Using basis dependence distance vectors in the modified Floyd–Warshall algorithm

Włodzimierz Bielecki · Krzysztof Kraska ·
Tomasz Klimek

Published online: 30 April 2014

© The Author(s) 2014. This article is published with open access at Springerlink.com

Abstract In this paper, we present a modified Floyd–Warshall algorithm, where the most time-consuming part—calculating transitive closure describing self-dependences for each loop statement—is computed applying basis dependence distance vectors derived from all vectors describing self-dependences. We demonstrate that the presented approach reduces the transitive closure calculation time for parameterized graphs representing all dependences in the loop in comparison with that yielded by means of techniques implemented in the Omega and ISL libraries. This increases the applicability scope of techniques based on transitive closure of dependence graphs and being aimed at building optimizing compilers. Experimental results for NASA Parallel Benchmarks are discussed.

Keywords Basis dependence vectors · Transitive closure · Floyd–Warshall algorithm · Arbitrarily nested loop · Parallelizing compiler

1 Introduction

Resolving many problems is based on calculating transitive closures of graphs [Diestel \(2010\)](#). In this paper, we deal with parameterized graphs whose number of vertices is represented with an expression including structure parameters. Such graphs can be

W. Bielecki · K. Kraska · T. Klimek (✉)
Faculty of Computer Science and Information Technology, West Pomeranian
University of Technology, ul. Żołnierska 49, 71-210 Szczecin, Poland
e-mail: tklimek@wi.zut.edu.pl

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

K. Kraska
e-mail: kkraska@wi.zut.edu.pl

represented by parameterized relations whose tuples represent vertices while constraints are responsible for defining edges Kelly et al. (1996). Transitive closure calculated for such relations can be used in optimizing compilers: to remove redundant synchronization Kelly et al. (1996), test the legality of iteration reordering transformations Kelly et al. (1996), apply iteration space slicing Beletska et al. (2011), form schedules for statement instances of program loops Bielecki et al. (2012); Hollermann et al. (1997); Deng et al. (1998). In general, calculating transitive closure of parameterized graphs is time-consuming Kelly et al. (1996); Beletska et al. (2009); Verdoolaege et al. (2011). Sometimes the time of transitive closure calculation prevents applying techniques for extracting coarse- and fine-grained parallelism because this time is not acceptable in practice (several hours and even several days Beletska et al. (2011); Bielecki et al. (2012)). This is why improving transitive closure calculation algorithms aimed at reducing their time complexity is an actual task.

The contributions of this paper over previous work are as follows:

- demonstration of how to calculate basis dependence distance vectors for parameterized program loops;
- proposition of a way of calculating the transitive closure of a dependence relation describing all self-dependences among the instances of a given loop statement by means of basis dependence distance vectors;
- suggestion to apply the transitive closure of a dependence relation describing all self-dependences among the instances of a given loop statement by means of basis dependence distance vectors in a modified Floyd–Warshall algorithm with the aim of reducing the calculation time of the transitive closure of a dependence graph representing all the dependences in a given program loop;
- development of an open source software implementing presented solutions and permitting for producing the transitive closure of a dependence graph describing all the dependences for the input program loop by means of the modified Floyd–Warshall algorithm;
- an evaluation of the effectiveness and efficiency of the presented algorithms and a comparison of them with those yielded by related work.

In this paper, we demonstrate how to reduce the time of calculating transitive closure describing self-dependences. For this purpose, we propose to find basis dependence distance vectors from all distance vectors describing self-dependences and then demonstrate how these vectors can be used for calculating transitive closure. For extracting such distance vectors, dependence relations, returned by means of a dependence analyzer, are used. Such relations (describing self-dependences) are characterized by the same arity (the number of tuple elements) of input and output tuples. Finally, we present experimental results showing how the time of transitive closure calculation is reduced for NAS benchmarks (<http://www.nas.nasa.gov>).

The rest of the paper is organized as follows. Section 2 introduces background. Section 3 presents an approach to calculate the transitive closure of a parameterized dependence graph. Section 4 describes related work. Section 5 presents results of an experimental study. Section 6 draws conclusions and briefly outlines future research.

2 Background

In this section, we briefly introduce basic definitions which are used throughout this paper.

The following concepts of linear algebra are used in the approach presented in this paper: vector, vector space, field, integral linear combination, linear independence. Details can be found in book [Schrijver \(1999\)](#).

Definition 1 (*Integer Lattice*) Let $\{a_1, a_2, \dots, a_m\}$ be a set of linearly independent integer vectors. The set $\Lambda = \{\lambda_1 a_1 + \lambda_2 a_2 + \dots + \lambda_m a_m \mid \lambda_1, \dots, \lambda_m \in \mathbb{Z}\}$ is called an integer lattice generated by the basis $\{a_1, a_2, \dots, a_m\}$.

Definition 2 (*Basis*) A basis B of an integer lattice Λ over field $\mathbb{Z}$ is a linearly independent subset of Λ that generates Λ . Every finite-dimensional vector space Λ has a basis [Shoup \(2005\)](#).

Definition 3 (*Presburger Arithmetic, Presburger Formula*) We define Presburger arithmetic to be the first-order theory over atomic formulas of the form:

$$\sum_{i=1}^n a_i x_i \sim c, \quad (1)$$

where a_i and c are integer constants, x_i are variables ranging over integers, and $\sim$ is an operator from $\{=, \neq, <, \leq, >, \geq\}$. A formula f is an atomic formula (1) or it is constructed from formulas f_1 and f_2 recursively as follows [Presburger \(1927\)](#):

$$f ::= \neg f_1 \mid f_1 \wedge f_2 \mid f_1 \vee f_2.$$

In this paper, we deal with the following definitions concerned program loops: iteration vector, loop domain (index set), parameterized loops, perfectly-nested loops, details can be found in papers [Griebl \(2004\)](#).

Definition 4 (*Structure Parameters*) Structure parameters are integer symbolic constants, generally defining array size, iteration bounds, etc. Structure parameters may be defined once in the prologue of the program, and may not be modified elsewhere.

Definition 5 (*Iteration Vector*) For a given statement S in a loop, the iteration vector $\vec{v} = (i_1, \dots, i_n)^T$ is the vector of the surrounding loop counters.

Definition 6 (*Iteration Space*) The iteration space of a given statement S in a given loop nest is a set of values taken by its iteration vector when executing the loop nest.

Definition 7 (*Dependence*) Two statement instances $S_1(\vec{I})$ and $S_2(\vec{J})$, where $\vec{I}$ and $\vec{J}$ are the iteration vectors, are dependent if both access the same memory location and if at least one access is a write.

Fig. 1 The parameterized vector $(N, 2)$ represented as the integral linear combination of the two linearly independent vectors with constant coordinates

Definition 8 (*Dependence Distance Set, Dependence Distance Vector*) We define a dependence distance set $\Delta_{S,T}$ as a set of differences between all such vectors of the same size that stand for a pair of dependent instances of statement T and S . We call each element of set $\Delta_{S,T}$ a (dependence) distance vector and denote it as $\delta_{S,T}$.

Definition 9 (*Uniform Dependence, Non-Uniform Dependence*) If each coordinate of vector $\delta_{S,T}$ (see Definition 8) is constant, then we call a correspondent dependence uniform, otherwise it is non-uniform. Griebel (2004).

Definition 10 (*Reduced Dependence Graph*) A Reduced Dependence Graph (*RDG*) is the graph where a vertex stands for every statement S and an edge connects statements S and T whose instances are dependent. The number of edges between vertices S and T is equal to the number of dependence relations $R_{S,T}$.

Definition 11 (*Uniform Loop, Quasi-Uniform Loop*) We say that a parameterized loop is uniform if it induces dependences represented by the finite number of uniform dependence distance vectors Griebel (2004). A parameterized loop is quasi-uniform if all its dependence distance vectors can be represented by an integral linear combination of the finite number of linearly independent vectors with constant coordinates.

