DIV, FLOOR, CEIL, MOD AND STEP FUNCTIONS IN NESTED LOOP PROGRAMS AND LINEARLY BOUNDED LATTICES

P.C. HELD and A.C.J. KIENHUIS
Delft University of Technology
Mekelweg 4
2600 GA Delft
The Netherlands
held@dutentb.et.tudelft.nl

ABSTRACT. This paper describes the conversion of nested loop programs into single assignment forms. The nested loop programs may contain the integer operators: *integer division*, *floor*, *ceil*, and *modulo*, in expressions and the stride, or step size, of for loops may be greater than one. The programs may be parametrized but must have static control. The conversion is done completely automatical by the tool $HiPars$. The output of HiPars is a *single assignment program* (SAP) and a dependence graph (DG). The description of the dependence graph is based on linearly bounded lattices. We will show the relation between the integer division operators in the SAP and linearly bounded lattices in the corresponding DG.

KEYWORDS. Nested loop programs, data dependencies, single assignment programs, linearly bounded lattices.

1 INTRODUCTION

Many algorithms in the field of signal-processing are available in the form of sequential programs in various programming languages such as Fortran and C.
To execute programs on dedicated parallel processor arrays [13][4], we need to know the data dependencies between the operations of the program [1].
We restrict ourselves to programs that belong to the class of nested loop programs. In addition, we require that the programs have static control and that the expressions inside the program are linear expressions. For the data dependence analysis, a linear programming technique (PIP = parametric integer programming) due to Feautrier can be used [5] [6]. We show that the class of nested loop programs which contains non-linear functions *integer division*, *modulo*, *ceil* and *floor* in its expressions can be converted to the previous. Based on the definition of integer division, we can express these operators in terms of linear control statements, allowing us to use PIP for the analysis of these programs too. The result of the dependence analysis is a single assignment program (SAP). Our goal is to write this SAP in the form of a piecewise regular dependence graph. The description of the nodes and edges

of the dependence graph is based on linearly bounded lattices [12]. We pay special attention to the problem of converting the integer division operators that occur in the SAP into lattice descriptions.

2 DEPENDENCE ANALYSIS OF NESTED LOOP PROGRAMS

We have implemented a tool, called $HiPars$, which finds the data dependencies inside nested loop programs (NLP) [7]. We allow these NLPs to contain two kinds of control statements. The for-loop control statement and the conditional statement. In addition, the NLPs contain assignment statements, which take the form of function calls. We restrict ourselves to the class of *affine* nested loop programs [4]. Some properties of affine nested loop programs are, among others: (1) expressions of loop bounds, conditionals, and indices of variables are affine; (2) the programs have static control; (3) the number of iterations of the program is independent on the value of the variables of the program; (4) the programs may be parametrized.

We will extend this class of programs to include nested loop programs containing the non-linear functions: *integer division*, *ceil*, *floor*, and *modulo*. The functions may appear in the expressions of the program. In addition we allow that the stride, or step size, of a for-loop to be greater than one.

Let a be an affine expression of variables with integral coefficients and b an integral constant. We denote integer division by $div(a, b)$. We define the other operators in terms of the div operator, with $\frac{a}{b}$ standing for the division of a and b:

- $floor(\frac{a}{b}) = div(a, b)$
  rounds the result of $\frac{a}{b}$ to the greatest integer smaller than or equal to $\frac{a}{b}$.
- $ceil(\frac{a}{b}) = -div(-a, b)$
  rounds the result of $\frac{a}{b}$ to the smallest integer greater than or equal to $\frac{a}{b}$.
- $mod(a, b) = a - b * div(a, b)$
  returns the value of the remainder of the integer division.

With these definitions, we rewrite nested loop programs containing these functions in terms of the div functions only.

Below we have listed program 2.1 containing two for-loops with stride two and div functions inside the expressions of the conditional statements.

Two functions A and B are *data-dependent* if function A uses as argument a variable which value is defined by the evaluation of function B. To find the data dependencies, we represent the functions evaluated by the nested loop program as sets of iterations. We call these sets of iterations *iteration-domains*.

