Systematic generation of executing programs for processor elements in parallel ASIC or FPGA-based systems and their transformation into VHDL-descriptions of the processor element control units

O. Maslennikow

Institute of Electronics, Technical University of Koszalin
Ul. Partyzantow 17, 75-411 Koszalin, Poland
E-mail: oleg@ie.tu.koszalin.pl

Abstract. In this paper, a method for the systematic generation of the executing programs for the processor element of the parallel ASIC or FPGA-based systems liked to processor arrays is proposed. In this method, the each processor element of the array has separate control unit and is controlled in an autonomous way, based on the executing program received from the host computer before computations. This method allows: (i) to minimize executing program size stored in the processor elements; (ii) to make sizes of these programs independent from sizes of input data sets; (iii) to provide the independence of program contents from the realized applied algorithm; (iv) to derive the VHDL-description of all processor element control units in the behavioral style.

1 Introduction

Advantages in VLSI technology have stimulated research in application - specific architectures, which are tailored to real-time applications. Among these architectures, which can have a different degree of specialization, and are destined for realization as ASIC or FPGA-based device [6], architectures like to processor arrays [2,3,4] (PA) are widely used.

Architectures of such systems are usually designed [2,3,4,5] using top-down methodology, i.e. applying methods of regular algorithms mapping. Such algorithms are usually expressed by systems of recursive equations or nested loops, or by regular dependence graph (DGs). Each node of such a DG corresponds to a certain operator (or iteration) of the original algorithm, and is associated with the integer vector $K = (k_1,...,k_n)$. All its nodes are located in the vertices K of a lattice $K^n \subset Z^n$, where K^n is called the index space. If the iteration corresponding to a node K_2 depends on the iteration corresponding to another node K_1, this dependence is represented by the dependence vector $d = K_2 - K_1$. In the course of mapping, a given algorithm AL with the dependence graph G is transformed into a set of structural schemes $C = <S, T, \Phi>$ of parallel systems implementing this algorithm. Here S is a directed graph called the array structure, T is the synchronization function specifying the computation time of nodes in the DG, and Φ is the set of the algorithm operators.

Using known mapping methods, efficient parallel architectures for implementing algorithms with regular data dependencies, as well as internal structures of their

processor elements (PE) have been designed. The next stage is designing the control units (CU) for all PE's of the system. If processor elements have separate control units with the program memory, then their executing program should be designed. In opposite case, the structure of the control unit or its VHDL-description should be derived for each EP of the system.

The existing mapping methods don't allow to solve these problems. Therefore, in this paper, the method for automatic generation of the executing programs for the arbitrary processor element of the parallel system implementing applied regular algorithm given by nested loops is proposed. Moreover, the transformation of the PE executing program into VHDL-description of its CU is shown. The main idea of the method consists of loading, before computation, into each PE control unit a control information from the host. This information represents all types of algorithm operators (or graph nodes) which should be executed in the PE, as well as ranges of changing coordinates k_i of nodes, for each type of operators which this PE should be performed. Moreover, every PE is provided with an expression for computing the coordinates of the next executed node of DG, in accordance with the timing schedule mapping. These data are parameters of the executing program, which should be carried out by PE. The template of this program is written in the program memory of PE or realized by hardware way in the PE control unit. During computation, in each time step, each PE determines the coordinates of the next executed node of DG and compares them with the given ranges. In the case of the positive answer, the control unit of this PE determines the type of operator (COP) which should be executed. Then COP is transformed into a real instruction forall PE blocks. In the case of the negative answer, the "empty" operator (NOP) is performed by PE. Details of the proposed method are illustrated on the example of the Gauss elimination algorithm.