Let us consider the parameterized dependence distance vector $(N, 2)$. It can be represented as $(0, 2) + a \times (1, 0)$, where $a \geq 1, a \in \mathbb{Z}$ (see Fig. 1).

Definition 12 (*Dependence Relation*) A dependence relation is a tuple relation of the form $\{[input_list] \rightarrow [output_list] : constraints\}$, where *input_list* and *output_list* are the lists of variables used to describe input and output tuples and constraints is a Presburger formula describing the constraints imposed upon *input_list* and *output_list*.

The general form of a dependence relation is as follows Kelly et al. (1996):

$$R = \left\{ [s_1, \dots, s_k] \rightarrow [t_1, \dots, t_k] : \bigvee_{i=1}^n \exists \alpha_{i1}, \dots, \alpha_{im_i} \text{ s.t. } F_i \right\},$$

where $F_i, i = 1, 2, \dots, n$ are represented by Presburger formulas, i.e., they are conjunctions of affine equalities and inequalities on the input variables $s_1, \dots, s_k$, the output variables $t_1, \dots, t_k$, the existentially quantified variables $\alpha_{i1}, \dots, \alpha_{im_i}$, and symbolic constants.

Definition 13 (*Domain of a Relation*) Let $R \in Z^n \rightarrow Z^m$ be a relation. The domain of R , $\text{dom } R$, is represented with the following set

$$\text{dom } R := s \rightarrow \{x_1 \in Z^n \mid \exists x_2 \in Z^m : x_2 = R(x_1)\}.$$

Definition 14 (*Range of a Relation*) Let $R \in Z^n \rightarrow Z^m$ be a relation. The range of R , $\text{ran } R$, is calculated as follows

$$\text{ran } R := s \rightarrow \{x_2 \in Z^m \mid \exists x_1 \in Z^n : x_1 = R^{-1}(x_2)\}.$$

Definition 15 (*Positive Transitive Closure*) Let R be an affine integer tuple relation, then the positive transitive closure R^+ of R is the union of all positive powers of R ,

$$R^+ = \bigcup_{k \geq 1} R^k, \quad \text{with } R^k = \begin{cases} R & \text{if } k = 1 \\ R \circ R^{k-1} & \text{if } k \geq 2. \end{cases} \quad (2)$$

Definition 16 (*Transitive Closure*) Transitive closure, R^* , is defined as follows Kelly et al. (1996):

$$R^* = R^+ \cup I,$$

where I is the identity relation. R^* describes the same connections in a dependence graph (represented by R) that R^+ does plus connections of each vertex with itself.

To check whether output returned by an algorithm represents exact transitive closure, we can use the well-known fact Kelly et al. (1996) that for an acyclic relation R (for such a relation $R \cap I = \emptyset$, where I is the identity relation) the following is true:

- if R^+ is exact transitive closure, then:

$$R^+ = R \cup (R \circ R^+),$$

- if R^+ is an over-approximation, then:

$$R^+ \not\subseteq R \cup (R \circ R^+).$$

In the next section, we analyse the time complexity of the proposed approach in a machine-independent way to assess the performance of algorithms. For this purpose, the RAM (*Random Access Machine*) model of computation is used. Under the RAM model, we measure time complexity by counting up an upper bound, O , on the number of steps that an algorithm takes for a given problem. Details on the model and the time complexity analysis can be found in paper Skiena (2008).

3 Approach to computing transitive closure

The goal of the algorithm presented below is to calculate the transitive closure of a dependence relation describing all the dependences in the arbitrary-nested loop.

3.1 Floyd–Warshall algorithm

To compute the transitive closure of a dependence relation representing all the dependences exposed for an arbitrarily nested loop, we use a modified form of the Floyd–Warshall (F-W) algorithm (see Algorithm 1). The idea of the F-W algorithm is the following. Let $\xrightarrow{*}$ denote a direct or transitive path between a pair of vertices in a dependence graph, whose intermediate vertices come from a specific set S . If the graph contains paths $v \xrightarrow{*} w$ and $w \xrightarrow{*} u$, then the graph also contains a path $v \xrightarrow{*} u$ such that its intermediate vertices come from the set $S \cup \{w\}$. F-W's algorithm iterates from 1 to n where n is the total number of statements in the loop and in the k -th iteration it takes into account the paths whose intermediate vertices come from the set $\{v_1, \dots, v_{k-1}\}$.

Algorithm 1 The modified Floyd–Warshall algorithm Kelly et al. (1996)

```

1:
  Input :  $D^{N \times N}$  array whose element  $i, j$  represents a dependence relation,  $R_{ij}$ , describing all direct
  dependences exposed for instances of statement  $i$  and statement  $j$ , where  $N$  is the total number of
  statements in the loop, if there exists a dependence from instances of statement  $i \leq N$  to instances of
  statement  $j \leq N$  then  $R_{ij} \neq \emptyset$ , otherwise  $R_{ij} = \emptyset$ .
2:
  Output :  $D^{N \times N}$  array, where each element  $D_{ij}$  represents a relation,  $R_{ij}$ , describing transitive closure
  between instances of statement  $i$  and instances of statement  $j$ .
3:
4: Method:
5:
6: for each statement  $k$ 
7:    $R_{kk}^* = D_{kk}^*$ 
8:   for each statement  $i$ 
9:     for each statement  $j$ 
10:       $D_{ij} = D_{ij} \cup (D_{kj} \circ R_{kk}^* \circ D_{ik})$ 
11:

```

The most time-consuming part in Algorithm 1 is the expression $D_{kj} \circ R_{kk}^* \circ D_{ik}$, where ‘ $\circ$ ’ denotes the composition operator applied to a pair of relations, D_{ik} describes all the dependences between instances of statement s_i and statement s_k . This means that if there is a dependence from iteration i_1 of statement s_i to iteration i_2 of statement s_k and a chain of self-dependences from iteration i_2 to iteration i_3 , R_{kk}^* , and finally a dependence from iteration i_3 of statement s_k to iteration i_4 of statement s_j (where D_{kj} describes all dependences between instances of statement s_k and statement s_j), then there is a transitive dependence from iteration i_1 to iteration i_4 . It should be clear that the objective of this technique is to update all the dependences through statements $1, 2, \dots, n$ in an iteration of each k -loop. In Sect. 3.3, we suggest calculating R_{kk}^* using a finite integral linear combination of basis dependence distance vectors Bielecki et al. (2013).

3.2 Replacing the parameterized vector with an integral linear combination of constant vectors

To find constant vectors whose integral linear combination represents the parameterized vector, we can apply the following theorem.

Theorem 1 Let v_p be a vector in $\mathbb{Z}^d$ and p_i are its parameterized coordinates in the i -positions. We may replace vector v_p with an integral linear combination of a constant vector v_c , $v_c \in \mathbb{Z}^d$, and unit normal vectors e_i , $e_i \in \mathbb{Z}^d$, where p_i are integer parametric coefficients, as follows:

$$v_p = v_c + \sum_i p_i \times e_i. \quad (3)$$

Proof Without loss of generality, we may assume that the first n positions of v_p have constant coordinates and the last q positions have parameterized ones. Then, we can write:

$$\begin{pmatrix} c_1 \\ \vdots \\ c_n \\ p_{n+1} \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ p_{n+1} \\ \vdots \\ p_d \end{pmatrix}, \quad (4)$$

where here and further $d - n = q$, the second vector can be written as the integral linear combination of unit normal vectors e_k and parameterized coefficients $p_{n+1}, \dots, p_d$ in the last d positions:

$$\begin{pmatrix} 0 \\ \vdots \\ 0 \\ p_{n+1} \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ p_{n+1} \\ \vdots \\ 0 \end{pmatrix} + \dots + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ p_d \end{pmatrix} = p_{n+1} \times \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \dots + p_d \times \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}. \quad (5)$$

Substituting (5) into (4), we obtain:

$$\begin{pmatrix} c_1 \\ \vdots \\ c_n \\ p_{n+1} \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \\ 0 \\ \vdots \\ 0 \end{pmatrix} + p_{n+1} \times e_{n+1} + \dots + p_d \times e_d. \quad (6)$$

It is obvious that if $v_c = \mathbf{0}$, then v_c can be rejected without affecting the result. $\square$

3.3 Algorithm for computing transitive closure

The idea of the algorithm presented in this section is the following. Given a set $\Delta_{S,T}$ of m dependence distance vectors in the n -dimensional integer space derived from a union of dependence relations R_{kk} , $k = 1, 2, \dots, q$, where q is the number of loop statements (it describes a chain of self-dependences of statement s_k in the loop), we first replace all parameterized vectors with constant vectors using Theorem 1 from Sect. 3.2.

