TÜBİTAK

Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 2135 – 2149
© TÜBİTAK
doi:10.3906/elk-1402-250

PLEA: Parametric loop bound estimation in WCET analysis

Saeed PARSA, Mehdi SAKHAEI-NIA*
Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

Received: 22.02.2014 • **Accepted/Published Online:** 05.07.2014 • **Final Version:** 15.04.2016

Abstract: Worst-case execution time (WCET) analysis of a program is important to verify the temporal correctness of real-time systems. Parametric WCET analysis represents the WCET of the program as a formula, where the unknown values affecting the WCET are parameterized. Many issues usually affect the WCET of a program, including the loop bound. In parametric timing analysis, instead of determining a constant upper bound for a loop, a symbolic formula represents the loop bound. In this paper, a new method is presented for the parametric loop bound analysis based on path analysis. Instead of considering the basic bocks on their own and independent of the rest, the execution paths within the loop body have to be analyzed. There are certain situations in which the execution of certain statements of an execution path affects the number of executions of all the basic blocks along the execution path. Therefore, more accurate estimation of the number of loop iterations is provided. The results of analysis on the Mälardalen benchmark suite reveal the accuracy of the proposed method.

Key words: Loop bound, parametric worst-case execution time analysis, timing analysis, real-time systems

1. Introduction

In real-time systems, correct temporal behavior is an essential property of the systems. This means that the correctness of a real-time system depends not only on the computational results but also on the time at which the result is produced. In embedded systems with hard real-time properties such as automotive/avionics systems, to make sure that tasks can meet deadlines, the execution times of all tasks must be known. The execution time of a task may vary among different runs of the task on the same platform. Tasks' input and hardware states affect the execution time of the task. Therefore, to ensure the completion of a task within the allocated time for the task, it is important to estimate the longest possible time a task might take, known as the worst-case execution time (WCET).

Most of the static WCET analyzers estimate a constant value for the WCET of a program. However, in some cases such as dynamic scheduling it is required to predict the WCET in terms of the program inputs at runtime [1]. To this end, today it is observed that the parametric timing analysis is the state of the art in WCET analysis research [2–4].

The main issue concerning WCET estimation is the loop bound analysis. In parametric timing analysis, instead of estimating a single numeric value for a loop upper bound, a symbolic formula is driven from the loop body. The formula is further instantiated to estimate the loop WCET for different loop parameters.

There has been no general approach to parametric WCET, but there are a number of semiautomated approaches to WCET that require manual annotation of the loop body [5]. There are also a few automated

*Correspondence: sakhaei@iust.ac.ir

approaches, which put restrictions on the loop structure [1,6,7]. The proposed method in [8] derives WCET formulas automatically without any restrictions on the loop structure. The derived results are very accurate, but due to the use of polyhedral flow analysis, there is a problem in the cost of high complexity. The major difficulty concerning parametric WCET analysis is the control flow analysis to find constraints on the program flow. In an implicit path enumeration technique (IPET) [9], constraints affecting the execution of each basic block within a loop control flow graph are extracted. The execution count of each basic block can be derived from the constraints as a formula in terms of the loop parameters. The results are further multiplied by the WCET of their related basic block that is achieved by low-level analysis to estimate the overall WCET of each basic block. Using a parameter integer programming technique, the parametric WCET of the loop is estimated by maximizing the sum of the overall worst case execution times of the basic blocks [2,7,8]. However, computing the WCET by considering the execution time of the loop underlying basic blocks in isolation does not necessarily provide an accurate and precise estimate of the loop bound. There are certain situations in which the execution of certain statements of an execution path affects the number of executions of all the basic blocks along the execution path. To resolve the difficulty, in the method proposed in this paper, instead of basic blocks, the IPET method is applied to each execution path in an iteration of a loop body to estimate the number of iterations of all basic blocks along the execution path. As a result, more accurate estimation of the number of loop iterations is provided.

The remaining parts of this paper are organized as follows: in Section 2, a brief introduction to static WCET analysis is provided. Our proposed method to drive the parametric loop bound is presented in Section 3. The method contains two major phases: symbolic flow analysis and symbolic quantitative flow analysis. Section 4, on symbolic flow analysis, describes how to construct the path constraint and expressions representing how the value of the loop variables changes for each execution path within the loop body. Section 5, on symbolic quantitative flow analysis, describes the way the model counting algorithm is used to count the execution number of each execution path based on the loop invariant variables. Section 6 presents the related work. Evaluations and conclusions are stated in Sections 7 and 8.