Program 2.1

Let M be a parameter.
Let `funcA` and `funcB` be two functions.
Let `a` be a two dimensional variable array.

```
for i = 1 to  M step 2,
   for j = 1 to 2 M step 2,
        if  i - 3 * div(i,3) <= 0 then
```

```
                [a(i,j)] = funcA( );
        end
        if j - 3* div(j,3) <= 0 then
                funcB( a(i,j) );
        end
   end
end
```

□

The kernel of HiPars is formed by a *parametric integer programming* routine (PIP). PIP differs from classical ILP in two ways. First the constant vector of the LP system may be parametrized leading to parametrized output by symbolic evaluation. Secondly, the objective function is defined by the lexicographical ordering of the feasible solutions.
In order to use PIP [5] for our dependence analysis we describe iteration domains by *polytopes*. Let **Z** be the set of integers. We define an *integral polytope* as a bounded set of points taking values in $\mathbf{Z}^n$ specified by a system of linear inequalities. By definition, a polytope is a dense space. When a program contains for-loops with stride other than one, the expressions of the lower and upper bounds are not sufficient to describe the values that the iterator takes on. We have to add an additional constraint specifying that the difference between values of the iterator is a multiple of the stride.

To model the stride, we introduce an integral variable q. Let lb be the lower- and ub be the upper-bound expression of the for-statement with iterator i and stride s. We model the stride by the equation:

$$i = q * s + lb \tag{1}$$

with $lb \leq i \leq ub$.
It is easy to show, that this equation together with the inequalities of the bounds form a polytope in the (i, q) space that define the values of iterator i. So we properly transformed the non-dense iteration domain in i into a dense iteration domain in a higher dimensional space at the cost of an additional variable in the domain description.

The outline of the sequel of the paper is as follows. In section 3, we will give the definition of integer division and substitute these operators inside NLPs by new control variables such that the resulting program contains only affine expressions. In section 4, we explain how a nested loop program is converted into a single assignment program and show that the SAP will contain additional control variables standing for integer divisions. Then our goal will be to write the SAP description as a DG with linearly bounded lattices. We will define linearly bounded lattices in section 5. In section 6 and section 7 we will derive the lattice vectors and lattice offsets, respectively. The procedure will be illustrated by an example.

3 INTEGER DIVISION

When expressions of bounds or conditionals contain an *integer division* operator, we have to do a similar transformation as we did for the stride in order to obtain polytope descriptions of iteration domains. For this purpose, we look at the definition of *integer division*.

Definition 3.1

Let a be an integer and b a positive integer. The result of *integer division* of a and b are integers q

and r, such that

$$a = b * q + r \quad (2)$$
$$0 \leq r \leq (b-1) \quad (3)$$

□

We call b the *divisor* and r the *remainder* of the division. The value of q is equal to $div(a, b)$. Observe that $div(5+3, 2) \neq div(5, 2) + div(3, 2)$, which shows that integer division is not a linear operation.

Definition 3.1 gives us a way to define the integer division operator $div(a, b)$ by two linear inequalities [9]. We write r as $a - b * q$ according to the equation and substitute it in the inequalities, resulting in:

$$0 \leq a - b * q \leq (b-1) \quad (4)$$

In the expressions inside the nested loop programs, we substitute each div operator by a control variable q and add the two inequalities defining variable q in the form of conditional statements.

After substitution of the div functions, the NLP contains only affine expressions. As a result, we can define all iteration domains by polytopes.

Example 3.1

Program 2.1 contains the conditional statement:

```
if i - 3* div(i,3) <= 0 then
```

We introduce variable q which we substitute for the function $div(i, 3)$ in the expression. By definition 3.1 we define $div(i, 3)$ by the inequalities:

$$0 \leq i - 3q \leq 2 \quad (5)$$

We add the inequalities as conditional statements before the original if-statement, resulting in the following statements:

```
if 0 <= i -3*q then
   if i - 3*q <= 2 then
     if i - 3* q <= 0 then
```

Now all inequalities are linear expressions of the variables i and q.

□

4 SINGLE ASSIGNMENT PROGRAM

After substitution of div operators by inequalities inside the NLP, we find the data dependencies between the variables of the program. $HiPars$ presents the result of the dependence analysis in the form of a single assignment program (SAP)[14] [3].

A complete dependence analysis of the program involves finding the dependencies for all right-hand side (RHS) variables appearing in the function call statements. A RHS variable can only be dependent on a left-hand side (LHS) variable of the same name. In case there are more LHS variables of the same name, we find first the solution between the RHS variable with each of the LHS variables

separately. This is done by PIP, which returns the solution in the form of the index of the LHS variable and the iteration-domain for which the solution is valid [5] [6]. The index may be undefined, which means that the RHS variable does not depend on the LHS variable. After applying PIP for all the LHS variables, we determine the lexicographical largest index among the indices, which is the *dependency*. Dependencies are linear functions on the iterators, parameters and variables standing for the integer divisions.