As a result, we get, $k, k \geq m$, dependence distance vectors with constant coordinates. This allows us to get rid of parameterized vectors and to form an integer matrix A , $A \in \mathbb{Z}^{n \times k}$, by inserting dependence distance vectors with constant coordinates into columns of A that generate integer lattice Λ .

To decrease the complexity of further computations, redundant dependence distance vectors are eliminated from matrix A by finding a subset of, $l, l \leq k$, linearly independent columns of A . This subset of dependence distance vectors forms the basis $B \in \mathbb{Z}^{n \times l}$ of A and generates the same integer lattice Λ as A does Schrijver (1999). Every element of integer lattice Λ can be expressed uniquely as a finite integral linear combination of the basis dependence distance vectors belonging to B .

After B is completed, we can work out relations R_{kk}^* , $k = 1, 2, \dots, q$, representing the exact transitive closure R_{kk} or its over-approximation. For each vertex x in the data dependence graph (where x is the source of a dependence, $x \in \text{dom } R_{kk}$), we can identify all vertices y (the destination(s) of a dependence(s), $y \in \text{ran } R_{kk}$) that are connected with x by a path of length equal or more than 1, where y is calculated as x plus an integral linear combination of the basis dependence distance vectors B , i.e. $y = x + B \times \lambda$, $\lambda \in \mathbb{Z}^l$. The part $B \times \lambda$ of the formula represents all possible paths in the dependence graph, represented by relation R_{kk} , connecting x and y . Moreover, we have to preserve the lexicographic order for y and x , i.e. $y - x \succ 0$. Below, we present the algorithm in a formal way.

Algorithm 2. *Calculating the transitive closure of a dependence relation R_{kk} describing a chain of self-dependences of a loop statement.*

Input: Dependence distance set $\Delta^{n \times m} = \delta_1, \delta_2, \dots, \delta_m$, where m is the number of n -dimensional dependence distance vectors.

Output: Exact transitive closure of relation R_{kk} or its over-approximation.

Method:

1. Replace each parameterized dependence distance vector in $\Delta^{n \times m}$ with an integral linear combination of vectors with constant coordinates. For this purpose apply Theorem 1 presented in Sect. 3.2.
2. Using all constant dependence vectors, form matrix $A \in \mathbb{Z}^{n \times k}$, $k \geq m$.
3. Extract a finite subset of, $l, l \leq k$, linearly independent columns from matrix $A \in \mathbb{Z}^{n \times k}$ over field $\mathbb{Z}^n$ and form matrix $B^{n \times l}$, representing the basis of the dependence distance vectors set, where linearly independent vectors are represented with columns of matrix $B^{n \times l}$. For this purpose apply the Gaussian elimination algorithm Shoup (2005); Rotman (2003).
4. Calculate relation R_{kk}^* representing the exact transitive closure of a dependence relation, describing all the dependences in the input loop, or its over-approximation, as follows:

$$R_{kk}^* = \left\{ [x] \rightarrow [y] \mid \exists \lambda \text{ s.t. } y = x + B^{n \times l} \times \lambda \wedge y - x \succ 0, \lambda \in \mathbb{Z}^l \wedge \right. \\ \left. \wedge y \in \text{ran } R_{kk} \wedge x \in \text{dom } R_{kk} \right\} \cup I, \quad (7)$$

where : R_{kk}^* describes a chain of self dependences of loop statement s_k , $B^{n \times l} \times \lambda$ represents an integral linear combination of the basis dependence distance vectors δ_i (the columns of $B^{n \times l}$, $1 \leq i \leq l$), $y - x \succ 0$ imposes the lexicographically forward constraints on the tuples of R_{kk}^* , I is the identity relation.

The resulting relation R_{kk}^* represents the exact transitive closure of relation R_{kk} or its over-approximation [Feautrier \(2012\)](#). To prove this, let us note that relation R_{kk}^+ represents all possible paths between vertices x (standing for dependence sources, $x \in \text{dom } R_{kk}$) and vertices y (standing for dependence destinations, $y \in \text{ran } R_{kk}$) in the dependence graph, represented with relation R_{kk} . Indeed, an integral linear combination of the basis dependence distance vectors $B^{n \times l} \times z$:

- reproduces all dependence distance vectors exposed for the loop,
- describes all existing (true) paths between any pair of x and y as an integral linear combination of all dependence distance vectors exposed for the loop,
- can describe not existing (false) paths in the dependence graph represented by relation R_{kk}^* .

There are two cases when false (not existing) paths may occur. The first one arises due the fact that in an integral linear combination of linearly independent columns of matrix A some coefficients can be negative. Let us consider the matrix A ,

$$A = \begin{bmatrix} 3 & 3 & 2 & \boxed{5} & \boxed{4} \\ 2 & 0 & -2 & \boxed{0} & \boxed{2} \end{bmatrix}.$$

The linearly independent columns are represented with the first three columns only because the rest ones can be represented as the integral linear combinations below (see [Fig. 2a](#)).

$$\begin{pmatrix} 5 \\ 0 \end{pmatrix} = 1 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} + 0 \times \begin{pmatrix} 3 \\ 0 \end{pmatrix} + 1 \times \begin{pmatrix} 2 \\ -2 \end{pmatrix}, \\ \begin{pmatrix} 4 \\ 2 \end{pmatrix} = 0 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} + 2 \times \begin{pmatrix} 3 \\ 0 \end{pmatrix} \boxed{-1} \times \begin{pmatrix} 2 \\ -2 \end{pmatrix}.$$

The basis, B , of matrix A is the following:

$$B = \left\{ \begin{pmatrix} 3 \\ 2 \end{pmatrix} \begin{pmatrix} 3 \\ 0 \end{pmatrix} \begin{pmatrix} 2 \\ -2 \end{pmatrix} \right\}.$$

Fig. 2 **a** Finding the basis B for matrix A , **b** False paths due to linear combinations of basis B

Fig. 3 False path in the dependence graph

The linear combinations of vectors belonging to basis B with negative coefficients lead to false paths, for example, as presented below (see Fig. 2b).

$$0 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} + 1 \times \begin{pmatrix} 3 \\ 0 \end{pmatrix} - 1 \times \begin{pmatrix} 2 \\ -2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \Rightarrow \underline{\text{FALSE PATH}},$$

$$1 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} + 0 \times \begin{pmatrix} 3 \\ 0 \end{pmatrix} - 1 \times \begin{pmatrix} 2 \\ -2 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \end{pmatrix} \Rightarrow \underline{\text{FALSE PATH}}.$$

The second case takes place when on a path between x and y , being described by R_{kk}^* , there exists a vertex w such that $w \in \text{ran } R_{kk} \wedge w \notin \text{dom } R_{kk}$. An example is presented in Fig. 3, where $x_2 \in \text{ran } R_{kk} \wedge x_2 \notin \text{dom } R_{kk}$. Relation R_{kk}^* , built according to (7), describes the false path between x_1 and x_4 depicted by the dotted line.

Summing up, we conclude that relation R_{kk}^* describes all existing paths in the dependence graph represented by relation R_{kk} and can describe not existing paths, i.e., $(R_{kk}^*)_{\text{exact}} \subseteq R_{kk}^*$; when relation R_{kk}^* does not represent not existing paths, $R_{kk}^* = (R_{kk}^*)_{\text{exact}}$.

