Buletinul Stiintific al Universitatii Baia Mare
Seria B, Matematică-Informatică, vol. VIII(1992), 77-87

ON PARALLEL EXECUTION IN LOOP EXIT SCHEMES

Iuliu Sorin POP

Abstract. This paper presents a mechanism which provides a possibility to control the parallel execution in **Loop Exit Schemes (LES)**.

0. Introduction

Beside detecting the statements which can be executed in parallel, a natural demand of parallel execution is represented by the automatic specifying of these statements during the execution. Referring to the facilities offered by the automatic processing, LES were choosen as a model of program schemes.

1. Definitions

Let M = A ∪ T be a terminal alphabet, where A is the set of assignment marks and T is the set of test marks. Let:

{ +, -, NULL, IF, THEN, ELSE, ENDIF, LOOP, ENDLOOP, EXIT }

be a set of reserved words and let LM = {1,2,...} be a set of loop-mark symbols.

Definition 1. A Loop Exit Free Scheme (LEFS, see [1]) over M is recursively defined as follows:

a) "NULL" is a LEFS; for each $a \in A$, "a" is a LEFS;

b) if $t \in T$, α and β are LEFS and i, j, $k \in LM$, then:

b1) "$\alpha\beta$"

b2) "IF t THEN α[EXIT$_i$][ELSE β[EXIT$_j$]]ENDIF",

where [w] means that w is optional

b3) "LOOP$_k$ αENDLOOP$_k$"

are LEFS;

c) Each LEFS is obtained from the rules a) and b) and satisfies:

c1) each two LOOPs must have two distinct loop-mark simbols;

c2) for each "$LOOP_k$ αENDLOOP$_k$" there is at least an $EXIT_k$ in α;

c3) if "$LOOP_k$ αENDLOOP$_k$" is a LEFS then $\alpha = \alpha' EXIT_k \alpha''$ and $EXIT_k$ appears only in α.

Definition 2: A Loop Exit Schemes (LES, see [1]) is a LEFS which satisfies:

c3') for each $EXIT_k$ there is "$LOOP_k \alpha ENDLOOP_k$" in the LEFS such that $\alpha = \alpha' EXIT_k \alpha''$

Definition 3: Let α be a LEFS. The skeleton word $S(\alpha)$ associated to α is defined by:

a) if $\alpha = \varepsilon$ then $S(\alpha) = \varepsilon$;

b) if α_1 and α_2 are two LEFSs then

b1) if $\alpha = \alpha_1 a \alpha_2$ and $a \in A$, then $S(\alpha) = S(\alpha_1)aS(\alpha_2)$;

b2) if $\alpha = \alpha_1 NULL \alpha_2$ then $S(\alpha) = S(\alpha_1)S(\alpha_2)$;

b3) if $\alpha = \alpha 1$ IF βENDIF α_2, then $S(\alpha)=S(\alpha_1)IS(\alpha_2)$;

b4) if $\alpha = \alpha_1$ $LOOP_k$ $\beta ENDLOOP_k \alpha_2$ then $S(\alpha) = S(\alpha_1)L_k S(\alpha_2)$.

Definition 4: Let $x_1 \alpha x_2 \beta y_2 \delta y_1$ be a LEFS, where $(x_i,\ y_i)$ can be:

(IF t THEN, ENDIF), or

(IF t THEN γ ELSE, ENDIF), or

($LOOP_k$, $ENDLOOP_k$).

The direct word from x_1 to x_2, denoted by $D(x_1 \alpha x_2)$, is recursively defined as follows:

a) if α is a LEFS then:

a1) if x_1 = IF t THEN then $D(x_1 \alpha x_2) = a + S(\alpha)$;

a2) if x_1 = IF t THEN γ ELSE then $D(x_1 \alpha x_2) = a - S(\alpha)$;

a3) if $x_1 = LOOP_k$ then $D(x_1 \alpha x_2) = S(\alpha)$

b) otherwise:

b1) if $\alpha = \alpha_1$ IF_i u $THEN_i \alpha_2$ and $\delta = \delta_2 ENDIF_i \delta_1$ then $D(x_1 \alpha x_2) = D(x_1 \alpha_1 IF_i)\ D(IF_i \alpha_2 x_2)$, where the symbol i was used to distinguish different "IF - ENDIF" structures;

b2) if $\alpha = \alpha_1 IF_i$ u $THEN_i \gamma ELSE_i \alpha_2$ and $\delta = \delta_2 ENDIF_i \delta_1$ then $D(x_1 \alpha x_2) = D(x_1 \alpha_1 IF_i) D(IF_i \alpha_2 x_2)$;

b3) if $\alpha = \alpha_1 LOOP_k \alpha_2 ENDLOOP_k \delta_1$ then $D(x_1 \alpha x_2) = D(x_1 \alpha_1 LOOP_k) B_k D(LOOP_k \alpha_2 x_2)$.