2 Deriving executing programs for the processor elements of PAs

We assume that an elementary loop nest (ELP) consists of a multi-level construction of *n* nested DO-statements including one another, with no exit from the loop body [1,5]. Each elementary nest is characterized by its dimension *n* (which is equal to the number of DO-statements) and defines the corresponding iteration space. Each of its nodes represents a single execution of the loop body, and is defined by an iteration vector $K = (k_1, k_2, \dots , k_n)$. If between two consequent DO-statements there exist a loop body, then such a loop construction will be called the composite loop nest. As a result, algorithms under consideration can be written in the following form:

```
do k1 =· a1 to b1 step  c1
    [ { statements of the loop body 1} ]
[ enddo]
    do k2 =· a2 to  b2 step  c2
          [ { statements of the loop body 2} ]
    [ enddo]
      . . . . . .
          do kn =· an to bn step  cn
                {statements of the loop body n}                    (1)
       enddo
```

```
      [ { statements of the loop body n-1} ]
   [ enddo]
   . . . . . .
  [ { statements of the loop body 1} ]
[ enddo]
```

Here a_j, b_j are expressions denoting the lower and upper limits of the loop at nesting level j, while c_j stands for the incrementing of variables k_j. Square brackets are used to denote that the corresponding statements may be absent, and q is the number of different loop bodies (or different types of operators), $q \leq n$.

Let us assume, that using one of known methods [1,5], the DG of an original algorithm was derived, and its description is given in a form of Table 1. Here $x_{i,j}$, $y_{i,j}$ and c_j denote the low limit, upper limit and increment values for the coordinate k_j respectively, where $j = 1, \ldots, n$, and $i = 1, \ldots, q$.

Table 1. Description of an algorithm DG

Operator type	Coordinate k_1			Coordinate k_2			. . .	Coordinate k_n		
	from	to	step	from	to	step	...	from	to	step
1	$x_{1,1}$	$y_{1,1}$	c_1	$x_{1,2}$	$y_{1,2}$	c_2	...	$x_{1,n}$	$y_{1,n}$	c_n
2	$x_{2,1}$	$y_{2,1}$	c_1	$x_{2,2}$	$y_{2,2}$	c_2	...	$x_{2,n}$	$y_{2,n}$	c_n
...	...			...			...	...		
q	$x_{q,1}$	$y_{q,1}$	c_1	$x_{q,2}$	$y_{q,2}$	c_2	...	$x_{q,n}$	$y_{q,n}$	c_n

Let us also assume, that the structure graph S of the target parallel system with dimension m, as well as the synchronization function T, have been already derived. This means that space F_S and time F_T components of the mapping function F are known. In other words, the integer $(m \times n)$ matrix F_S determines the m-dimensional hyperplane, such that the projection of the DG onto this hyperplane gives the structure S. Besides, a node of the DG with coordinates $K = (k_1, k_2, \ldots, k_n)$ will be executed in the PE with the coordinates $F_S \cdot K$ at the time step t given by the expression

$$t = F_T \cdot K + const_1 . \qquad (2)$$

Remark. In this paper, we will assume that all the m column-vectors of the function F_S are equal to the first m coordinate vectors $k_1, k_2, \ldots k_m$ of the space W^n.

In this case, each PE of structure S will execute only those operators of the algorithm (or nodes $K = (k_1, k_2, \ldots, k_n)$) for which values of $(k_1, k_2, \ldots, k_m)$ are equal respectively to values of the first m coordinates of the PE. In other words, each PE will execute a set of nodes of the DG which belong to the hyperplane given by the last $(n-m)$ coordinates $(k_{m+1}, k_{m+2}, \ldots, k_n)$ of the space W^n . The time component F_T of the mapping F gives the sequential order of executing the nodes belonged to this hyperplane, in a PE. Thus, for the implementation of the autonomous PE control in processor arrays, the following steps should be performed.

1. Based on the description of the algorithm DG represented by the table 1 and known coordinates of PEs, the host generates a reduced tables of the DG description which is different for different processor elements. This table includes ranges of the last *(n-m)* coordinates $(k_{m+1}, k_{m+2}, \ldots, k_n)$ of DG. These ranges describe the set of those nodes of the DG which are mapped in the given PE. The number g of rows in the reduced table is equal to the number of different types of operators of the algorithm which are mapped into a given PE, so we have $1 \le g \le q$.