3.4 Time complexity

The first tree steps of Algorithm 2 can be accomplished in polynomial time.

1. As we have proved in Sect. 3.2, the task of replacing parameterized vectors with a linear combination of vectors with constant coordinates can be done in $O(d^2)$ operations.

Fig. 4 Reduced dependence graph for the illustrative example

2. The task of forming a dependence matrix using all k constant dependence vectors in $\mathbb{Z}^n$ requires $O(kn)$ operations (memory accesses).
3. The task of identifying a set of linearly independent columns of matrix A , $A \in \mathbb{Z}^{n \times k}$ with constant coordinates to find the basis can be done in polynomial time by the Gaussian elimination algorithm. According to [Cohen and Megiddo \(1991\)](#), this computation can be done in $O(ldk)$ arithmetic operations.

To calculate relation R_{kk}^* in step 4 of Algorithm 2, we use the Presburger arithmetic. In general, calculations based on the Presburger arithmetic are not characterized by polynomial time complexity [Kelly et al. \(1996\)](#).

3.5 Illustrating example

Let us illustrate Algorithms 1 and 2 by means of the following example.

```

for k=1 to N1
  for i=1 to N2
    S1 : A(k,i) = B(k-1)
        for j=1 to N3
          S2 : B(k) = B(k+2) + A(k-3,k)
        end
      end
    end
  end
end

```

The following relations describe dependences in the loop above.

```

R12 := { [k,k+3] -> [k+3,i',j] : 1<=k<=N2-3, N1-3
                                     & 1<=i'<=N2 & 1<=j<=N3 },
R21 := { [k,i,j] -> [k+1,i'] : 1<=k<N1 & 1<=i,i'<=N2
                                     & 1<=j<=N3 },
R22 := { [k,i,j] -> [k+2,i',j'] : 1<=k<=N1-2 & 1<=i,i'<=N2 &
                                     1<=j,j'<=N3 } union
        { [k,i,j] -> [k,i,j'] : 1<=j<j'<=N3 & 1<=k<=N1 &
                                     1<=i<=N2 } union
        { [k,i,j] -> [k,i',j'] : 1<=i<i'<=N2 & 1<=k<=N1 &
                                     1<=j,j'<=N3 }.

```

They form the reduced dependence graph presented in [Fig. 4](#).

Algorithm 1 calls Algorithm 2 to calculate relation R_{kk}^* for each iteration k of the outermost loop.

For $k = 1$, the dependence distance set $\Delta_{1,1} = \emptyset$ because $R_{11} = \emptyset$, so we get $R_{11}^* = \emptyset \cup I = \{[k, i] \rightarrow [k, i]\}$.

For $k = 2$, we get the following dependence relations:

$$\begin{aligned} R_{22} = & \{[k, i, j] \rightarrow [k+2, i', j'] : 1 \leq k \leq N1-2 \ \& \ 1 \leq i \leq N2 \ \& \\ & 1 \leq j \leq N3 \ \& \ 1 \leq j' \leq N3 \ \& \ 1 \leq i' \leq N2\} \text{ union} \\ & \{[k, i, j] \rightarrow [k, i, j'] : 1 \leq j < j' \leq N3 \ \& \ 1 \leq k \leq N1 \ \& \\ & 1 \leq i \leq N2\} \text{ union} \\ & \{[k, i, j] \rightarrow [k, i', j'] : 1 \leq i < i' \leq N2 \ \& \ 1 \leq k \leq N1 \ \& \\ & 1 \leq j \leq N3 \ \& \\ & \ \& \ 1 \leq j' \leq N3\} \text{ union} \\ & \{[k, i, j] \rightarrow [k+4, i', j'] : 1 \leq k \leq N2-4, \ N1-4 \ \& \ 1 \leq i \leq N2 \\ & \ \& \ 1 \leq j \leq N3 \ \& \ 1 \leq j' \leq N3 \ \& \ 1 \leq i' \leq N2\}, \end{aligned}$$

and yield the following dependence distance set $\Delta_{2,2}$

$$\Delta_{2,2} = \left\{ \begin{pmatrix} 2 \\ N2 \\ N3 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ N3 \end{pmatrix}, \begin{pmatrix} 0 \\ N2 \\ N3 \end{pmatrix}, \begin{pmatrix} 4 \\ N2 \\ N3 \end{pmatrix} \right\}.$$

Applying Algorithm 2, we yield the following results.

1. *Replace all parameterized dependence distance vectors.* The first parameterized

vector $\begin{pmatrix} 2 \\ N2 \\ N3 \end{pmatrix}$ is replaced with the linear combination of the vector $\begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$ and the

unit normal vectors $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ as follows:

$$\begin{pmatrix} 2 \\ N2 \\ N3 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} + N2 \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + N3 \times \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

The second parameterized vector $\begin{pmatrix} 0 \\ 0 \\ N3 \end{pmatrix}$ is replaced with the linear combination

of the vector $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ and the unit normal vector $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ as below:

$$\begin{pmatrix} 0 \\ 0 \\ N3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + N3 \times \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

The third parameterized vector $\begin{pmatrix} 0 \\ N2 \\ N3 \end{pmatrix}$ is replaced with the linear combination of the vector $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ and the unit normal vectors $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ as follows:

$$\begin{pmatrix} 0 \\ N2 \\ N3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + N2 \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + N3 \times \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

The last parameterized vector $\begin{pmatrix} 4 \\ N2 \\ N3 \end{pmatrix}$ is replaced with the linear combination of the vector $\begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix}$ and the unit normal vectors $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ as below:

$$\begin{pmatrix} 4 \\ N2 \\ N3 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix} + N2 \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + N3 \times \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

The resulting dependence distance set $\Delta_{2,2}$ contains the vectors with constant coordinates only:

$$\Delta_{2,2} = \left\{ \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix} \right\}.$$

2. *Form a dependence matrix.* The matrix A , where all the constant dependence vectors from set $\Delta_{2,2}$ are placed in columns, is as follows:

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

3. *Find the basis of the dependence distance set.* A set of linearly independent columns of matrix $A \in \mathbb{Z}^{n \times k}$ over field $\mathbb{Z}^n$, that can generate every vector in A , holds the following matrix B :

$$B = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

4. *Calculate the exact transitive closure of a dependence relation describing all the dependences in an input loop or its over-approximation, R_{22}^* .* Form relation R_{22}^* as follows:

$$\begin{aligned}
R_{22}^* := & \left\{ \begin{aligned} & [i, j, k] \rightarrow [i', j', k'] \mid \exists \lambda \text{ s.t. } \begin{pmatrix} i' \\ j' \\ k' \end{pmatrix} = \begin{pmatrix} i \\ j \\ k \end{pmatrix} + \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \lambda \wedge \\ & \wedge \begin{pmatrix} i' \\ j' \\ k' \end{pmatrix} - \begin{pmatrix} i \\ j \\ k \end{pmatrix} \geq 0, \lambda \in \mathbb{Z}^3 \wedge \\ & \wedge \begin{pmatrix} i' \\ j' \\ k' \end{pmatrix} \in \text{ran } R_{22} \wedge \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in \text{dom } R_{22} \end{aligned} \right\} = \\
& \left\{ [k, i, j] \rightarrow [k', i', j'] : \exists \alpha : 2\alpha = k + k' \wedge 1 \leq k \leq k' - 2 \wedge 1 \leq i \leq N2 \right. \\
& \quad \left. 1 \leq j, j' \leq N3 \wedge 1 \leq i' \leq N2 \wedge k' \leq N1 \right\} \cup \\
& \left\{ [k, i, j] \rightarrow [k, i, j'] : 1 \leq j < j' \leq N3 \wedge 1 \leq k \leq N1 \wedge 1 \leq i \leq N2 \right\} \cup \\
& \left\{ [k, i, j] \rightarrow [k, i', j'] : 1 \leq i < i' \leq N2 \wedge 1 \leq k \leq N1 \wedge 1 \leq j \leq N3 \wedge \right. \\
& \quad \left. 1 \leq j' \leq N3 \right\} \cup \\
& \left\{ [k, i, j] \rightarrow [k, i, j] \right\}
\end{aligned}$$

Relation R_{22}^* represents exact transitive closure since $R_{22}^+ = R_{22} \cup (R_{22} \circ R_{22}^+)$, i.e., $R_{22}^* = (R_{22}^+)^{exact}$.