2. Static WCET analysis

The WCET is the longest execution time that a program requires to be executed on a given target hardware [10]. Static WCET analysis provides safe upper bounds of the WCET of a program. A safe bound means that the estimated WCET will be larger than or equal to the real WCET. Typically, static WCET analysis is done in three phases, including flow analysis, low-level analysis, and calculation. Among these three phases, flow analysis is applied to the source or object code of a program in order to find constraints on the program flow. The constraints coming from the structure of a program are the examples of program flow constraints. Flow constraints can also be the bounds on loop iterations, or infeasible paths, which are the program paths that due to semantic constraints will never be taken in practice. A mathematical model of the hardware is used in low-level analysis to estimate the worst-case timing for the atomic units of a program. In order to achieve accuracy in WCET analysis, a low-level analysis normally considers complex hardware features such as caches and pipelines. Finally, in order to produce a concrete upper bound for the WCET, we can combine the results from flow and low-level analysis in the calculation phase. The IPET is a widely used technique for WCET calculation [10]. In this technique, different information from the other phases (flow analysis and low-level analysis) is represented as integer linear constraints and the WCET is regarded as a linear expression that should be maximized. An integer linear programming (ILP) solver is used to estimate the WCET of the program.

In IPET calculation, the program flow is modeled via arithmetical constraints. The effort is then to maximize the execution time of the program under these constraints. For each basic block, b, of the program, the WCET of the basic block, T_b, is computed by low-level analysis. Supposing that X_b represents the execution count of block b, the WCET of the program is obtained by maximizing the following summation formula:

$$\sum_{b \in B} T_b.X_b \tag{1}$$

where B is the set of all basic blocks of the program. X_b is constrained by the program structure and the program functionality [10]. The structural constraints are extracted from the CFG of program. For the CFG in Figure 1, the extracted structural constraints are as follows.

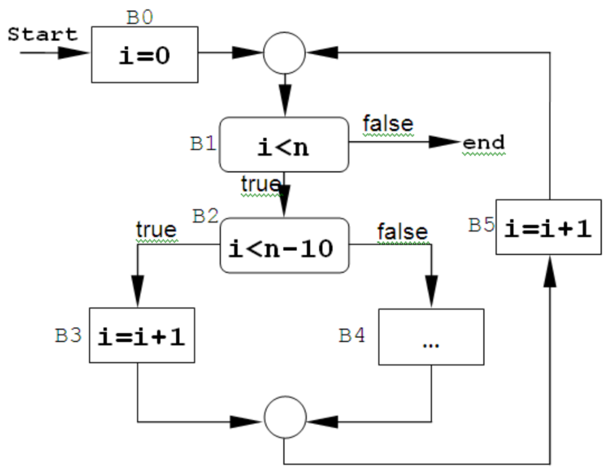

Figure 1. A simple CFG of a program [8].

$$X_{B0} + X_{B5} = X_{B1}$$

$$X_{B1} = X_{B2}$$

$$X_{B2} = X_{B3} + X_{B4}$$

$$X_{B3} + X_{B4} = X_{B5}$$

The functionality constraints denote loop bounds and other path information that depends on the functionality of the program. If n is equal to 15, the functionality constraints are as follows.

$$X_{B0} = 1$$

$$X_{B1} < 15$$

The constraint set that is achieved by combining structural and functionality constraints is passed to the ILP solver with Eq. (1) to be maximized. IPET just gives the worst-case count on each basic block and no information about precise execution order. In the proposed method, the IPET method is applied to each execution path in an iteration of a loop body, instead of a basic block, to estimate the number of iterations of all basic blocks along the execution path.

3. Method

In order to estimate the parametric WCET of a loop, all different execution paths within the loop body are identified. For each execution path, p, the execution count, X_p, subject to the path constraint of p is computed. Considering T_p as the execution time of p, the overall WCET of the loop is estimated as follows:

$$WCET_{Loop} = \max(\sum_{p \in P} T_p . X_p) \tag{2}$$

It is assumed that T_p is already estimated through low-level analysis of the loop body [10]. Here, the main concern is to estimate the execution count, X_p. Our parametric loop bound estimation algorithm (PLEA) is described in Figure 2. We have applied a symbolic flow analysis method to extract the parametric flow information of the parametric loops. The flow information includes the paths' constraints and a set of symbolic expressions representing how the value of the loop variables changes. The path constraint characterizes the loop variables' value assignments for which the loop iterates through the path. Considering the paths' constraints, the number of iterations of each execution path is estimated by applying a model counting algorithm. The model counting addresses the problem of computing the number of distinct variable assignments for which a constraint evaluates to TRUE.

In Step1 of the PLEA it is assumed that a low-level analysis method is already applied to estimate the WCET of the loop components. In Step2, for each execution path, the path constraint is constructed. The constraint constructed for each execution path characterizes the loop variables' value assignments, or in other words constraint models, for which the loop iterates through the execution path. In Step3, the symbolic expressions are built that represent how to change the values of the loop variables is done. Using these expressions, the number of values assigned to the loop variables is verified to improve the preciseness of the loop bound estimation. Step2 and Step3 are described in Section 4 as symbolic flow analysis.