The complete solution of the dependence analysis of a single RHS variable may consists of multiple dependencies defined on mutually exclusive iteration domains [8].

The procedure to construct the SAP of the NLP is straightforward [3]. First we substitute LHS-variables by variables with unique names and with identity functions as indexing functions. Next we replace each RHS variables by the corresponding LHS variable in which the dependency is used as indexing function. If the dependency is undefined, we do not substitute the RHS variable. The iteration domains belonging to the dependencies are inserted in the SAP in the form of conditional statements.

Below we show the SAP of example 2.1 that is automatically generated by $HiPars$.

Example 4.1

Let `M` be a parameter.
Let `funcA` and `funcB` be two functions.
Let `a_1` be a two dimensional variable and `in0` a temp variable.

```
% SAP Generated By HiPars Version 2.08

for i=1 to  M step 2,
  for j=1 to M step 2,

    q1=div(i,3);
    if -i+3*q1>=0 then

      [ a_1( i,j ) ] = funcA( );

    end

    q2=div(j,3);
    if -j+3*q2>=0,

      q1=div(i+1,2);
      if -i+2*q1-1>=0 then
        q2=div(j+1,2);
        if -j+2*q2-1>=0 then
          q3=div(2*q1+2,3);
          if -i+3*q3-3 >=0 then
             q4 = div(q3,2);

             in0 = a_1(i,j);
```

```
          else
             in0 = a(i,j);
          end
        else
          in0 = a(i,j);
        end
      else
           in0 = a(i,j);
      end

      funcB( in0 );

    end
  end
end
```

□

Observe that the SAP contains additional div operators. The div functions are introduced by PIP during the process of finding a solution of the integer programming problem. To find an integral solution, PIP adds extra inequalities, called *cut planes*, to the polytope. The construction of the cut plane involves integer division. If parameters are involved in the division, PIP introduces a new parameter in order to linearize the cut plane [2] [5] [10]. These new parameters corresponds to the div functions in the SAP.

5 LINEARLY BOUNDED LATTICE

The SAP is in fact an intermediate format in the *HiFi* design system [13]. Our goal is to represent the NLP as a dependence graph (DG). Nodes of the DG represent functions of the DG and edges data dependencies between the functions. We use domains to represent the elements of the graph in a reduced way. We define domains as *linearly bounded lattices* [12][4].

A domain is specified by a polytope and a lattice. Let I be an index vector of length n and let K be a vector of m integral variables. With $A \in \mathbf{Z}^{m \times n}$ an integral matrix and $B \in \mathbf{Z}^m$ a constant vector, we define a polytope by [10]:

$$AK \quad \geq B \tag{6}$$

Now, with L an $n \times m$ integral matrix and $O \in \mathbf{Z}^n$ an integral vector, we define a lattice as:

$$I \quad = \quad LK + O \tag{7}$$

We say, that the lattice of I is generated by the columns of L, with the variables of K bounded by the polytope. We call O the *offset* of the lattice and the columns of L the *lattice vectors*.

An example of a two dimensional lattice is:

Example 5.1

$$\begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = \begin{pmatrix} 6 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \end{pmatrix} + \begin{pmatrix} 3 \\ -1 \end{pmatrix} \tag{8}$$

□

Below we will show how to write the SAP as linearly bounded lattices. We will focus on the lattice specification defined by the div functions inside the SAP.

6 INDEX DOMAINS WITH DIVS

In this section we will show the relation between lattices and the integer division functions.

To illustrate the subject, we take as example the piece of code of program 4.1 containing four div statements:

Example 6.1

q1 = div$(i+1, 2)$
if $-i + 2 * q1 - 1 \geq 0$
  q2=div$(j+1, 2)$
  if $-j + 2 * q2 - 1 \geq 0$
    q3=div$(2 * q1 + 2, 3)$
    if $-i + 3 * q3 - 3 \geq 0$
      q4= div$(q3, 2)$
          $a_1(3 * q3 - 3, 2 * q2 - 1)$

□

The example shows that div functions may be nested. The div functions and inequalities of the example form a part of specification of the iteration domain of the dependency for variable a_1. Our goal is to describe iteration domains of variables as linearly bounded lattices. The method is based on the so-called *hermite normal decomposition* [10]. Other approaches can be found in [9] [11].

To find the lattice defined by the integer divisions and inequalities involving the q's, we start by writing the div's as equations by setting the remainders r to zero.