The final result is the following

$$\begin{aligned}
D(1, 1) &= \{[k, k+3] \rightarrow [k', i'] : \text{Exists}(\alpha : 2\alpha = 4+k+k' \ \& \\ & \quad \& 1 \leq k \leq N2-3, k' \leq 6 \ \& 1 \leq i' \leq N2 \ \& k' \leq N1 \ \& 1 \leq N3)\} \text{ union} \\ & \quad \{[k, k+3] \rightarrow [k+4, i'] : 1 \leq k \leq N2-3, N1-4 \ \& 1 \leq i' \leq N2 \ \& 1 \leq N3\}, \\
D(1, 2) &= \{[k, k+3] \rightarrow [k+3, i', j] : 1 \leq k \leq N2-3, N1-3 \ \& 1 \leq i' \leq N2 \ \& \\ & \quad \& 1 \leq j \leq N3\} \text{ union} \\ & \quad \{[k, k+3] \rightarrow [k', i', j] : \text{Exists}(\alpha : 2\alpha = 1+k+k' \ \& \\ & \quad \& 1 \leq k \leq k'-5, N2-3 \ \& 1 \leq j \leq N3 \ \& 1 \leq i' \leq N2 \ \& k' \leq N1)\}, \\
D(2, 1) &= \{[k, i, j] \rightarrow [k', i'] : \text{Exists}(\alpha : 2\alpha = 1+k+k' \ \& \\ & \quad \& 1 \leq k \leq k'-3 \ \& 1 \leq i \leq N2 \ \& 1 \leq j \leq N3 \ \& 1 \leq i' \leq N2 \ \& \\ & \quad \& k' \leq N1)\} \text{ union} \\ & \quad \{[k, i, j] \rightarrow [k+1, i'] : 1 \leq k \leq N1 \ \& 1 \leq i \leq N2 \ \& 1 \leq j \leq N3 \ \& \\ & \quad \& 1 \leq i' \leq N2\}, \\
D(2, 2) &= \{[k, i, j] \rightarrow [k, i', j'] : 1 \leq i < i' \leq N2 \ \& 1 \leq k \leq N1 \ \& \\ & \quad \& 1 \leq j \leq N3 \ \& 1 \leq j' \leq N3\} \text{ union} \\ & \quad \{[k, i, j] \rightarrow [k, i, j'] : 1 \leq j < j' \leq N3 \ \& 1 \leq k \leq N1 \ \& \\ & \quad \& 1 \leq i \leq N2\} \text{ union} \\ & \quad \{[k, i, j] \rightarrow [k', i', j'] : \text{Exists}(\alpha : 2\alpha = k+k' \ \& \\ & \quad \& 1 \leq k \leq k'-2 \ \& 1 \leq i \leq N2 \ \& 1 \leq j \leq N3 \ \& 1 \leq i' \leq N2 \ \& \\ & \quad \& 1 \leq j' \leq N3 \ \& k' \leq N1)\} \text{ union} \\ & \quad \{[k, i, j] \rightarrow [k, i, j]\}.
\end{aligned}$$

4 Related work

Numerous algorithms for calculating the transitive closure of affine integer tuple relations have been proposed [Kelly et al. \(1996\)](#); [Beletskaya et al. \(2009\)](#); [Verdoolaege et al. \(2011\)](#); [Ancourt et al. \(2010\)](#); [Boigelot \(1998\)](#); [Bozga et al. \(2009\)](#); [Eve and](#)

Kurki-Suonio (1977). However, in most of them authors focus on relations whose domain and range are non-parametric polyhedra Ancourt et al. (2010); Bozga et al. (2009); Eve and Kurki-Suonio (1977). The limitation of many known algorithms is that they require that the arity of input and output tuples (the number of tuple elements) of relations has to be the same Beletska et al. (2009). This is why we limit related work only to techniques dealing with parameterized relations whose tuple arities are different in general and relations can describe dependences available in program loops.

On a different line of work, Bozga et al. Bozga et al. (2009) have studied the computation of transitive closure for the analysis of counter automata (register machines) and they have implemented their method in the tool called FLATA Bozga et al. (2009). In this context, relation $R(x, x')$ is a relation that can be written as the finite number of conjunctions of terms of the form $\pm x_i \pm x_j \leq a_{i,j}$, $\pm x'_i \pm x_j \leq b_{i,j}$, $\pm x_i \pm x'_j \leq c_{i,j}$, $\pm x'_i \pm x'_j \leq d_{i,j}$, $\pm 2x_i \leq e_{i,j}$ or $2x'_i \leq f_{i,j}$, where x and y describe counter values, either at the current step, or at the next step, $a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j}, e_{i,j}, f_{i,j} \in \mathbb{Z}$ are integer constants and $1 \leq i, j \leq n$, $i \neq j$. As we can see, this class of relations does not involve parameters, existentially quantified variables or unions, i.e., it cannot represent dependences in program loops. This is why we do not compare this technique with ours.

To our best knowledge, techniques for computing the transitive closure of parameterized affine integer tuple relations with different input and output arities of tuples were the subject of the investigation of a few papers only Kelly et al. (1996); Verdoolaege et al. (2011); Verdoolaege (2012). Kelly et al. Kelly et al. (1996) proposed a modified Floyd–Warshall algorithm but they have not implemented it in the Omega library (<http://www.cs.umd.edu/projects/omega/>). Fourteen years later Verdoolaege has improved and implemented his version of the Floyd–Warshall algorithm in the ISL library (<http://www.kotnet.org/skimo/isl/>), but that algorithm and implementation are not the same as ours.

Verdoolaege Verdoolaege et al. (2011); Verdoolaege (2012) treats each of input relations $R_{i \leq m}$ as vertices $V_{i \leq m}$ of the directed graph G , where m is the total number of input relations. There exists a directed path E_{ij} from vertex $V_{i \leq m}$ to vertex $V_{j \leq m}$ (R_j can immediately follow R_i) if the range of R_i intersects the domain of R_j , i.e., if

$$R_j \circ R_i \neq \emptyset. \quad (8)$$

In order to calculate the transitive closure of a dependence relation R , Verdoolaege Verdoolaege et al. (2011); Verdoolaege (2012) constructs m^2 relations

$$R_{ij} = \bigcup_{i, j \text{ s.t. } R_j \circ R_i \neq \emptyset}^m R_j \circ R_i. \quad (9)$$

Then he applies Algorithm 1 and returns the union of all output $R_{i,j}$ as transitive closure. In our implementation, we use information gathered with the Petit dependence analyzer Kelly et al. (1996) to insert a dependence relation describing dependences

between instances of statements i and j as element i, j of array D (element D_{ij}). Then we call Algorithm 1 to get transitive closure. Information provided with Petit permits us to reduce the time complexity of the Floyd–Warshall algorithm implementation due to skipping a connection check between each pair of input dependence relations (see formula 7).