In Step4 and Step5, a model counting algorithm is invoked to return a satisfying assignment or model for all variables appearing in the constraint formulas, built in Step2. Using a parametric model counting technique, it is possible to count the number of loop iterations through an execution path based on the loop invariant variables. A detailed description of Step4 and Step5 of the algorithm is presented in Section 5.

3.1. An example

In this section, to clarify the applicability of the PLEA, a worked example is presented. Consider the CFG in Figure 1, including a loop with two different execution paths:

$$\begin{aligned} &ep\#1 : B1B2B3B5, \\ &ep\#2 : B1B2B4B5. \end{aligned} \tag{3}$$

Applying a symbolic flow analysis, in Step2 of PLEA, the path constraints *pc#1* for *ep#1* and *pc#2* for *ep#2* are constructed as follows:

$$\begin{aligned} &pc\#1 : (i >= 0) \text{ and } (i < n) \text{ and } (i < n - 10), \\ &pc\#2 : (i >= 0) \text{ and } (i < n) \text{ and } (i >= n - 10). \end{aligned} \tag{4}$$

```
Algorithm PLEA
Input: A loop CFG
Output: $PWCET_{loop}$
Method:
Step1- // Low-level analysis
  Compute WCET of all loop components including
  $WCET_{ep}$ : WCET of different loop execution paths,
  $WCET_{tp}$ : WCET of different termination paths,
  $WCET_{S}$ : WCET of statements, executed only once;
Step2- // Construct path constraint, PC, for each exec. path.
  For each execution path, ep do
      Construct path constraint, $PC_{ep}$ // (see Figure 3)
Step3- // Build expressions representing how the value of loop
        // variables changes along each execution path, ep.
  For each execution path, ep do
      For each loop variable lv in ep do
          Build a symbolic expression, $CV_{ep,lv}$ , representing how
              change in value of lv occurred along ep.//(see Figure 4)
Step4- //Compute no. models for each path constraint, $PC_{ep}$.
   Compute No. $Models_{PCep}$for each $PC_{ep}$ considering
       loop invariant variables, using a model counting algorithm [11].
Step5- //Compute step length for each execution path
  For each execution path, ep, do
       $SL_{ep}$ = Step length of ep//(see Figure 6)
Step6- // Solve the following optimization problem using parametric
       //integer linear programming (PIP) techniques [12]
  $PWCET_{Loop}$= $WCET_{S}$
                 + MAX ($\sum_{All\ EP} WCET_{ep} * \lceil No.\ Models_{PCep}/SL_{ep} \rceil$)
                 + $MAX_{ALL\ TP}(WCET_{tp})$
Step 7- // At the end return the parametric WCET of loop
  Return $PWCET_{LOOP}$
```

Figure 2. Parametric loop bound estimation algorithm.

In Step3, symbolic expressions representing change in value of the loop variable i, $CV_{1,i}$, in execution path *ep#1* and $CV_{2,i}$ in execution path *ep#2* are built as follows:

$$\begin{aligned} CV_{1,i} &: i = i + 2, \\ CV_{2,i} &: i = i + 1. \end{aligned} \tag{5}$$

In Step2, the term $(i >= 0)$ is also added to the path constraints *pc#1* and *pc#2* because initially the value of i in this path is zero and this value is increased in each iteration of the loop.

Suppose that the value of the loop parameter n is equal to 15. A model for *pc#1* would be $i = 3$, which represents the variable assignment for which *pc#1* evaluates to *TRUE*. Using a model counting algorithm [11], the number of models evaluated to *TRUE* for *pc#1* will be 5. For *pc#1*, the models, i.e. the values that can be assigned to the loop variable i, will be equal to *0*, *1*, *2*, *3*, and *4*. However, according to changes in value of the loop variable i through *ep#1*, $CV_{1,i}$, the step length of loop variable i is equal to *2*. Therefore, models *1* and *3* are not visited while the loop iterates through *ep#1*. Accordingly, the maximum number of iterations through *ep#1* will be $\lceil 5/2 \rceil = 3$.

If loop invariant variables, such as n, cannot be statically determined at the beginning of the loop, the number of models would be parametric. The number of models, *#models*, derived in Step4 and Step5 of PLEA for the constraints *pc#1* and *pc#2* are as follows.

$$X_1 = \#models(pc\#1) = \begin{cases} \lceil (n-10)/2 \rceil & n \geq 11 \\ \\ 0 & \text{other} \end{cases}$$
$$X_2 = \#models(pc\#2) = \begin{cases} 10 & n \geq 11 \\ n & 0 < n \leq 10 \\ 0 & \text{other} \end{cases} \tag{6}$$

At the end, after the last iteration of the loop body through its execution paths *ep#1* or *ep#2*, the loop termination path, *tp*, is executed and the iteration of the loop is terminated. In this example, there is only one loop termination path, *tp#1*, which is:

$$tp\#1\text{: } B1\ end. \tag{7}$$

In the last step of the PLEA, a parametric IPET calculation is performed to derive a parametric WCET as a symbolic equation for the loop.