2. For the each PE, the host generates a partially computed expression (3) of the following form:

$$T = F_T \cdot K + const_1 = f_1 \cdot k_1 + f_2 \cdot k_2 + \ldots + f_n \cdot k_n + const_1 . \quad (3)$$

For each node $K = (k_1, k_2, \ldots, k_n)$ of the DG, this equation determines the time step in which this node will be executed. Note, that in the expression (3), the values of the first m coordinates of nodes are equal to the values of coordinates $(k_1, k_2, \ldots, k_m)$ of given PE, while $(f_1, f_2, \ldots, f_n)$ are coefficients of the time component F_T . Therefore, each the expression (3) is reduced to the following expression:

$$T = f_{m+1} \cdot k_{m+1} + f_{m+2} \cdot k_{m+2} + \ldots + f_n \cdot k_n + const_2 , \quad (4)$$

where $const_2 = f_1 \cdot k_1 + f_2 \cdot k_2 + \ldots + f_m \cdot k_m + const_1$. (5)

3. For each PE, the host forms the template of its execution program, which can be represented by the following form:

```
t=1
do Jn =· an  to  bn step  sn
   do Jn-1 = an-1  to  bn-1 step  sn-1
     . . . . . .
       do Jn-m+1 =· an-m+1  to  bn-m+1 step  sn-m+1
         { calculation of expression (5) to determine T } ;
         { finding of the operator type based on the current values of Ji};
         while t ≠T  do t = t+1 {no useful operation};
         { execution of the operator };                                  (6)
       enddo
     . . . . . .
   enddo
enddo,
```

where J_i , a_i , b_i and s_i are values of k_i , $min\{x_{r,i}, r = 1, \ldots, g\}$, $max\{y_{r,l}, r = 1, \ldots, g\}$ and c_i respectively, placed in the order of increasing of F_T coefficients f_i, $i=m+1,\ldots,n$.

Analysis of the construction (6) shows that if either sizes $x_{r,l}$ or $y_{r,l}$ of input data are changed, or other mappings F_S , F_T are used, or even another regular applied algorithm is implemented by the system, then only the number of DO-statements and their parameters a_i, b_i, c_i , as well as the form of the expression (4), should be changed in the executing program (6). Consequently, the template (6) may be stored in the program memory of PE control unit or may be realized in hardware way (if there is no program memory in the PE control unit).

3. An example. Design of the executing programs for the Gauss elimination algorithm

The algorithm corresponding to the Gauss elimination without pivoting is presented by the construction (7).

```
do k1 =· 1 to N-1 step 1
  do k2 =· k1+ 1 to N step 1
    m (k1, k2) = a(k2, k1) / a(k1, k1); /*Operator type 1*/
  enddo
  do k2 = k1 to  N step 1
    do k3 = k1+1 to N step 1
      a (k2, k3) = a(k2, k3) - m(k2, k1) * a(k1, k3);   /*Operator type 2*/        (7)
    enddo
  enddo
enddo
```

The table of this algorithm DG description (table 1) transforms to the table 2.

Table 2. Description of the Gauss elimination algorithm DG

Operator Type	Coordinate K_1			Coordinate k_2			Coordinate k_3		
	from	to	step	from	to	step	from	to	step
1	1	N-1	1	k_1+1	N	1	k_1	k_1	1
2	1	N-1	1	k_1+1	N	1	k_1+1	N	1

Let us assume, that the following mapping operator F has been obtained as the result of using mapping method [4]:

$$F(m+1,n)=\begin{bmatrix} F_S(m,n) \\ F_T(1,n) \end{bmatrix}=\begin{matrix} k_1 & k_2 & k_3 \\ \end{matrix}\begin{bmatrix} 1 & 0 & 0 \\ 1 & N & 1 \end{bmatrix} \quad (m=1)$$

Note that the derived F_S value corresponds to the projection of the graph G onto the axis k_1. As a result, the p-th PE of this processor array, where $p = 1, ..., N-1$, will execute the nodes K of the graph with the coordinates $K = (p, k_2, k_3)$. The obtained F_T value determines the following value of the constant $const_1$ in the expression (3):

$$const_1 = 1 - F_T \cdot K^* = -(N + 1),$$

where $K^* = (1, 2, 1)$ are coordinates of the first executed node of the graph G.