Because of the differences between our implementation of the Floyd–Warshall algorithm and that of Verdoolaege [Verdoolaege et al. \(2011\)](#); [Verdoolaege \(2012\)](#), in this paper we investigate only how different concepts of calculating the relation R_{kk}^* impact the time complexity of the Floyd–Warshall algorithm. For this purpose, we have chosen for calculating R_{kk}^* algorithms implemented in the ISL (<http://www.kotnet.org/skimo/isl/>) and Omega (<http://www.cs.umd.edu/projects/omega/>) libraries. Those algorithms are based on computing parametric powers R^k and then projecting out the parameter k by making it existentially quantified. As a trivial example, consider the relation $R = \{[x] \rightarrow [x + 1]\}$. The k th power of R for arbitrary k is $R^k = \{[x] \rightarrow [x + k] \mid k \geq 1\}$ and the transitive closure is then $R^+ = \{[x] \rightarrow [y] \mid \exists k \in \mathbb{Z}_{\geq 1} : y = x + k\} = \{[x] \rightarrow [y] \mid y \geq x + 1\}$. Both the algorithms consider the difference set ΔR of the relation, but in the ISL library (<http://www.kotnet.org/skimo/isl/>) if all differences Δ_i s are singleton sets, i.e., $\Delta_i = \{\delta_i\}$ with $\delta_i \in \mathbb{Z}^d$, then R^+ is calculated as follows:

$$R^+ = \{x \longrightarrow y \mid \exists k_i \in \mathbb{Z}_{\geq 0} : y = x + \sum_i k_i \delta_i \wedge \sum_i k_i = k > 0\}, \quad (10)$$

which is essentially the same as that of Beletskaya et al. [Beletskaya et al. \(2009\)](#). If some of the Δ_i s are parametric, then each offset Δ_i is extended with an extra coordinate $\Delta'_i = \Delta_i \times \{1\}$, that is the constant equal to one. Paths constructed by summing such extended offsets are of length k encoded as the difference of their final coordinates, so R^+ can then be decomposed into relations R_i^+ , one for each Δ_i as follows

$$R^+ = ((R_m^+ \cup I) \circ \dots \circ (R_2^+ \cup I) \circ (R_1^+ \cup I)) \cap \{x' \rightarrow y' \mid \exists k_{>0} : y_{d+1} - x_{d+1} = k\}, \quad (11)$$

with

$$R_i^+ = s \mapsto \{x' \rightarrow y' \mid \exists k \in \mathbb{Z}_{\geq 1}, \delta \in k \Delta'_i(s) : y' = x' + \delta\}. \quad (12)$$

Each non-parametric constraint $A_1 x + c_1 \geq 0$ of $\Delta'_i(s)$ from (11) is transformed into the form $A_1 x + k c_1 \geq 0$ and the rest of constraints are rewritten without any changes. For more details see [Verdoolaege et al. \(2011\)](#); [Verdoolaege \(2012\)](#).

While the algorithms implemented by Verdoolaege [Verdoolaege et al. \(2011\)](#); [Verdoolaege \(2012\)](#) in the ISL library (<http://www.kotnet.org/skimo/isl/>) are designed to compute overapproximations, Kelly et al. [Kelly et al. \(1996\)](#) in the Omega library (<http://www.cs.umd.edu/projects/omega/>) propose a heuristic algorithm to compute an under-approximation that does not guarantee calculating exact transitive closure.

5 Experimental results

The goals of experiments were to evaluate the effectiveness and time complexity of the proposed approach for calculating relation R_{kk}^* and using it in the modified Floyd–Warshall algorithm for loops provided by the well-known NAS Parallel Benchmark (NPB) Suite from NASA (<http://www.nas.nasa.gov>) and compare received results with the effectiveness and time complexity of techniques implemented in the ISL (<http://www.kotnet.org/skimo/isl/>) and Omega (<http://www.cs.umd.edu/projects/omega/>) tools. We have implemented the presented algorithms as an ANSI-C++ software module. The source code of the module was compiled using the gcc compiler v4.3.0 and can be download from: <http://www.sfs.zut.edu.pl/files/mfw-omega.tar.gz>. Experiments were conducted using an Intel Core2Duo T7300@2.00GHz machine with the Fedora Linux v12 32bit operating system.

The implementation calculates transitive closure according to Algorithm 1 and permits to choose the three options for producing the relation R_{kk}^* by means of: (i) Algorithm 2, (ii) Omega, and (iii) ISL. Under our experiments, we have examined only such loops provided by NPB that expose dependences. There exist 75 imperfectly nested loops and 58 perfectly nested loops in NPB that expose dependences. The results of our experiments are collected in Table 1, where time is presented in seconds. The columns “Proposed algorithm”, “ISL”, and “Omega” present the time of calculating transitive closure by means of the Floyd–Warshall algorithm, where relations R_{kk}^* were calculated by means of applying Algorithm 2, the ISL, and Omega tools, respectively.

Analyzing the results presented in Table 1, we can derive the following conclusions. All techniques under experiments are able to calculate transitive closure for all NBP loops exposing dependences. The exactness of the presented approach is the same as that of techniques implemented in Omega and ISL. i.e., all techniques under experiments produce exact transitive closure for the same loops. Calculating relation R_{kk}^* by means of Algorithm 2 is less time-consuming in comparison with techniques implemented in Omega and ISL that reduces the time of calculating the transitive closure of a relation describing all the dependences in the loop by means of the Floyd–Warshall’s algorithm. For all loops, we obtained the shortest time of producing transitive closure.

The explanation is that each relation R_{kk}^* that we compose in Algorithm 1 (line 10) consists of two relations, $R_{kk}^+ \cup I$. If there are m disjuncts in the input relation, R_{kk} , then the direct application of the composition operation just like in formula (11) may therefore result in a relation with 2^m disjuncts that is computationally expensive. In general, applying formula (7) results in the number of disjuncts that is much less than 2^m . This permits us to conclude that the presented approach is faster than other well-known approaches.

Table 1 The results of the experiments on the proposed approach to computing transitive closure

Source loop name	Number of relations	Proposed algorithm		ISL ¹		Omega ²	
		ex	<i>t</i> [s]	ex	Δt [s]	ex	Δt [s]
<i>Perfectly-nested loops</i>							
BT_error.f2p_5	31	1	0.2451	1	1.1081	1	2.4072
BT_initialize.f2p_8	3	1	0.0006	1	0.0017	1	0.0040
BT_initialize.f2p_9	1	1	0.0006	1	0.0009	1	0.0015
BT_rhs.f2p_1	46	1	2.1681	1	3.5402	1	4.1763
BT_rhs.f2p_5	128	1	1.3442	1	16.1829	1	4.1989
CG_cg.f2p_3	1	1	0.0007	1	0.0018	1	0.0014
CG_cg.f2p_4	10	1	0.0035	1	0.0061	1	0.0225
CG_cg.f2p_6	1	1	0.0004	1	0.0009	1	0.0019
CG_cg.f2p_8	1	1	0.0006	1	0.0013	1	0.0013
FT_auxfnct.f2p_1	1	1	0.0006	1	0.0006	1	0.0029
FT_auxfnct.f2p_2	1	1	0.0004	1	0.0005	1	0.0020
LU_HP_l2norm.f2p_2	9	1	0.0451	1	0.0829	1	1.1249
LU_HP_jacld.f2p_1	2634	1	74.0127	1	75.1268	1	120.5502
LU_HP_jacu.f2p_1	2634	1	74.0840	1	75.0540	1	108.8616
LU_HP_pintgr.f2p_11	4	1	0.0111	1	0.0170	1	0.0516
LU_HP_pintgr.f2p_2	109	1	0.1409	1	0.4151	1	0.3499
LU_HP_pintgr.f2p_3	6	1	0.0103	1	0.0167	1	0.0669
LU_HP_pintgr.f2p_7	6	1	0.0107	1	0.0195	1	0.0504
LU_jacld.f2p_1	2594	1	397.1930	1	538.9371	1	446.8944
LU_jacu.f2p_1	2594	1	397.0249	1	543.8912	1	421.9345
LU_l2norm.f2p_2	9	1	0.0445	1	0.0560	1	0.4220
LU_pintgr.f2p_11	6	1	0.0117	1	0.0165	1	0.0487
LU_pintgr.f2p_2	109	1	0.1405	1	0.4151	1	0.3238
LU_pintgr.f2p_3	6	1	0.0114	1	0.0233	1	0.0667
LU_pintgr.f2p_7	6	1	0.0118	1	0.0166	1	0.0602
MG_mg.f2p_1	1	1	0.0008	1	0.0058	1	0.0019
MG_mg.f2p_11	1	1	0.0006	1	0.0009	1	0.0027
MG_mg.f2p_12	1	1	0.0005	1	0.0006	1	0.0014
MG_mg.f2p_13	1	1	0.0001	1	0.0001	1	0.0019
MG_mg.f2p_4	1	1	0.0001	1	0.0001	1	0.0009
SP_error.f2p_5	31	1	0.1996	1	0.9823	1	2.4040
SP_initialize.f2p_8	3	1	0.0001	1	0.0002	1	0.0004
SP_ninvr.f2p_1	103	1	3.5246	1	26.9442	1	8.5589
SP_pinvr.f2p_1	103	1	3.5322	1	27.0151	1	8.6393
SP_rhs.f2p_1	64	1	4.8383	1	51.3303	1	8.2041
SP_rhs.f2p_5	127	1	1.3437	1	16.7552	1	4.2837