Suppose the WCET of the loop components, estimated via a low-level analysis method, are as follows:

$$\begin{aligned} &WCET(B0) = T_0 WCET(ep\#1) = T_1, \\ &WCET(ep\#2) = T_2 WCET(tp\#1) = T_3. \end{aligned} \tag{8}$$

The loop WCET is calculated by the following formula:

$$PWCET(loop) = MAX(T_0 + (\sum_{i=1,2} T_i X_i) + T_3). \tag{9}$$

The constraints $X_i \leq P_i$ must be added, and then maximizing Eq. (9) subject to these constraints with PIP [12] is done. The solution will depend on parameters P_i. Let us assume in the WCET of loop components in Eq. (9) that $T_0 = 10$, $T_1 = 50$, $T_2 = 80$, and $T_3 = 10$. The loop WCET is obtained by PIP as follows.

$$\begin{aligned} &PWCET = 20 + 50P_1 + 80P_2 \\ &\text{for } P_1 = X_1 \text{ and } P_2 = X_2 \end{aligned} \tag{10}$$

Thus, by substitution of P $_i$ from Eq. (6) into Eq. (10), the following is obtained.

$$PWCET(loop) = \begin{cases} n*25+570 & n \geq 11 \\ n*80+200< & n \leq 10 \\ 20 & \text{other} \end{cases} \tag{11}$$

4. Symbolic flow analysis

In Step2 and Step3 of the PLEA, symbolic expressions representing the path constraints and how the value of the loop variables changes through each execution path are built. In order to extract these expressions for each execution path, in the control flow graph (CFG) of the loop, the execution paths within the loop body should be identified. An execution path, EP, is a sequence of finite nodes in the CFG of a loop, starting from the head of the loop to the start node of a loop back-edge so that each node appears only once in the sequence. In fact, each branch condition within a loop may originate two different execution paths across the loop body. A branch condition addresses the condition, C, in a branch node, B, e.g., an if or while statement, determining the node to be executed immediately after the branch node. A path constraint, PC, for an execution path, ep, is defined as follows.

$$PC(ep) = \bigwedge_{b \in ep} C(b) \tag{12}$$

A path constraint is formed as the conjunction of all the conditions $C(b)$ of branch node b along an execution path, *ep*.

The path constraint characterizes the variables' value assignments for which the program is executed through the path.

As shown in Figure 3, a path constraint is formed as the conjunction of all the branch conditions along an execution path. The branch conditions are extracted while traversing the CFG backward and along the execution path form the source node of the loop back-edge within the execution path to the loop entry point.

The value of a branch condition depends on the value of its constituents at the program point where the branch condition is located. To eliminate the dependencies, all the branch conditions are expressed in terms of the variables whose values are not defined along the subpath of the execution path, from the loop entry point up to the point where the branch condition is located [13]. Therefore, the path constraint stratification is performed base on the program state at the entry point of the loop. The program state is represented by a tuple whose element represents the value of variable.

A path constraint constituent could be one of the following:

- **Constant value.**
- **Loop invariant variable:** a variable whose value is defined before the loop body and not altered across the loop body.
- **Loop variable:** a variable whose value is modified within the loop body.

Loop invariant variables are considered as parameters and applied as constituents of a parametric expression, estimating the number of the loop iterations.

A path constraint is conjunct with the loop variable constraints. A loop variable constraint for a loop variable i depends on whether the initial value, V_i, of i at the loop entry point is increased or decreased within

```
PC(execution-path ep)
Begin
  Traverse backwards ep starting from its back-edge
  For each branch condition, bc, along ep do
    For each loop variable, lv, in bc do
      Replace lv with the expression computing lv at ep entry point;
    End For
    $PC_{ep}$ = strcat($PC_{ep}$, “and”, bc);
  End For;
  For each loop variable, lv, in PC do
  Begin
    lv-InitalVal = value of lv at the loop entry point;
    If the value of lv is increased within the loop body  Then
      lv-constraint = “lv >= lv-InitialVal”
    else
      lv-constraint = “lv <= lv-InitialVal”;
    $PC_{ep}$ = strcat($PC_{ep}$, “and”, lv-constraint);
  EndFor
Return $PC_{ep}$
```

Figure 3. Constructing path constraint.

the loop body is $i \geq V_i$ or $i \leq V_i$, respectively. We only analyze the loops with monotone loop variables. A loop variable is monotone if, on all execution paths through the loop body, it is incremented by the expressions that are either all positive or all negative.

