependence relations

Prof. dr hab. inż. Włodzimierz BIELECKI

Prof. dr hab. inż. Włodzimierz Bielecki is head of the Software Technology Department of the West Pomeranian University of Technology, Szczecin. His research interest includes parallel and distributed computing, optimizing compilers, extracting both fine- and coarse grained parallelism available in program loops.

e-mail: wbielecki@wi.zut.edu.pl

Mgr inż. Maciej PIETRASIK

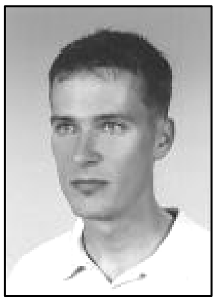
Maciej Pietrasik is Phd student of the Software Technology Departament of the West Pomeranian University of Technology, Szczecin. His research interest includes transitive closure dependence relations problem and extracting fine- coarse grained parallelism available in program loops.

e-mail: maciej.pietrasik@gmail.com

Mgr inż. Tomasz KLIMEK

Tomasz Klimek is Phd student of the Software Technology Departament of the West Pomeranian University of Technology, Szczecin. His research interest includes parallel and distributed computing, transitive closure algorithms and optimizing compilers.

e-mail: tklimek@wi.zut.edu.pl



Abstract

A classification of dependence relations representing exact dependences in program loops is presented. The class of a relation causes the choice of techniques for program loop parallelization. Techniques to recognize the class of a relation are presented. The implementation of these techniques by means of the Omega library is discussed. Results of an experimental study aimed at recognizing classes of dependence relations extracted for popular benchmarks (Livermore Loops, NAS, and UTDSP) are outlined.

Keywords: affine loops, dependence relations, program transformation, parallelization.

Techniki identyfikacji klas relacji zależności w pętlach programowych

Streszczenie

W artykule dokonano podziału relacji zależności występujących w pętlach programowych. Na podstawie przeprowadzonych obserwacji wyodrębniono sześć podstawowych klas takich relacji. Trafne rozpoznanie danej klasy relacji opisującej zależności, determinuje dobór odpowiedniej techniki transformacji pętli programowej i tym samym pozwala na uzyskanie znacznie większego jej stopnia równoległości w porównaniu z metodami bazującymi na rozwiązaniach przybliżonych. Rozwiązania takie, zawierają zdecydowanie większą liczbę zależności, aniżeli ich faktyczna liczba wystąpień. W celu ułatwienia procesu identyfikacji poszczególnych klas relacji zależności, przedstawiono szereg formalnych metod ich rozpoznania wykorzystujących szeroki wachlarz mechanizmów zawartych w bibliotece Omega. Na potrzeby przeprowadzonych badań zaimplementowano narzędzie, w ramach którego przeanalizowano zestawy pętli trzech popularnych benchmarków : Livermoore, NAS i UTDSP. Uzyskane wyniki pozwoliły wyciągnąć wnioski odnośnie procentowego udziału relacji zależności w zaproponowanych przez autorów klasach.

Słowa kluczowe: pętle afiniczne, relacja zależności, transformacje pętli, przetwarzanie równoległe.

1. Introduction

Dependence relations permit for an exact representation of dependences in program loops that allows for extracting more fine- and coarse-grained parallelism in program loops in comparison with techniques based on over-approximations of dependences. Dependence over-approximations describe more dependences than those actually existing in program loops. This reduces extracted parallelism. Dependence relations can belong to different classes, and it is very important to know what is the class of a particular relation because this affects the choice of techniques being used to extract parallelism available in program loops, e.g. a way of calculating the transitive closure of a relation. In this paper, we present classes of dependence relations, techniques to recognize the class of a given relation, and results of an experimental study for the three popular benchmarks: Livermore Loops, NAS, and UTDSP. The goal of experiments was verifying suggested techniques for recognizing classes of dependence relations and collecting data answering the question what is the number and percentage of relations belonging to presented classes of relations.