Let N be a matrix of which the rows are the normals of these equations. Let $Q = (q_1, .., q_m)$ be the vector of variables of the m divisions and let I be the vector of the iterators. We write the system of equations defined by the div's as:

$$N \begin{pmatrix} I \\ Q \end{pmatrix} = 0$$

Example 6.2

With $I = (i, j)^t$ and $Q = (q_1, q_2, q_3, q_4)^t$, the system of equations of example 6.1 is:

$$\begin{aligned} i - 2q_1 &= 0 \\ j - 2 * q_2 &= 0 \\ 2q_2 - 3q_3 &= 0 \\ q_3 - 2q_4 &= 0 \end{aligned}$$

Thus matrix N is

$$N = \begin{pmatrix} 1 & 0 & -2 & 0 & 0 & 0 \\ 0 & 1 & 0 & -2 & 0 & 0 \\ 0 & 0 & 2 & 0 & -3 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix}$$

□

We assume that the system has a solution. Otherwise, we would have removed this piece of code from the program by dead code elimination procedures.

The system has m equations in $n + m$ variables. Because each row k introduces variable q_k it follows that the rows of N are independent. The nullspace of the system is thus n-dimensional, equal to the dimension of the iteration-space. We will call the variables corresponding to the nullspace the *free variables* of the system.

To find the solution, we use the *hermite normal decomposition* [10]. This procedure gives us two matrices C_1 and C_2 such that:

$$N[C_1 C_2] = [H0]$$

in which matrix H is called the hermite normal form of N. Matrix H has an inverse because the rows of N are linearly independent. Observe that matrix C_2 consists of the vectors of the n nullspace vectors of N as $NC_2 = 0$. So any linear combination of the vectors of C_2 added to a given solution s will also be a solution of the system. Because we are only interested in the values of I, we decompose matrix C_1 into C_{11}, size n by n, and C_{12} and decompose matrix C_2 into matrices C_{21} and C_{22} as follows:

$$C = \begin{bmatrix} C_1 & C_2 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{21} \\ C_{12} & C_{22} \end{bmatrix}$$

Now, the columns of matrix C_{21} are the lattice vectors. So the hermite normal form gives us directly lattice matrix L defined by the divs.

Example 6.3

Hermite normal decomposition of matrix N gives:

$$C_1 = \begin{pmatrix} 1 & 0 & -2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

and matrix C_2:

$$C_2 = \begin{pmatrix} 6 & 0 \\ 0 & 2 \\ 3 & 0 \\ 0 & 1 \\ 2 & 0 \\ 1 & 0 \end{pmatrix}$$

The iterators i and j are defined by matrix C_{21}. Let K_f be the vector of free variables. We write $I = (i, j)^t$, with offset O still to be determined, as:

$$I = \begin{pmatrix} 6 & 0 \\ 0 & 2 \end{pmatrix} K_f + O$$

□

7 LATTICE OFFSET

Next we have to find the lattice offsets. Let $B = (b_1, .., b_m)^t$ be the vector of the divisors of the integer divisions, with remainder r_k between $0 \leq r_k \leq b_k$. An offset O must first of all be an integral solution of the system:

$$\mathbf{0} \leq N \begin{pmatrix} O \\ Q \end{pmatrix} \leq B \tag{9}$$

Apart from these inequalities there may be others in the program that restrict the value of the variables standing for the integer divisions. We disregard inequalities not involving Q as they do not affect the lattice offset.

Let $< N_q, B_q >$ be the system of all inequalities involving Q. We assume that $N_q C_2 = 0$. When this assumption is satisfied, we may use the vectors of C_{21} as lattice vectors because the variables corresponding to C_{21} are free.

Let K_b be the vector of variables corresponding to matrix C_1 and let K_f be the vector of variables corresponding to matrix C_2.

We define $(O, Q)^t$ as

$$\begin{pmatrix} O \\ Q \end{pmatrix} = [C_1 C_2] \begin{pmatrix} K_b \\ K_f \end{pmatrix} \tag{10}$$

and substitute it in the polytope:

$$N_q \begin{pmatrix} O \\ Q \end{pmatrix} \quad \geq \quad B_q \tag{11}$$

after which we obtain the polytope:

$$N_q C_1 K_b \quad \geq \quad B_q \tag{12}$$

This polytope defines all offsets $O = C_{11} K_b$ of the lattice and we call it the *lattice offset domain*. The number of offsets depend on the value of the divisors b. The lattices corresponding to the polytope in q are defined by:

$$I = C_{21} K_f + O \tag{13}$$

$$O = C_{11} K_b \tag{14}$$

$$N_q C_1 K_b \geq B_q \tag{15}$$

The lattices are bounded by remaining inequalities of the nested loop program. These inequalities together with a lattice define an iteration-domain.