Based on these data, the reduced form of the DG description table is formed, which for the p-th PE of the array, is represented by the table 3, where $J_2 = k_2$ and $J_3 = k_3$ because $|f_2| < |f_3|$, while $k_1 = p$. The data from this table should be passed to the p-th PE of the array ($p = 1, ..., N-1$). Based on the component $F_T = (f_1, f_2, f_3) = (1, 1, N-1)$, the expression (4) is represented in the following form (for the p-th PE):

$$T = f_2 \cdot k_2 + f_3 \cdot k_3 + const_1 = k_2 + (N-1) \cdot k_3 + const_2, \qquad (8)$$

where $const_2 = 1 \cdot p + const_1 = k_1 - N - 1$.

The executing program for the p-th PE of the array is formed in the following form:

Table 3. Table of the p-th PE operations

Operator Type	Coordinate J_2			Coordinate J_3		
	from	to	step	from	to	step
1	p+1	N	1	p	p	1
2	p+1	N	1	p+1	N	1

```
t=1
do J3 =· x1,3 to  N step  1
    do J2 =· x1,2 to N step  1
        T  = J2 + (N-1)·J3 + p - N – 1;                                    (9)
        while t ≠T  do  t = t+1 {no useful operation};
        { determination of the operator code based on current values of Ji};
        { execution of the operator of the algorithm };
    enddo
enddo
```

where $a_3 = min\{ x_{1,3}, x_{2,3} \} = p$; $b_3 = max\{ y_{1,3}, y_{2,3} \} = N$; $c_3 = 1$;
$a_2 = min\{ x_{1,2}, x_{2,2} \} = p + 1$; $b_2 = max\{ y_{1,2}, y_{2,2} \} = N$; $c_3 = 1$.

4 Transformation of the executing program to VHDL-description of the PE control unit

When the target parallel system is realized as the ASIC or FPGA circuit, its structure and the internal structures of all PEs should be described in the HDL language [6]. In order to this, transformation of the PE executing program into corresponding VHDL-description of the PE control unit should be carried out. Note, that in this case, the PE control unit represents the „black box" with the RESET and CLOCK inputs and COP (code of operation) outputs.

The template of VHDL description of the control unit "architecture" which has been obtained from the corresponding program template (6) and should be generated by corresponding CAD environment is following (without a declarative part):

```
entity control_unit is
        generic (log2g  : integer:=2);
                port (CLK : in std_logic;
                Reset     : in std_logic;
                COP       : out std_logic_vector (1 to log2g));
        end entity control_unit;
architecture control_unit_a of control_unit is
-- declaration of constants and parameters isn't show in this program
```

```
begin
  process
    variable T,i,q,s,time_step,kop:integer;
    variable J:vector;
      begin  if Reset='1' then time_step:=0; J(n):=a(n); COP<=(others=>'0');
                else
                 Label_n: while J(n) <= b(n) loop
                     J(n-1):=a(n-1);
                     Label_n-1: while J(n-1) <= b(n-1) loop
                     . . . . . .
                       J(m+1):=a(m+1);
                       Label_m+1: while J(m+1) <= b(m+1) loop
                         T:= J(n)*f(n) + J(n-1)*f(n-1) + ...+ J(1)*f(1) + const2;
                          Label0: for i in 1 to g loop
                                  s:=0;
                                  for q in m+1 to n loop
                                    if J(q)>=X(i,q) and J(q)<=Y(i,q)then s:=s+1;end if;
                                  end loop;
                                  if s=n-m then kop:=i; exit Label0; end if;
                           end loop Label0;
                       label1: while T/=time_step loop time_step:=time_step+1;
                               COP<=(others=>'0'); wait on clk; wait on clk;
                       end loop label1;
                       if s=n-m then
                                 COP<=CONV_STD_LOGIC_VECTOR(kop,log2g);
                          else COP<=(others=>'0');    end if;
                       time_step:=time_step+1; wait on clk; wait on clk;
                         J(m+1):= J(m+1)+c(m+1);
                       end loop Label_m+1;
                     J(m+2):= J(m+2)+c(m+2);
          . . . . . .
                  end loop Label_n-1;
                J(n):= J(n)+c(n);
             end loop Label3;
           end if;
           wait on reset;
        end process;
end architecture control_unit_a;
```