2. Dependence relations

Integer tuple relations can concisely summarize many types of information gathered from an analysis of scientific codes. For example, they can be used to precisely describe which iterations of a statement are data dependent of which other iterations. The following is an example of a relation from 1-tuples to 2-tuples :

$$\{ [i] \rightarrow [i', j'] : 1 \le i = i' = j' \le n \}$$

The relation above describes data dependences among instances of statement 1 and instances of statement 2 for the loop presented below:

```
        do 2 i = 1, n
s1:        a( i, i ) = 0
           do 2 j = 1, i
s2:           b( i, j ) = b( i, j ) + a ( i, j )
```

We use the term *dependence relation* rather than tuple relation when they describe data dependences. There is a data dependence from statement/instance of statement s_1 to statement/instance of statement s_2 (statement/instance of statement s_2 depends on statement/instance of statement s_1) if and only if:

2.1 both statements/instances of statement(s) access the same memory location and at least one of them stores into it,

2.2 there is a feasible run-time execution path from s_1 to s_2.

Each pair of dependent statement/instances is called a *dependence*. Any ordering-based optimization that does not violate dependences of a program does not change the output of the program. Dependences represent two different kinds of constraints on program transformations. First there are constraints designed to ensure that data is produced and consumed in the correct order. The dependences that arise from these constraints are called *data dependences*. The other constraint that gives rise to dependences is a control flow. A dependence that arises due to a control flow is called a *control dependence*. Although both data and control dependences must be considered when correctly parallelizing a program, we will concentrate exclusively on data dependences. There are three ways that a dependence can arise in a program [5]:

True dependence – the first statement/instance of statement stores into a location that is later read by the second statement/instance of statement. The statements below expose such a dependence.

```
S1    X = …
S2    … = X
```

Antidependence – the first statement/instance of statement reads from a location into which the second statement/instance of statement later stores. Such a dependence takes place for the following statements.

```
S1    … = X
S2    X = …
```

Output dependence – both statements/instance of statements write into the same location. The statements below originate such a dependence.

```
S1    X = …
S2    X = …
```

Dependences can be represented by dependence relations which are much more powerful abstraction than the traditional dependence distance or direction abstractions. The loop above has dependence distance (0), but that does not tell us that only the last iteration of loop *j* presented above is involved in the dependence. This type of additional information is crucial for determining the legality of applying advanced loop transformations. For that reason we divided the dependence relations into six main groups described in the next subsection.

3. Classes of dependence relations

Below, we consider the classes of parameterized affine integer tuple relations and present techniques permitting us to recognize them. These techniques can be implemented by means of many well-known public available tools, but we chose the functionality available in the Omega project software. Below we present classes of dependence relations being considered in this paper.

3.1 *d*-form relations [4] – a relation *R* is said to be in *d*-form if it can be written as

$$\left\{\begin{aligned}&[i_1,i_2,\dots,i_m]\to[j_1,j_2,\dots,j_m]:\ \forall p, 1\le p\le m,\\ &\qquad\qquad \exists\alpha_p\ s.t.\,(L_p\le j_p-i_p\le U_p\wedge j_p-i_p=M_p\alpha_p)\end{aligned}\right\} \quad (1)$$

where L_p and U_p are constants and M_p is an integer. If L_p is $-\infty$ or U_p is $+\infty$, the corresponding constraints are not included in the above equation. As we can see, if a *d*-form relation is described on an unbounded region, it has only constraints on the difference between the corresponding elements of the input and output relation tuples. Otherwise we should impose additional constraints on the domain and range of a relation. The following is the example of a relation described on the unbounded region

$$R=\{[i,j]\to[i',j']:i'-i\ge1\wedge j'-j=2\} \quad (2)$$

and below there is the example of a relation described on the bounded region:

$$R=\{\ [i,j]\to[i',j']:i'-i\ge1\wedge j'-j=2\wedge1\le i<n\wedge1\le j\le m-2\ \} \quad (3)$$

To check whether relation *R* belong to the *d*-form class, we create a relation $r=diffToRel(difference(R))$, where *difference*(*R*) is the function from the Omega library which returns a set of dependence distance vectors of relation *R*. When we impose constraints on the domain and range of a relation *r* as follows $r=(r\setminus domain(R))/range(R)$, we can check the following condition:

$$r-R=R-r=\varnothing \quad (4)$$

If it is true, then we can conclude that relation *R* satisfies the necessary condition to be a *d*-form relation. This condition means that if relation *r* constructed from a set of dependence distance vectors of relation *R* is equivalent to *R,* then relation *R* belong to the *d*-form class.