A special case is when the offset domain contains a single point. Then the lattice descriptions reduces to $I = C_{21}K_f + O$, and we do not have to enumerate the lattice offset domain.

Example 7.1

In example 6.1 there are three if-statements defining inequalities in q:

$$-i + 2 * q1 - 1 \geq 0 \tag{16}$$
$$-j + 2 * q2 - 1 \geq 0 \tag{17}$$
$$-i + 3 * q3 - 3 \geq 0 \tag{18}$$
$$\tag{19}$$

After the substitution $I = C_{11}K_b$ and $Q = C_{12}K_b$, we get inequalities in variables of K_b:

$$-k_1 \geq 1$$
$$-k_2 \geq 1$$
$$-k_1 - k_3 \geq 3$$

By the same substitution we get for the inequalities of the remainders:

$$-1 \leq k_1 \leq 0$$
$$-1 \leq k_2 \leq 0$$
$$-2 \leq k_3 \leq 0$$
$$-1 \leq -k_3 + 2k_4 \leq 1$$

After some computation we find that $K_b = (-1, -1, -2, -1)^t$ is the only solution. So that the offset

$$O = C_{11}K_b = (3, -1)^t$$

□

8 CONCLUSION

This paper shows the relation between several forms of describing the data dependencies of nested loop programs. In particular we have explained the relation between integer divisions inside the Single Assignment Programs generated by $HiPars$ and linearly bounded lattices in descriptions of Dependence Graphs.

We have extended the class of nested loop that $HiPars$ can take as input to programs containing non-linear integer division functions inside the expressions. This extension is based on the definition of integer division by which we can linearize the expressions in the program at the cost of additional variables.

The conversion of SAP with div functions to linearly bounded lattice descriptions is achieved by taking the hermite normal form of the matrix N defined by integer divisions. This decomposition leads to matrices C_1 and C_2, with corresponding variable vectors K_b and K_f, respectively. Matrix C_2 defines the lattice vectors with the variables of K_f as free variables. The domain of lattice offsets is formed by a polytope in variables of K_b. The polytope is characterized by matrix C_1 and inequalities

of the variables standing for the integer divisions.

As a result, we have transformed the polytopes defined by a SAP into linearly bounded lattices to be used in the description of the corresponding DG.

References

[1] U. Banerjee. *Dependence Analysis for Supercomputing*. Kluwer Academic Publishers, 1988.

[2] L. Brickman. *Mathematical Introduction to Linear Programming and Game Theory*. Springer-Verlag, 1989.

[3] Jichun Bu. *Systematic Design of Regular VLSI Processor Arrays*. PhD thesis, Delft University of Technology, Delft, The Netherlands, May 1990.

[4] Ed Deprettere, Peter Held, and Paul Wielage. Model and methods for regular array design. *Int. J. of High Speed Electronics and systems*, 4(2):Special issue on Massively Parallel Computing–Part II, 1993.

[5] P. Feautrier. Parametric integer programming. *Recherche Opérationelle; Operations Research*, 22(3):243–268, 1988.

[6] P. Feautrier. Dataflow analysis of array and scalar references. *Int. J. Parallel Programming*, 20(1):23–51, 1991.

[7] Peter Held. Hipars' reference guide. Technical report, Dept. Electrical Engineering, Delft University of Technology, 1993.

[8] Peter Held and Ed F. Deprettere. Hifi: From parallel algorithm to fixed-size vlsi processor array. In Francky Catthoor and Lars Svensson, editors, *Application-Driven Architecture Synthesis*, pages 71–92. Kluwer Academic Publishers, Dordrecht, 1993.

[9] F.Balasa F.Franssen F.Catthoor H.De Man. Transformation of nested loops with modulo indexing to affine recurrences. In C.Lengauer P.Quinton Y.Robert L.Thiele, editor, *Special issue of* Parallel Processing Letters *on* Parallelization techniques for uniform algorithms. World Scientific Pub., 1994.

[10] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1988.

[11] W. Pugh. A practical algorithm for exact array dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.

[12] L. Thiele and U. Arzt. On the synthesis of massively parallel architectures. *Int. J. of High Speed Electronics and Systems*, 4(2):99–131, 1993.

[13] Alfred van der Hoeven. *Concepts and Implementation of a Design System for Digital Signal Processing*. PhD thesis, Delft University of Technology, Delft, The Netherlands, October 1992.

[14] Kung S. Y. *VLSI Array Processors*. Prentice Hall, 1988.