Table 1 continued

Source loop name	Number of relations	Proposed algorithm		ISL ^a		Omega ^b	
		ex	t [s]	ex	Δt [s]	ex	Δt [s]
SP_txinvr.f2p_1	271	1	64.3626	1	328.6192	1	75.9909
SP_tzetar.f2p_1	288	1	51.7971	1	269.7633	1	63.2336
UA_adapt.f2p_2	8	1	0.0516	1	0.0760	1	0.2282
UA_diffuse.f2p_1	5	1	0.0787	1	0.1692	1	94.0265
UA_diffuse.f2p_2	3	1	0.0010	1	0.0021	1	0.0029
UA_mason.f2p_18	1	1	0.0011	1	0.0013	1	0.0057
UA_precond.f2p_5	30	0	0.5973	0	3.0276	0	9.8670
UA_setup.f2p_1	1	1	0.0002	1	0.0003	1	0.0008
UA_setup.f2p_16	3	1	0.0014	1	0.0029	1	0.0111
UA_setup.f2p_6	4	1	0.0039	1	0.0052	1	0.1116
UA_transfer.f2p_1	1	1	0.0003	1	0.0006	1	0.0009
UA_transfer.f2p_10	1	1	0.0012	1	0.0015	1	0.0019
UA_transfer.f2p_13	1	1	0.0049	1	0.0013	1	0.0008
UA_transfer.f2p_15	1	1	0.0006	1	0.0016	1	0.0009
UA_transfer.f2p_18	1	1	0.0004	1	0.0014	1	0.0009
UA_transfer.f2p_2	1	1	0.0032	1	0.0065	1	0.0009
UA_transfer.f2p_3	1	1	0.0012	1	0.0012	1	0.0013
UA_transfer.f2p_5	1	1	0.0031	1	0.0046	1	0.0065
UA_transfer.f2p_6	1	1	0.0032	1	0.0037	1	0.0057
UA_transfer.f2p_7	1	1	0.0024	1	0.0029	1	0.0081
UA_transfer.f2p_8	1	1	0.0017	1	0.0035	1	0.0058
UA_transfer.f2p_9	1	1	0.0040	1	0.0015	1	0.0099
<i>Imperfectly-nested loops</i>							
BT_error.f2p_2	107	1	2.9712	1	8.8938	1	9.1521
BT_error.f2p_3	6	1	0.0038	1	0.0069	1	0.0071
BT_error.f2p_6.t	6	1	0.0036	1	0.0082	1	0.0069
BT_exact_rhs.f2p_2	1553	0	32.2311	0	61.3898	0	73.3091
BT_exact_rhs.f2p_3	1553	0	31.9385	0	68.5866	0	74.3960
BT_exact_rhs.f2p_4	1553	0	32.1856	0	61.1335	0	73.9969
BT_initialize.f2p_2	42	1	0.2836	1	0.3948	1	3.0076
BT_initialize.f2p_3	42	1	0.2888	1	0.3964	1	2.9983
BT_initialize.f2p_4	42	1	0.2882	1	0.3934	1	3.0522
BT_initialize.f2p_5	42	1	0.3139	1	0.4283	1	3.0035
BT_initialize.f2p_6	42	1	0.3181	1	0.3944	1	3.0082
BT_initialize.f2p_7	42	1	0.2986	1	0.3966	1	3.0189
BT_rhs.f2p_3	702	0	26.1909	0	268.7525	0	38.5441
BT_rhs.f2p_4	510	0	16.4426	0	236.8264	0	26.1494
CG_cg.f2p_7	2	1	0.0017	1	0.0022	1	0.0317

Table 1 continued

Source loop name	Number of relations	Proposed algorithm		ISL ^a		Omega ^b	
		ex	t [s]	ex	Δt [s]	ex	Δt [s]
LU_blts.f2p_1	4885	1	3632.8071	1	4267.0205	1	5078.6317
LU_buts.f2p_1	5640	1	4010.8654	1	5673.0981	1	5612.8839
LU_erhs.f2p_2	66	1	0.1669	1	0.5987	1	5.9636
LU_erhs.f2p_3	640	0	72.3339	0	164.4464	0	107.5848
LU_erhs.f2p_4	640	0	74.6972	0	192.2292	0	104.2774
LU_erhs.f2p_5	640	0	32.5497	0	237.4519	0	58.9116
LU_HP_blts.f2p_1	3232	0	216.5695	0	216.8695	0	218.7895
LU_HP_buts.f2p_1	3593	0	250.4280	0	447.2031	0	267.1930
LU_HP_erhs.f2p_2	66	1	0.1640	1	0.8398	1	6.4393
LU_HP_erhs.f2p_3	640	0	72.5601	0	263.6080	0	115.7859
LU_HP_erhs.f2p_4	640	0	74.5099	0	262.0617	0	116.0602
LU_HP_erhs.f2p_5	640	0	32.9287	0	236.9649	0	57.8919
LU_HP_rhs.f2p_1	17	1	0.2142	1	1.5149	1	1.2455
LU_HP_rhs.f2p_2	640	0	72.5522	0	387.5539	0	115.3880
LU_HP_rhs.f2p_3	640	0	74.3030	0	262.0265	0	115.4032
LU_HP_rhs.f2p_4	640	0	32.4699	0	237.7602	0	57.5956
LU_rhs.f2p_1	17	1	0.2175	1	1.5029	1	1.2170
LU_rhs.f2p_2	640	0	71.9027	0	279.4004	0	115.5498
LU_rhs.f2p_3	640	0	73.6644	0	277.5648	0	114.4854
LU_rhs.f2p_4	1412	0	199.7893	0	968.4744	0	354.9285
MG_mg.f2p_10	18	1	0.0041	1	0.0043	1	0.0047
MG_mg.f2p_3	3	1	0.0001	1	0.0002	1	0.0001
MG_mg.f2p_5	24	0	0.6285	0	0.7923	0	2.1224
MG_mg.f2p_6	29	0	0.9173	0	0.9739	0	2.1649
MG_mg.f2p_7	510	1	2.0639	1	17.9808	1	5.5680
MG_mg.f2p_8	55	0	2.2999	0	2.3069	0	7.1395
MG_mg.f2p_9	18	1	0.0036	1	0.0043	1	0.0047
SP_error.f2p_2	107	1	2.4962	1	9.2583	1	9.1577
SP_error.f2p_3	6	1	0.0039	1	0.0043	1	0.0081
SP_error.f2p_6	6	1	0.0013	1	0.0014	1	0.0073
SP_exact_rhs.f2p_2	1553	0	32.0930	0	97.8048	0	81.8412
SP_exact_rhs.f2p_3	1553	0	32.1471	0	106.6423	0	81.0354
SP_exact_rhs.f2p_4	1553	0	32.2977	0	102.4652	0	81.5785
SP_initialize.f2p_2	24	1	0.2242	1	0.5455	1	3.0368
SP_initialize.f2p_3	24	1	0.2234	1	0.3989	1	3.1722
SP_initialize.f2p_4	24	1	0.2214	1	0.3971	1	3.0657
SP_initialize.f2p_5	24	1	0.2239	1	0.3943	1	3.0336
SP_initialize.f2p_6	24	1	0.2216	1	0.4103	1	3.0342