Definition 5: Let S be a LES. The language L(S) associated to S is generated by the following context free grammar:

$G(S) = (\{\nabla\} \cup \{I_j, j\geq 0\} \cup \{L_k, B_k, k\geq 0\}, M \cup \{+, -\}, P, \nabla)$,

where "∇" is a new symbol, I_j is a nonterminal for "IF_j $-ENDIF_j$", if this structure exists, L_k and B_k are two nonterminals for "$LOOP_k$ - $ENDLOOP_k$" if this structure exists, and the set P of the productions is constructed by the following rules:

a) $\nabla \rightarrow S(\alpha)$

b) for each "IF_j t $THEN_j$ $\alpha ENDIF_j$" we consider the productions:

b1) $I_j \rightarrow b -$

b2) $I_j \rightarrow b + S(\alpha)$, if no $EXIT_k$ exists such that $\alpha = \alpha' EXIT_k$;

c) for each IF_j t $THEN_j$ $\alpha ELSE_j$ β $ENDIF_j$ we consider the productions:

c1) $I_j \rightarrow b + S(\alpha)$, if $\alpha \neq \alpha' EXIT_k$

c2) $I_j \rightarrow b - S(\beta)$, if $\beta \neq \beta EXIT_k$

d) for each "$LOOP_k$ $\alpha_1\alpha_2\delta$ $ENDLOOP_k$" we consider the productions:

d1) $L_k \rightarrow S(\alpha_1\alpha_2\delta)\ L_k$ and

$B_k \rightarrow S(\alpha_1\alpha_2\delta)\ B_k$

d2) $L_k \rightarrow D(LOOP_k\alpha_1 IF_j)t + S(\beta)$, if

$\alpha_2 = IF_j t\ THEN_j\ \beta EXIT_k ENDIF_j$, or

$\alpha_2 = IF_j t\ THEN_j\ \beta EXIT_k\ ELSE_j\ \gamma ENDIF_j$

d3) $L_k \rightarrow D(LOOP_k\alpha_1 IF_j)t - S(\beta)$, if

$\alpha_2 = IF_j t\ THEN_j\ \gamma\ ELSE_j\ \beta\ EXIT_k\ ENDIF_j$.

Definition 6: Let S be a LEF. The static word associated to S is obtained from S by erasing all reserved symbols.

Definition 7: A word $z = a_{i1}x_{i1}a_{i2}x_{i2}...a_{is}x_{is}$, where $x_{ij} \in \{+, -, \varepsilon\}$, is a section for S if there is $w \in L(S)$ such that

a) $w = xyz$;

b) $i_j < i_{j+1}$ for $j = 1,s-1$;

c) if $x \neq \varepsilon$ then $x' a_{io}x_{io}$, with $i_o > i_1$

d) if $y \neq \varepsilon$ then $y = a_{is+1}y'$, with $i_s > i_{s+1}$

The set of all sections is denoted by SEC(S). Intuitively, a section is a maximal sequence of statements of S such that their order of execution is the same with their order in the text of the program (static word).

Definition 8: A word $z = a_{i1} \ldots a_{im} \in M^+$ is a sequence of assignments with final test (AFTS) if there is $w \in L(S)$ such that:

$\nabla \Rightarrow^* \alpha a_{i1} a_{i2} \ldots a_{im} x_{im} \beta \Rightarrow^* w$,

and one of the following conditions is verified:

a) $a_{i1}, \ldots, a_{im-1} \in A$, $a_{im} \in T$ (and certainly $x_{im} \in \{+, -\}$),

$\alpha = \varepsilon$ or $\alpha = \alpha' a_j x_j$ where $a_j \in T$, $x_j \in \{+, -\}$ and

$\beta \in (M \cup \{I, L, B, +, -\})^*$.

(this is the case when the sequence effectively ends with a test);

b) $a_{i1}, \ldots, a_{im} \in A$, $x_{im} = \varepsilon$, $\beta = \varepsilon$, $\alpha = \varepsilon$ or $\alpha = \alpha' a_j x_j$, where $a_j \in T$ and $x_j \in \{+, -\}$ (this is the case when the sequence represents the last part of the program, the assignments preceeding the STOP statement).

c) $a_{i1} a_{i2} \ldots a_{im} = a_1 \ldots a_n = w$ (this is the trivial case when there is no test in the program, which is a sequence of assignments)

By AFT(S) we denote the set of sequences of assignements with final test associated to the scheme S.