3.2 Uniform relations – a relation *R* is said to be uniform if it just like *d*-form relations has constraints on the difference between the corresponding elements of the input and output tuples and these differences are always constants, i.e,

$$\{[i_1,i_2,\dots,i_m]\to[j_1,j_2,\dots,j_m]:\ \forall p,1\le p\le m,\ j_p-i_p=M_p\} \quad (5)$$

For a relation being described on a bounded region, we can add additional constraints on the domain and range of the relation. Below is an example of the uniform relation

$$R=\{[i,j]\to[i',j']:i'-i=2\wedge j'-j=3\wedge1\le i\le n-2\wedge1\le j\le m-3\} \quad (6)$$

To check whether relation *R* has a uniform distance vector, we create two sets: min = *minimize* (*difference*(*R*)) and max = *maximize* (*difference* (*R*)), where *minimize* and *maximize* are the functions of the Omega library which calculate the minimal and maximal dependence distance vector from the given set *difference*(*R*). If the following condition is true:

$$\min-\max=\max-\min=\varnothing, \quad (7)$$

this means that dependence distance vector *difference*(*R*) is constant and *R* is a uniform relation.

3.3 Relations describing dependence chains only [1][2] – a relation *R* describes graphs of the chain topology only if it satisfies the following condition.

For each *s*/*d*=source/destination:
$s\in domain(R)$, $d\in range\ (R)$, there exists the exactly one destination/source *d*/*s*. For this purpose, we have to check whether there exist such s_i and s_j that $s_i\ne s_j$ and $R(s_i)=R(s_j)$ as well as such d_i and d_j that $d_i\ne d_j$, and $R^{-1}(d_i)=R^{-1}(d_j)$.

Below there is the example of a relation describing dependence chains only

$$R=\{[i,j]\to[i+1,j']:n+2j'=4j\wedge1\le i\wedge i+2j'<4j\wedge1\le j'\wedge3j'\le4j\} \quad (8)$$

Relation R describes a graph of the chain topology only if it satisfies the following condition:

$$(lf \circ R) \wedge R = (\neg lf \circ \neg R) \wedge \neg R = \varnothing \qquad (9)$$

where lf is the lexicographically forward relation such that $\forall\, x \rightarrow y \in R,\ 0 \prec y - x$. The first part of the condition means that there does not exist such r_i and r_j, $r_i \neq r_j$, $r \in range(R)$ that $R^{-1}(r_i)=R^{-1}(R_j)$, while the second part of the condition means that there does not exist such d_i and d_j, $d_i \neq d_j$, $d \in domain(R)$ that $R(d_i)=R(d_j)$.

3.4 Relations with coupled index variables - a relation R has coupled index variables if one or more input or output tuple elements are represented by an expression including two or more index variables. An example of such a relation is below

$$R=\{[l,i,k]\rightarrow[l',i-k-1,k']:1\le l<l'\le loop \wedge i<n \wedge 0\le k' \wedge 2+k+k'\le i \wedge 0\le k\} \qquad (10)$$

To check whether relation R has coupled index variables we:

(i) create a relations R_i, $1 \le i \le m$, in the following way

$$R_i = \{[s_i] \rightarrow [t_i] : \exists(s_j, t_j\ s.t.\ 1 \le j \neq i \le m \wedge [s_1, s_2, ..., s_m] : \text{constraints from } R \wedge [t_1, t_2, ..., t_m] : \text{constraints from } R\ )\ \} \qquad (11)$$

where m is the number of index variables of the input and output tuples of R and all these relations consist only of the i-th index variable. The rest of index variables we make to be existentially quantified.