Table 1 continued

Source loop name	Number of relations	Proposed algorithm		ISL ^a		Omega ^b	
		ex	t [s]	ex	Δt [s]	ex	Δt [s]
SP_initialize.f2p_7	24	1	0.2227	1	0.3936	1	3.0376
SP_rhs.f2p_3	699	1	10.7808	1	231.7330	1	20.0821
SP_rhs.f2p_4	507	1	14.1710	1	156.5537	1	23.3080
UA_adapt.f2p_1	10	1	0.0469	1	0.0640	1	0.0930
UA_adapt.f2p_10	14	1	0.0136	1	0.0164	1	0.0264
UA_adapt.f2p_11	11	1	0.0134	1	0.0162	1	0.0306
UA_adapt.f2p_9	14	1	0.0058	1	0.0163	1	0.0256
UA_diffuse.f2p_3	1	1	0.0004	1	0.0005	1	0.0009
UA_diffuse.f2p_4	1	1	0.0015	1	0.0042	1	0.0147
UA_diffuse.f2p_5	1	1	0.0013	1	0.0039	1	0.0018
UA_precond.f2p_3	1	1	0.0009	1	0.0016	1	0.0017
UA_precond.f2p_4	1	1	0.0003	1	0.0003	1	0.0005
UA_setup.f2p_14	31	1	0.2973	1	0.9474	1	0.3562
UA_setup.f2p_15	15	1	0.2610	1	0.3649	1	0.2836
UA_transfer.f2p_11	6	1	0.0048	1	0.0049	1	0.0260
UA_transfer.f2p_12	7	1	0.0146	1	0.0044	1	0.0249
UA_transfer.f2p_14	8	1	0.0240	1	0.0373	1	0.0780
UA_transfer.f2p_16	4	1	0.0018	1	0.0033	1	0.0150
UA_transfer.f2p_17	17	1	0.0238	1	0.0354	1	0.0867
UA_transfer.f2p_19	4	1	0.0030	1	0.0033	1	0.0122
UA_transfer.f2p_4	3	1	0.0047	1	0.0018	1	0.0132
UA_utils.f2p_12	20	1	0.1816	1	0.1635	1	0.7387

(ex: 1—exact result, 0—over-approximation; Δt : difference between the transitive closure calculation time of a known correspondent technique and that of the presented approach)

^a Integer Set Library—a library for manipulating sets and relations of integer points bounded by affine constraints (available at <http://repo.or.cz/w/isl.git>)

^b Omega Project—frameworks and algorithms for the analysis and transformation of scientific programs (available at <http://www.cs.umd.edu/projects/omega/>)

6 Conclusion

In this paper, we presented a modified Floyd–Warshall algorithm, where the most time consuming part (transitive closure describing self-dependences in the program loop) is calculated by means of basis dependence distance vectors. We demonstrated how to calculate basis dependence distance vectors for parameterized program loops and how to apply them to calculate the transitive closure of a dependence relation describing all self-dependences among the instances of a given loop statement by means of basis dependence distance vectors.

This solution results in reducing the time of the transitive closure calculation of parameterized graphs representing dependences in program loops. Reducing this time is due to using a finite integral linear combination of basis dependence distance vectors

to calculate the R_{kk}^* term in a modified Floyd–Warshall algorithm. Reducing the time of the transitive closure calculation was proved by means of numerous experiments with NPB benchmarks. The presented approach can be used for resolving many optimizing compilers problems: redundant synchronization removal (Presburger 1927), testing the legality of iteration reordering transformations (Presburger 1927), iteration space slicing (Beletska et al. 2011), forming schedules for statement instances of program loops (Bielecki et al. 2012). In our future work we plan to study the application of the presented approach for extracting both coarse- and fine-grained parallelism for different popular benchmarks.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

- Ancourt C, Coelho F, Irigoin F (2010) A modular static analysis approach to affine loop invariants detection. *Electron Notes Theor Comput Sci* 267:3–16
- Beletska A, Barthou D, Bielecki W, Cohen A (2009) Computing the transitive closure of a union of affine integer tuple relations. In: COCOA09. Springer, Berlin, pp. 98–109
- Beletska A, Bielecki W, Cohen A, Palkowski M, Siedlecki K (2011) Coarse-grained loop parallelization: iteration space slicing vs affine transformations. *Parallel Comput* 37(8):479–497
- Bielecki W, Kraska K, Klimek T (2013) Transitive closure of a union of dependence relations for parameterized perfectly-nested loops. In: Malyshkin V et al. (eds) PaCT-2013, LNCS, vol 7979. Springer, Heidelberg, pp 37–50
- Bielecki W, Palkowski M, Klimek T (2012) Free scheduling for statement instances of parameterized arbitrarily nested affine loops. *Parallel Comput* 38:518–532. <http://dx.doi.org/10.1016/j.parco.2012.06.001>
- Boigelot B (1998) Symbolic methods for exploring infinite state spaces. Ph.D. thesis, Université de Liège
- Bozga M, Girlea C, Iosif R (2009) Iterating octagons. ETAPS 2009, TACAS'09. Springer, New York, pp 337–351
- Cohen E, Megiddo N (1991) Recognizing properties of periodic graphs, DIMACS series in discrete mathematics and theoretical computer science, vol 4. American Mathematical Society, pp 135–146
- Deng X, Dymond P (1998) On multiprocessor system scheduling. *J Comb Optim* 1(4):377–392
- Diestel R (2010) Graph theory, 4th edn. Springer, New York
- Eve J, Kurki-Suonio R (1977) On computing the transitive closure of a relation. *Acta Informatica* 25. X 8(4):303–314
- Feautrier P (2012) Approximating the transitive closure of a Boolean-Affine relation, IMPACT 2012 second international workshop on polyhedral compilation techniques, Paris, France, <http://impact.gforge.inria.fr/impact2012/>
- Griebel M (2004) Automatic parallelization of loop programs for distributed memory architectures. Fakultät für Mathematik und Informatik Universität Passau, Habilitation
- Hollermann L, Tsan-sheng H, Lopez D, Vertanen K (1997) Scheduling problems in a practical allocation model. *J Comb Optim* 1(2):129–149
- Integer Set Library, <http://www.kotnet.org/skimo/isl/>
- Kelly W, Maslov V, Pugh W, Rosser E, Shpeisman T, Wonnacott D (1996) New User Interface for Petit and Other Extensions. User Guide
- Kelly W, Pugh W, Rosser E, Shpeisman T (1996) Transitive closure of infinite graphs and its applications. LCPC'95, Columbus, Ohio, vol 1033. Springer, New York, pp 126–140
- NASA Advanced Supercomputing Division, <http://www.nas.nasa.gov>
- Omega Library, <http://www.cs.umd.edu/projects/omega/>

- Presburger M (1927) Über de vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchen, die addition als einzige operation hervortritt. In: Comptes Rendus du Premier Congrès des Mathématicienes des Pays Slaves, 395, Warsaw, pp 92–101
- Rotman J (2003) Advanced modern algebra, 2nd edn. Prentice Hall, Upper Saddle River
- Schrijver A (1999) Theory of linear and integer programming. Series in Discrete Mathematics
- Shoup V (2005) A computational Introduction to number theory. Cambridge University Press, New York
- Skiena S (2008) The algorithm design manual, 2nd edn. Springer, Berlin
- Verdoolaege S (2012) Integer set library—manual, Tech. rep. 2012, Version: isl-0.11 www.kotnet.org/skimo/isl/manual
- Verdoolaege S, Cohen A, Beletskaya A (2011) Transitive closures of affine integer tuple relations and their overapproximations. SAS, pp 216–232