In [1],[2],[3],[4] are shown some properties of S schemes and of the associated grammars. Algorithmic possibilities for determining the sets AFT(S) and SEC(S) are also presented there. In [4] are indicated some possibilities for automatic detection of the statements which can be executed in parallel. They are based on the AFT sequences and on the program scheme concept agreeing with S.Greibach [5].

Let us consider the following example of LES scheme:

```
a1
a2
LOOP1
      IF a3 THEN EXIT1 ENDIF
      a4
      a5
ENDLOOP1
a6
```

We have: $L(S) = a_1 a_2 (a_3 - a_4 a_5)^* a_3 + a_6$,

$AFT(S) = \{a_1 a_2 a_3, a_4 a_5 a_3, a_6\}$.

An example of an adequate program scheme is:

```
i:= 1;
k:= 1;
```

```
LOOP₁
   IF i ≥ 3 THEN EXIT₁ ENDIF
   k:= i * k
   i:= i + 1;
ENDLOOP₁
i:= k;
```

2. The Mechanisms Ideea

For automatic specifying of the statements which can be executed in parallel in a LES scheme we shall consider a couple (M,S) (see [9]) where M is a finite automaton and S is a set of queues. Each queue contains a word formed over

$\Sigma = \Sigma_i \cup \Sigma_e$, where:

$\Sigma_i = \{a^i \,/\, a \in M\}$ (the beginning symbols associated to the statements of the LES).

$\Sigma_e = \{a^e \,/\, a \in M\} \cup \{\alpha^e\}$ (the set of ending symbols associated to the statements of the LES, α^e being an extra symbol used to initiate the mechanism).

The automaton's alphabet is {+, -}.

The behavior of this mechanism follows the following rules.

(1). A control state q is composed from a state of the automaton M and the content of the queues;

(2). A statement a is said to be "enabled" in a state q iff there is any queue containing the symbol a^i and for each queue F which contains the symbol a^i there is a word $x \in (\Sigma\setminus\{a^i\})^*$ and a symbol $b^e \in \Sigma_e$ such that xb^ea^i is a prefix of the content of F;

(3). The execution of the statement a has as effect the replacement of the first occurrence of a^i (if it exists) by a^e in every queue F;

(4). Any transition of the automaton produces the appending of some words to the end of the queues.

The queues specify the relationships existing between the statements of the LES.

3. The Value Transmission Based Parallelism

This parallelism is based on the value transmission between statements, the relation "Seg" being generated.

Definition 9. If $x \in L(S)$ is a computation of a LES and $a,b \in M$ two statements, we say there is a value transmission from the n^{th} occurrence of the symbol a to the p^{th} occurence of the symbol b in x iff:

- the n^{th} occurrence of a precedes the p^{th} occurrence of b;
- a variabile m is an output of the statement a and an input of the statement b;
- there is no other assignment of m between these two occurrences of a and b.

We denote this situation with (a,n,b,p) ∈ Seg(x,m).

Similary the "segment of a variabile" concept is defined in [7] and [8].

Definition 10. Two sequences x and y are saied to be (Seg) equivalent iff:

- for every statement a, $E(\Sigma(a),x) = E(\Sigma(a),y)$, where $\Sigma(a)$ means {a} if a ∈ A, (a+,a−) if a ∈ T and E(A,z) with $A \subseteq \Sigma$ denotes the word obtained from z by erasing the symbols which do not belong to A.
- for every variable m, Seg(x,m) = Seg(y,m)

Notice that Logrippo ([7],[8]) has demonstrated that a maximal parallelism may be reached by retaining from the sequential program, sequentiality only between occurrences of statements linked by relation Seg(x) = ∪{Seg(x,m)/m - variable} (therefore the statements between which exists a value transmission).

For the program scheme presented above we have:

statement number	1	2	3	4	5	6
input variables	∅	∅	i	i,k	i	k
output variables	i	k	∅	k	i	i

4. The Mechanism Construction

(S) The queue set consists of the union of the sets I(a), where a ∈ M, I(a) is the set of the queues associated to the statement a and is defined by:

$$I(a) = \begin{cases} \{F_{a,m} / m \in D(a)\} & \text{if } D(a) \neq \emptyset \\ \{F_{a,0}\}, & \text{if } D(a) = \emptyset \end{cases}$$

(D(a) denotes the set of input variables of the statement a).