(ii) create relation R_c, as the composition of relations R_i, $1 \le i \le m$, as follows:

$$R_c = \{[s_1, s_2, ..., s_m] \rightarrow [t_1, t_2, ..., t_m] : \mathop{\forall}_{i=0}^{m} \text{constraints on } s_i \text{ from } R_i \wedge \text{constraints on } t_i \text{ from } R_i\} \qquad (12)$$

(iii) if $R_c \neq R$ then we can conclude that relation R has coupled index variables. Let us consider the following example.

(i) For the above relation R, we create the following three relations

$$R_1 = \{[l] \rightarrow [l'] : \exists(i,k,k' : 1 \le l < l' \le loop\ \wedge i < n \wedge 0 \le k' \wedge 2 + k + k' \le i \wedge 0 \le k)\} = \{[l] \rightarrow [l'] : 1 \le l < l' \le loop \wedge 3 \le n\}$$

$$R_2 = \{[i]\rightarrow[i'] : \exists(l,l',k,k' : i''=i-k-1 \wedge 1\le l<l'\le loop \wedge i<n \wedge 0\le k' \wedge 2+k+k'\le i \wedge 0\le k)\} = \{[i]\rightarrow[i'] : 1\le i'<i<n \wedge 2\le loop\}$$

$$R_3 = \{[k] \rightarrow [k'] : \exists(l,l',i : 1 \le l < l' \le loop\ \wedge i < n \wedge 0 \le k' \wedge 2 + k + k' \le i \wedge 0 \le k)\} = \{[k] \rightarrow [k'] : 0 \le k \wedge 0 \le k' \wedge 2 \le loop \wedge 3 + k + k' \le n\ \}$$

(ii) Next we create relation R_c, as the composition of relations R_1, R_2, and R_3.

$$R_c = \{[l,i,k] \rightarrow [l',i',k'] : \underbrace{1 \le l < l' \le loop\ \wedge 3 \le n}_{\text{constraints from } R_1} \wedge \underbrace{1 \le i' < i < n \wedge 2 \le loop}_{\text{constraints from } R_2} \wedge \underbrace{0 \le k \wedge 0 \le k' \wedge 2 \le loop \wedge 3 + k + k' \le n}_{\text{constraints from } R_3}\} = \{[l,i,k] \rightarrow [l',i',k'] : 1 \le l < l' \le loop\ \wedge 1 \le i' < i < n \wedge 0 \le k' \wedge 3 + k + k' \le n \wedge 0 \le k\}$$

$R_c - R \neq \varnothing$, i.e. a relation R has coupled index variables.

3.5 Relations with non-coupled index variables - a relation R has non-coupled index variables if each its tuple element is represented by an expression including one index variable. For this type of relations we can apply the same approach as described above for relations with coupled index variables, and if the following condition is true:

(i) $R_c=R$, then we can conclude that relation R has not coupled index variables, i.e,

$$R = \{[i,j] \rightarrow [i',2j] : 1 \le i < i' \le n \wedge j \ge 1 \wedge 2j \le m\} \qquad (13)$$

3.6 Relations with different numbers of index variables of input and output relation tuples

The example of such a relation is as follows:

$$R=\{[i]\rightarrow[i',j] : 1\le i<i'\le n \wedge 1\le j\le m\} \qquad (14)$$

To check whether relation R has different numbers of index variables of input and output relation tuples, we use the functions $R.n\_inp()$ and $R.n\_out()$ from the Omega library which returns the numbers of input and output index variables. If the following condition is true

$$R.n\_inp() \neq R.n\_out(),$$

this means that relation R has the different numbers of index variables of the input and output relation tuples.