In Step3, the symbolic expressions representing how change in values of the loop variables occurs are built. The symbolic expression reflects the operations performed on the loop variables through each execution path. The operations include those statements that directly or indirectly affect the value of the loop. Applying a data flow analysis, all the statements affecting the loop variable are located [13].

As described in Figure 4, expression(exp),which represents how to change the value of a loop variable *lv* along the execution path *ep*, is built by backward traversal and analysis of the statements along the execution path *ep*. The traversal begins at statement *stm* and ends with the loop entry point. The statement *stm* in the first call of *chgValOfLv*, in Figure 4, is the source node of the back-edge across *ep*.

5. Symbolic quantitative flow analysis

Symbolic quantitative flow analysis is concerned with computing the count of models of a path constraint. The model counting is the problem of computing the number of distinct variable assignments for which a constraint is evaluated as TRUE. The model counting problem is formalized as counting the integer points $x \in Z_d$ such that $Ax \geq b$ is satisfied, where A is an integer matrix and b is an integer vector [14].

In fact, a model of a path constraint represents a program state that satisfies the path constraint. In other words, this program state includes the value of loop variables that causes the path constraint to be evaluated as true. Hence, all models of the path constraint will be equivalent to all possible program states that can satisfy

```
Function chgValOfLv(lv, ep, stm)
 While traversing backwards ep starting from stm
     Find the first statement S as follows:
         S:  lv = Any_Expression;
 If (found)
     build a symbolic expressions, exp, representing
     the Any_Expression within S;
 For each variable, v, appearing in exp do
     replace v with chvalOfLv(v, ep, S)
Return exp
```

Figure 4. Building the expressions that represent how change in value of the loop variable is done.

the path constraint. Therefore, the number of these program states will be equal to the maximum times of iteration through this path. In Figure 5, a constraint and its graphical representation are shown. The program state *(7, 6)* is a model. If the variables i and j in the constraint are replaced by their values in program state *(7, 6)*, the constraint is evaluated to be true. All 65 integer points that are specified in Figure 5 satisfy the constraint. Therefore, if the constraint represents the constraint of an execution path within the loop body, the maximum number of iterations that the loop may iterate through this path is equal to 65.

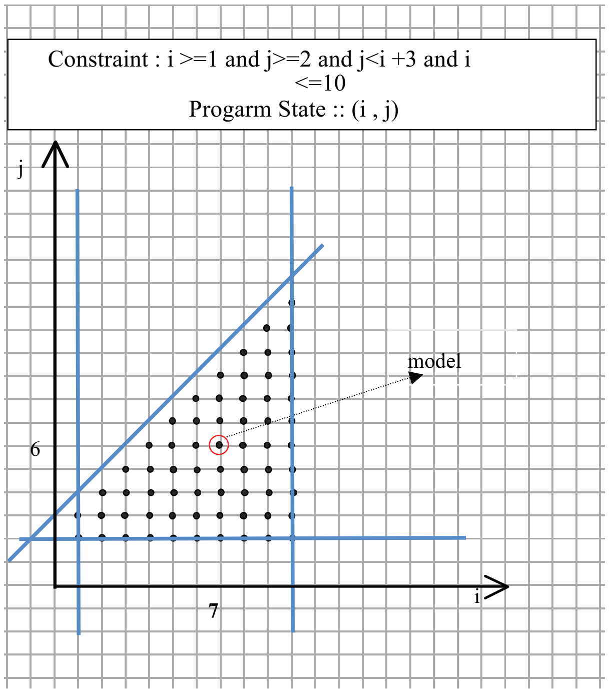

Figure 5. A constraint and its graphical representation.

To derive a loop parametric WCET, the parametric execution count of each execution path within the loop body is computed [8,15–17]. To achieve this, a parameterized model count for each path constraint is computed. A parameterized model counting method computes the integer points, x, as a function of the parameter vector, p, such that "$Ax \geq Bp + c$" is satisfied for the integer matrices A and B and the constant integer vector c [11]. In this paper, an extension of Barvinok's algorithm [11], implemented in the Barvinok tool (www.kotnet.org/˜skimo/barvinok/), is used to compute the parametric execution count of a loop execution path.

In Step4 and Step5 of the PLEA, described in Figure 2, the parametric execution count of each execution path within a loop body is computed symbolically. To achieve this, the parameterized model count of each path constraint is computed. Assume that L is a loop containing a number of different execution paths, $ep \in EP$, and a number of termination paths, $tp \in TP$, and a set, S, of instructions, which are executed only once in L. The execution count, X_{ep}, of each execution path *ep* is computed as follows.

$$X_{ep} = \lceil \#model(C_{ep})/SL_{ep} \rceil \tag{13}$$

In the above relation, SL_{ep} indicates the step length of *ep* and $\#model(C_{ep})$ is the model count of the *ep* path constraint, C_{ep}. In Step5 of the PLEA algorithm, the step length, SL_{ep}, of each execution path, *ep*, is used. It is described in Figure 6.

```
StepLength(ep)
  Let I is a model of $CP_{ep}$ in the form of $(lv_{I1},\ldots,lv_{In})$
              that $lv_{Ik}$ is a loop variable
  Let next model, J in the form of$(lv_{J1},\ldots,lv_{Jn})$ and $lv_{Jk} = CV_{ep,} * X_{ep}$
  For each $lv_k$ do
    delta-$lv_k$ = compute Min$(lv_{Ik} - lv_{Jk})$
  $SL_{ep}$ = MIN $_{\text{all lv}}$ (delta-lv)
```

Figure 6. Computing the step length of execution path.