(M) Corresponding to each test statement of the LES we consider a state of the automaton and to this set of states we add a final state.

Between two states q_1 and q_2 which are not final we consider the existence of a transition for every appearance in distinct circumstances (see lowerdown) the following situation:

- Let a_1 and a_2 be the test statements corresponding to q_1 and q_2. There is a word w in the language associated to the LES such that this one contains a subsequence y bounded at left by a_1 and at right by a_2, and neither a_1 nor a_2 do appear in y.

Between an unfinal state q_1 and the final one we consider the existence of a transition for every appearance in distinct circumstances (see lowerdown) of the following situation.

- Let a be the test statement which generated the state q_1. There is a word w in the language associated to the LES such that its suffix is made of a sequence ay, where a does not appear in y.

Definition 11: The sequence y is called the link sequence associated to the transition while ya_2 is named the enriched link sequence associated to the transition.

We notice that in case of a transition to the final state , the statement a_2 from the definition above is a void statement, so the link sequence associated to the transition is identical with the enriched one.

Refering to the definition 8 the following property is clear:

Property 1: Every enriched link sequence is an element of the AFT set.

Through the appearance in distinct circumstances of the previous situation we may understand the uniqueness of the 3 - tuple (a_1, a_2, y).

- every transition will be marked with " + " or " - " according to the ending mode of the test statement a_1 which leads us to the statement a_2 through the link sequence y.

Remark. We notice that transitions are made only from states corresponding to test statements. Also, agreeing with the particular structure of the language associated to a LES, if a LES contains test statements then there always exists an unique symbol $a_k \in T$ such that it appears in every word of the language, and the first appearance of this symbol preceding all the other possible appearences of an test simbol (the prefix bounded by a_k in every word of the language being also a constant AFT sequence). This symbol will generate the initial state of the automaton. If the LES doesn't contain any test simbol, the associated language contains only a single word w. In this case the automaton contains only a single state, which is final and also initial.

Definition 12. If $F_{a,m} \in I(a)$ we denote by $O_p(Fa,m)$ the following set:

$$O_p(F_{a,m}) = \begin{cases} \{a\}, & \text{if } D(a) = \emptyset \text{(and } m=0) \\ \{b \in M/m \in R(b)\}, & \text{if } a \in A \text{ and } D(a) \neq \emptyset \\ \{a\} \cup \{b \in M/m \in R(b)\}, & \text{if } a \in T \text{ and } D(a) \neq \emptyset \end{cases}$$

(R (a) denotes the set of output variables of the statement a).

We notice that this set holds the sets of statements which have an output variable which is input variable of a, with the following remarks:

- The first situation appears if one statement does not use any value produced by other statements. In this case, for its reexecution, this statement must only wait for the ending of the process of value storing in the variables affected by itself.

- The last situation corresponding to a test statement, case in which that test must not be preceeded by the statements from the block controlled by itself, including its reexecution.

Now we shall define the appending operations:

- A transition between a state q_1 and q_2 appends to any quene $F_{a,m}$ a word $x \in (\{b^t/b \in O_p(fa,m)\})^*$ which corresponds exactly to the sequence of symbols belonging to $Op(F_{a,m})$ appearing in the enriched link sequence associated to this transition.

- The queues are initialized with the simbol α^*, followed by the result of an appending operation defined analogous through considerring of a transition from a virtual state, corresponding to the moment which precedes the execution, to the initial state. Agreeing with the remark above, the AFT sequence, which may be associated as an enriched link sequence to this transition, is always the same.

- Any transition (therfore any appending operation) is performed only after the execution of the test statement which generated the state from which the transition is made.

Remark. In order to simplify, we consider in this model that each operator is instantaneously executed. A solution for avoiding this restriction is given in [9] and consists in decomposing each operator into a beginning operator and an ending one having in common a variable which is an output of the former and an input of the latter.

The value transmision (and also maintaining the Seg-relations in the LES) can be done through allocating of an additional memory zone for each variable, these zones being able to keep the value of the corresponding variable. For every variable, the initial memory zone is used only for reading and the second additional zone is used for writing. The parallelism model accepted here allows simultaneous readings from a memory zone and a single writing. The value stored in the additional zone is copied into the initial zone only after ending of all statements which were simultaneously launced in execution.