4. Experiments

The techniques recognizing classes of dependence relations presented in Section 2.1 were implemented in a tool based on the Omega library [3]. Using this tool, we have carried out experiments with Livermore [7], UTDSP [9], and NAS [8] benchmarks. Petit [6] was used to extract dependence relations for these benchmarks. Results of experiments are presented in Table 1. For each suite under experiments (Livermore loops, UTDSP, NAS), the most relations belong to d-forms. Numerous relations are uniform. Relations with the different numbers of index variables of input and output tuples often occur in benchmarks under experiments. Relations describing dependence chains were discovered for NAS benchmarks only (no such relations in the Livermore and UTDSP suites). There are less than 1% relations with coupled index variables.

Tab. 1. Wyniki badań
Tab. 1. Results of experiments

Benchmark	All relations	*d*-form	% *d*-form
	1.	2.	3.
Livemoore	1148	767	66,81 %
UTDSP	700	301	43 %
NAS	52098	31417	60,35 %

Benchmark	Uniform	% Uniform	Non-uniform chains	% Non-uniform chains	Relations with coupled index variables	% Relations with coupled index variables
	4.	5.	6.	7.	8.	9.
Livemoore	280	24,39 %	0	0	3	0,26 %
UTDSP	124	17,71 %	0	0	2	0,29 %
NAS	8359	16 %	10	0,02%	48	0,09 %

Benchmark	Relations with non-coupled index variables	% Relations with non-coupled index variables	Domain intersection Range = ∅	% Domain intersection Range = ∅	Relations with different numbers of index variables of input and output tuples	% relations with different numbers of index variables of input and output tuples
	10.	11.	12.	13.	14.	15.
Livemoore	1044	90,94%	22	1,92%	76	6,62 %
UTDSP	423	60,04%	26	3,71 %	244	34,86 %
NAS	39738	76,27%	772	1,48 %	11459	22 %

The experimental results demonstrate that for most loops exposing *d*-form and uniform relations, we can apply well-known techniques to discover parallelism available in loops. However, there exist relations with coupled index variables for which novel techniques are required permitting for extracting both fine- and coarse-grained parallelism.

5. Conclusion

We have presented a classification of dependence relations describing exact dependences in program loops. Techniques permitting for recognizing classes of relations were introduced. These techniques were implemented in a tool that was used to carry out experiments with popular benchmarks(Livermore loops, UTDSP, NAS). Experiment results demonstrate that most relations belong to the *d*-form or uniform class of relations for which extracting parallelism can be fulfilled by means of well-known approaches. But there exist relations with coupled index variables that require devising novel techniques to extract parallelism, for example, there is the need in developing techniques to calculate the transitive closure of a union of dependence relations where one or more ones are relations with coupled index variables. Devising such techniques is the goal of our future research.

6. References

[1] Bielecki W., Klimek T., Trifunovič K.: Calculating Exact Transitive Closure for a Normalized Affine Integer Tuple Relation, Electronic Notes in Discrete Mathematics, 33 (2009), pp. 7–14.
[2] Bielecki W., Klimek T., Trifunovič K.: Obliczenie potęgi k znormalizowanej afinicznej relacji, Metody Informatyki Stosowanej Nr 2/2008 (Tom 15).
[3] Kelly W., Maslov V., Pugh W., Rosser E., Shpeisman T., Wonnacott D.: The Omega library interface guide, Technical Report CS-TR-3445, Dept. of Computer Science, University of Maryland, College Park, March 1995.
[4] Kelly W., Pugh W., Rosser E., Shpeisman T.: Transitive clousure of infinite graphs and its applications, Languages and Compilers for Parallel Computing, 1995.
[5] Kennedy K., Allen John R.: Optimizing compilers for modern architectures: a dependence-based approach, Morgan Kaufmann Publishers Inc., 2001.
[6] Kelly W., Maslov V., Pugh W., Rosser E., Shpeisman T., Wonnacott D.: New User Interface for Petit and other Extensions, Technical Report, Dept. of Computer Science, University of Maryland, College Park, December 1996.
[7] http://www.netlib.org/benchmark/livermoore
[8] http://www.nas.nasa.gov/Software/NPB/
[9] http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.html

Artykuł recenzowany