Here constants *n* and *m* represent the dimensions of the DG and the structure of the processor array respectively. The constants *g* and *log2g* are the number of the operator types in DG and the number of the lines in the output COP respectively.

The results of simulation of the VHDL-model of the second PE (*p=2*) control unit in ActiveVHDL v.4.2 environment are represented in the fig. 1. These results prove the correctness both the obtained executing program (9) and the proposed method.

5 Conclusions

In this paper, the method for the systematic generation of the executing programs for the arbitrary processor element of the parallel ASIC or FPGA-based device implementing applied regular algorithm given by nested loops has been proposed. This method allows: (i) to minimize executing program size stored in the processor elements; (ii) to make sizes of these programs independent from sizes of input data sets; (iii) to provide, to a maximum possible extent, the independence of program contents from the realized applied algorithm; (iv) to provide the possibility of generation of these programs in the fully automatic way, based on the description of the algorithm dependence graph and device structure; (v) to derive the VHDL-description of all processor element control units in the behavioral style. Correctness of the proposed method has been illustrated on the example of the Gauss elimination algorithm.

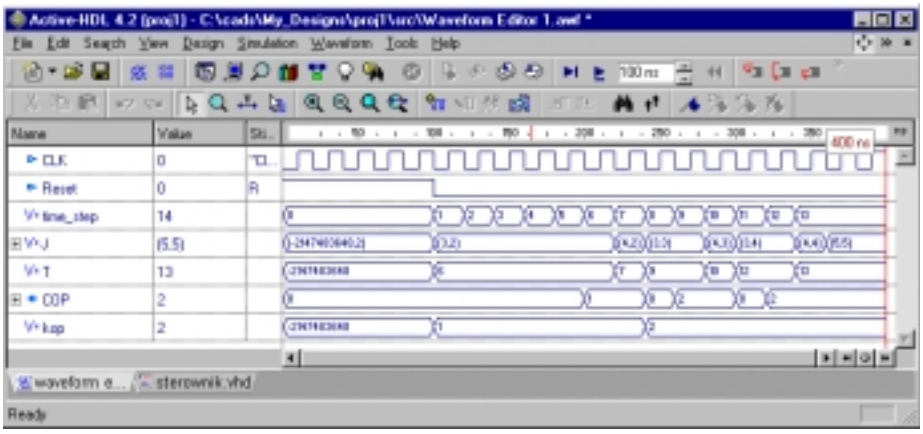

Fig. 1. Simulation results of VHDL model of the second PE control unit ($p=2$)

References

1. U. Banerjee, An introduction to a formal theory of dependence analysis. *J. Supercomput.*, 1988 (2) 133-149.
2. A. Darte, Y. Robert, Mapping uniform loop nests onto distributed memory architectures. *Parallel Computing*, 1994 (20) 679-710.
3. S.Y. Kung, VLSI Array Processors, *Prentice Hall, Englewood Cliffs*, 1988.
4. R. Wyrzykowski, J. Kanevski, O. Maslennikov, Mapping recursive algorithms into processor arrays. *Int. Workshop Parallel Numerics'94, Smolenice (Slovakia)*, 1994, pp.169-191.
5. R.Wyrzykowski, J. Kanevski, O. Maslennikov, A method for deriving dependence graphs of recursive algorithms for processor array designs. *Proc. Int. Workshop Parallel Numerics'95, Sorrento (Italy)*, 1995, pp.263-280.
6. G.R. Goslin. A Guide to Using Field Programmable Gate Arrays (FPGAs) for Application-Specific Digital Signal Processing Performance. *Xilinx, Inc.*, 1995.