Applying a low-level analysis method to compute the WCET of S, *ep*, and *tp*, the parametric WCET, $PWCET$, of L is derived as follows.

$$\begin{aligned} PWCET(L) &= MAX(WCET_S) \\ &\quad + MAX(\textstyle\sum_{AllEP} WCET_{ep} * X_{ep}) \\ &\quad + MAX_{ALLTP}(WCET_{tp}) \end{aligned} \tag{14}$$

This is an optimization problem that can be solved by applying the parametric integer linear programming (PIP) techniques [12].

In a nested loop, the inner loop is considered as a statement for which the WCET is already derived as a parametric formula.

However, this is an overestimation for nonrectangular loop nests. The number of iterations of the inner loop within the nonrectangular loop nests depends on the loop variables of the outer loops. Therefore, the loop variables of the outer loops will be the loop invariant variables of the inner loop. The WCET of the inner loop will be parameterized on the loop variables of the outer loop. In this respect, the WCET of the path including the inner loop may vary for each model of the path constraint, which results in a loss of precision.

6. Related work

Recently, a number of research groups have addressed various methods for parametric WCET analysis. In [3] a measurement-based method is presented that does not guarantee a safe bound. The safe bound means that the estimated WCET is larger than or equal to the real WCET. For parametric WCET analysis, most existing methods are based on static analyses that are different in automating the identification of parameters as well as the analysis method, the loop structures, and the level of the code representation.

In [5] the manual annotations of the source code are required to define the parameters that affect the WCET of a piece of code. These annotations also define the expressions that describe the WCET of a section code. The manual annotation is a labor-intensive and error-prone process for which preserving updates is hard due to the changes in program code. The proposed method in this paper automates the process of parameter identification as well as constructing the expressions that describe the WCET. The presented methods in [6] and [18] are concentrated on the parametric loop bound analysis for the nested loop. In [1] the parametric expressions are derived using fixed-point caching behavior. This method only handles the well-structured nonrecursive code, and it does not handle the cases in which there are nested parametric expressions inside the loop nests. In [7] the parameters of the program are identified and the dependencies between the parameters and variables are computed. Using these dependencies, the parametric loop bound for the parameterized loops is derived as an expression. At the end, using IPET calculations, a symbolic ILP, which is solved to derive the parametric timing formula, is constructed. The proposed parametric loop analysis is not accurate enough [2], since the dependencies are extracted through the value analysis method, which only considers the simple dependencies at the basic block. Since the analysis is done on a machine code, the estimated WCETs of the basic blocks are closer to the real WCET. The proposed method is based on the intermediate code (byte code), which facilitates the flow analysis in addition to tightening the estimated WCET to the real value.

The PLEA is similar to [8] and [2] in IPET calculation. In [8] and [2] the constraints of the program flows are extracted from the program code using polyhedral abstract interpretation. There is a tradeoff between cost and precision of analysis. Although those methods are more accurate than our method, due to the complexity of the polyhedral abstract interpretation they may have problems for complex programs. The complexity of polyhedral abstract interpretation is $O\ (2^n)$ both in space and time. Our method uses symbolic data flow analysis instead of polyhedral abstract interpretation, which has a linear complexity. Instead of extracting the constraints according to the basic blocks in [2] within a loop, we extract the constraints of the execution paths within the loop body, causing the infeasible paths to be considered in the analysis to increase the estimation precision. The number of basic blocks within the loop body for more than 90% of the loops is in practice less than 10 basic blocks [19]. Thus, the number of enumerated paths is not too large.

By considering how the value of the loop variables changes across the execution paths, the execution count of the execution paths can be estimated more accurately. Note that since our method is based on the intermediate code rather than the source code, the estimated WCET is more precise and accurate.

7. Evaluation

The main contribution and significance of our proposed method is to estimate the execution count of the execution paths within a loop body. In this respect, the experimental results presented in this section are obtained to evaluate this significant characteristic of the proposed method. The experiment is carried out with a number of programs included in the Mälardalen benchmark suite [20]. These programs contain context-sensitive loops whose iteration numbers may vary in different invocations of the loops. The programs are listed in Table 1. We have selected these programs because they satisfy all the constraints of the PLEA method. The PLEA method assumes all the variables to be of integer or Boolean type. Furthermore, the value of the variables of the type pointer and array are simplified by treating them as unknown values. Function calls are emulated by in-lining. Switch statements and unstructured control flow are not supported.