For the presented example, keeping the notations we get:

$I(a_1) = \{Fa_1, 0\}$

$I(a_2) = \{Fa_2, 0\}$

$I(a_3) = \{Fa_3, i\}$

$I(a_4) = \{Fa_4, i;\ Fa_4, k\}$

$I(a_5) = \{Fa_5, i\}$

$I(a_6) = \{Fa_6, k\}$

and correspondingly the followings:

$Op(Fa_1, 0) = \{a_1\}$

$Op(Fa_2, 0) = \{a_2\}$

$Op(Fa_3, i) = \{a_1,\ a_3,\ a_5,\ a_6\}$

$Op(Fa_4, i) = \{a_1,\ a_5,\ a_6\}$

$Op(Fa_4, k) = \{a_2,\ a_4\}$

$Op(Fa_5, i) = \{a_1,\ a_5,\ a_6\}$

$Op(Fa_6, k) = \{a_2,\ a_4\}$

Therefore the finite automaton obtained is:

Where: (a) denotes the statement which generates the state from which a transition is made.

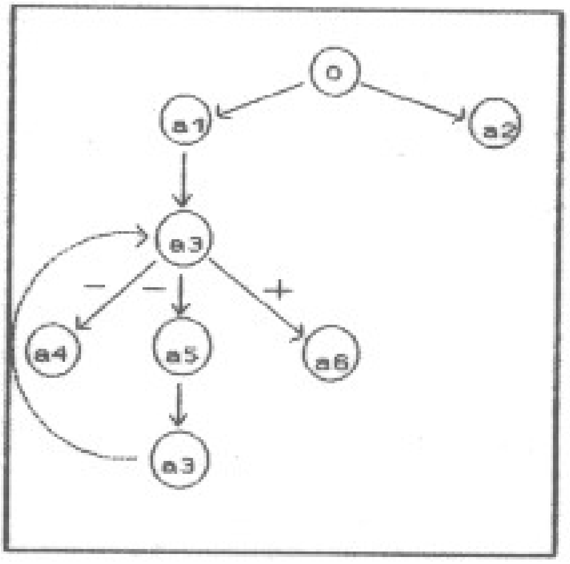
[a] denotes the statement which generates the destination state of the transition.

"x" denotes the link sequence associated to the transition.

The appending operations are:

quene	$Fa_1,0$	$Fa_2,0$	Fa_3,i	Fa_4,i	Fa_4,k	Fa_5,i	Fa_6,k
initial	$\alpha=a_1^1$	$\alpha=a_2^1$	$\alpha=a_1^1a_3^1$	$\alpha=a_1^1$	$\alpha=a_2^1$	$\alpha=a_1^1$	$\alpha=a_2^1$
$q_1\text{->}q_1$			$a_5^1a_3^1$	a_5^1	a_4^1	a_5^1	a_4^1
$q_1\text{->}q_2$			a_6^1	a_6^1		a_6^1	

In this case, the precedence graph of the parallel executable statements is:



R E F E R E N C E S

1. BOIAN, F.M.: "Loop - Exit Schemes and Grammars: Properties Flowchartablies", Studia Universitatis Babeş-Bolyai, Mathematica 31(1986), no 3.

2. BOIAN, F.M.: "Reversible execution with Loop Exit schemes", Studia Universitatis, Babeş-Bolyai, Mathematica 32(1987), no 3.

3. BOIAN, F.M.: "Reducing the Loop Exit Schemes", Mathematica, Cluj-Napoca, 28(1986), no 1.

4. BOIAN, F.M., FRENTIU, M., KASA, Z.: "Parallel execution in Loop Exit schemes", Babeş-Bolyai University, Seminar on Computer Science, Preprint no 9, 1988.

5. GREIBACH, S.: "Theory of program structures: schemes, semantics, verifications", (Lecture notes in computer science), Springer Verlag, 1975.

6. KELLER, R.M.:"Parallel program schemata and maximal parallelism", Journal ACM, 20(1973), no 3-4.

7. LOGRIPPO, L.: "Renamings and Economy of Memory in Program Schemata", Journal ACM, 25(1978), no 1.

8. LOGRIPPO, L.: "Renamings, Maximal Parallelism and Space - Time Tradeoff in Program Schemata", Journal ACM, 26(1979), no 4.

9. ROUCARIOL, G.: "Transformation of sequential programs into parallel programs", in Parallel Processing Systems-Advanced Course", D.J. Evans (ed), Cambridge Univ.Press, 1982.

University "Babeş Bolyai" Cluj-Napoca
Faculty of Mathematics
str. M. Kogălniceanu nr.1
3400 CLUJ-NAPOCA
ROMÂNIA