Table 1. Loop bound analysis of Mälardalen benchmark.

Program	#B	#P	%P
Adpcm	5	2	40
matmult	2	0	0
Crc	3	0	0
fft1	4	1	25
loop3	2	2	100
Total	16	5	31

In Table 1 the symbols *#B*, *#P*, and *%P* indicate the number of bounded loops, the number of bounded loops whose step lengths are more than one, and the percentage of these loops bounded, respectively. Those loops whose step lengths are more than one can be bounded precisely by applying our method, the PLEA.

Since the PLEA does not provide any low-level analysis, the WCET of all the program statements is assumed to be the constant of 10 clock cycles and the WCET of the function calls is assumed to be about 200 clock cycles.

As shown in Table 1, our proposed parametric loop bound estimation method can build a symbolic formula for *Adpcm* loops. The step lengths of about 40% of these loops are greater than one and, as described in Sections 3 and 4, the execution counts of these loops can be formulated by applying our method only. In Figure 7, the two loops, *For #1* and *For #2*, included in the main function of the *Adpcm* program are listed. The parametric WCET formula for these loops, derived by applying the PLEA method, is presented in Table 2. For different values of the parameter *IN_ END*, the WCET computed by the formulas derived by the PLEA and the WCET computed without considering the changes in values of loop variables ($WCET_{WCV}$) are presented.

Table 2. WCET of the Adpcm program loops.

Function	WCET formula derived By PLEA	IN_ END parameter	PLEA WCET	WCET_{WCV}
main/For#1	20+IN_ END/2× 220	10	1120	2220
		50	5520	11020
		100	11020	22020
main/For#2	20+IN_ END/2× 240	10	1120	2240
		50	6020	1220
		100	12020	24020

In the PLEA, a constraint is imposed on a path rather than a basic block within a loop body. Using the constraints on paths results in more precise estimation of the WCET because of analyzing infeasible paths [21,22]. If the number of models for a path constraint is equal to zero, the path is certainly an infeasible path. The infeasible paths of the inner loop in the complex function of the *janne_ complex* benchmark program, shown in Figure 8, are as follows.

Infeasible Path 1: $S_4 S_5 S_8 S_9$

Infeasible Path 2: $S_4 S_6 S_7 S_8 S_9$

Hence, in Eq. (14) for these infeasible paths, X_{ep} is equal to zero and these paths do not have an impact on the calculated WCET bound by the PLEA. Because of ignoring infeasible paths, the number of enumerated paths will also be reduced.

```
//Adpcm main function loops
…
main(){
…
//For #1
for (i = 0; i < IN_END; i += 2){
     compressed[i / 2] = encode(test_data[i], test_data[i + 1]);
//For #2
for (i = 0; i < IN_END; i += 2) {
                    decode(compressed[i / 2]);
                    result[i] = xout1;
                    result[i + 1] = xout2;
          }
…
}
```

Figure 7. Source code of Adpcm benchmark.

```
//janne_complex, complex function
…
(S1) intcomplex(int a, int b)
(S2) {    while (a < 30) {
(S3)        while (b < a) {
(S4)            if (b > 5)
(S5)                    b = b * 3;
(S6)            else
(S7)                    b = b + 2;
(S8)            if (b >= 10 && b <= 12)
(S9)                        a = a + 10;
(S10)           else
(S11)                           a = a + 1;
            }
(S12)       a = a + 2;
(S13)       b = b - 10;
         }
(S14)   return 1;
      }
```

Figure 8. Source code of janne_ complex benchmark.

Because the benchmark programs are small, the analysis time of these programs is less than 1 s and it is not possible to compare the methods based on analysis time.

8. Conclusion

Loop iterations are most often dependent on a number of parameters defined before the execution of the loop body. Therefore, the loops of the WCET could be best defined in terms of the parametric formulas. Computing a parametric WCET by considering the execution time of the loop underlying basic blocks in isolation does not necessarily provide an accurate and precise estimate of the loop bound. There are certain situations in which the execution of certain statements of an execution path affects the number of executions of all the basic blocks along the execution path. To resolve the difficulty, instead of considering the basic blocks on their own and independent of the rest, the execution paths within the loop body have to be analyzed. In this way, when analyzing an execution path, changes in the value of the loop variables could be captured through data flow analysis of the loop variables along the execution path. This allows us to calculate the step length of each iteration along the execution path that leads to accurate estimation of the number of iterations for the execution path. When estimating the number of iterations of the basic blocks, to achieve a tight estimation, in addition to the basic blocks, the execution paths defining interdependencies among the basic blocks and also all the constraints imposed by the infeasible paths have been considered. The PLEA is also advantageous for tight estimation of the WCET in low-level analysis of programs because of considering paths rather than basic blocks.

While our proposed method is advantageous in terms of computational cost compared to polyhedral abstract interpretation methods, the path constraints extracted by the PLEA could be unbounded in rare situations. This limitation will be addressed in our future work, where we aim to resolve the problem while preserving the low computational cost of the algorithm. The large number of paths in a loop body is a challenging problem for path enumeration algorithms. However, it has been shown that the number of basic blocks in a loop body may not exceed 10 in practice. Therefore, the number of resultant paths in a loop body is limited and can be easily handled for real-world applications. Furthermore, because of pruning the branch conditions with no impact on loop iterations, we have decreased the number of subject paths for further analysis.

Acknowledgment

The authors express appreciation to Dr Björn Lisper for his valuable and constructive suggestions.

References

[1] Vivancos E, Healy C, Mueller F, Whalley D. Parametric timing analysis. In: Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems; 2001; Snowbird, UT, USA. pp. 88-93.

[2] Bygde S, Ermedahl A, Lisper B. An efficient algorithm for parametric WCET calculation. J Syst Architect 2011; 57: 614-624.

[3] Marref A. Evolutionary techniques for parametric WCET analysis. In: 12th International Workshop on Worst-Case Execution Time Analysis; 2012; Pisa, Italy. pp. 103-115.

[4] Huber B, Prokesch D, Puschner PP. A formal framework for precise parametric WCET formulas. In: 12th International Workshop on Worst-Case Execution Time Analysis; 2012; Pisa, Italy. pp. 91-102.

[5] Bernat G, Burns A. An approach to symbolic worst-case execution time analysis. In: Proceedings of the 25th Workshop on Real-Time Programming; 2000; Palma, Spain.

[6] Coffman J, Healy C, Mueller F, Whalley D. Generalizing parametric timing analysis. In: Proceedings of the 2007 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools; 2007; New York, NY, USA. pp. 152-154.

[7] Altmeyer S, Hümbert C, Lisper B, Wilhelm R. Parametric timing analysis for complex architectures. In: Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications; 2008; Kaohsiung, Taiwan. pp. 367-376.

[8] Lisper B. Fully automatic, parametric worst-case execution time analysis. In: Proceedings of the 3rd International Workshop on Worst-Case Execution Time Analysis; 2003; Porto, Portugal. pp. 77-80.

[9] Li YTS, Malik S. Performance analysis of embedded software using implicit path enumeration. In: Proceedings of the 32nd Design Automation Conference; 1995. pp. 456-461.

[10] Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T et al. The worst-case execution time problem – overview of methods and survey of tools. ACM T Embedded Comput Syst 2008; 7: 1-53.

[11] Verdoolaege S, Seghir R, Beyls K, Loechner V, Bruynooghe M. Counting integer points in parametric polytopes using Barvinok's rational functions. Algorithmica 2007; 48: 37-66.

[12] Feautrier P. Parametric integer programming. RAIRO Recherche Opérationnelle 1998; 22: 243-268.

[13] Parsa S, Sakhaei-nia M. Modeling flow information of loops using compositional condition of controls. The journal of supercomputing. 2015; Feb 1; 71(2): 506-36.

[14] Bayardo RJ, Pehoushek JD. Counting models using connected components. In: 17th International Conference on Artificial Intelligence; 2000. pp. 157-162.

[15] Clauss P, Loechner V. Parametric analysis of polyhedral iteration spaces. J VLSI Signal Process 1998; 19: 179-194.

[16] Loechner V, Meister B, Clauss P. Precise data locality optimization of nested loops. J Supercomput 2002; 21: 37-76.

[17] Beyls K, D'Hollander E. Generating cache hints for improved program efficiency. J Syst Architect 2005; 51: 223-250.

[18] van Engelen RA, Gallivan KA, Walsh B. Parametric timing estimation with Newton-Gregory formulae. Concur Comp-Pract E 2006; 18: 1435-1463.

[19] Engblom J. Static properties of commercial embedded real-time programs, and their implication for worst-case execution time analysis. In: Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium; 1999; Vancouver, Canada. pp. 46-55.

[20] Gustafsson J, Betts A, Ermedahl A, Lisper B. The Mälardalen WCET benchmarks – past, present and future. In: Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis; 2010; Brussels, Belgium. pp. 137-147.

[21] Lisper B. Fully Automatic, Parametric Worst-Case Execution Time Analysis. MRTC Report. Västerås, Sweden: Mälardalen Real-Time Research Centre of Mälardalen University, 2003.

[22] Chen T, Mitra T, Roychoudhury A, Suhendra V. Exploiting branch constraints without exhaustive path enumeration. In: 5th International Workshop on Worst-Case Execution Time Analysis; 2005.