Journal of mathematics and computer science 8 (2014), 205-214

Contents list available at JMCS

Journal of Mathematics and Computer Science

Journal Homepage: www.tjmcs.com

A XML-Based Representation of Timing Information for WCET Analysis

Saeed Parsa[1]
Department of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran

parsa@iust.ac.ir

Mehdi Sakhaei-nia
Department of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran

sakhaei@iust.ac.ir

Article history:
Received June 2013
Accepted July 2013
Available online July 2013

Abstract

The Worst-Case Execution Time (WCET) analysis is an important stage in development process and verification of hard real-time systems. In this article the use of XML as a standard for exchanging timing information amongst timing analysis tools is proposed. Timing information resulted from automatic analysis of programs can be represented in XML format. Considering the type of information required for estimating the worst case execution time of programs, a set of XML tags is offered in this paper. Timing information resulted from analyzing a program by a timing analysis tool could be annotated within the program. The annotated code could be simply applied by other tools for relatively more accurate estimation of the worst case execution times. The paper also clears the way for future studies on using XML-based representation for extraction of information.

Keywords: Real-time systems; WCET; program representation

1. Introduction

The validity of hard real-time systems strongly relies on the upper bounds of their task's WCET. Reliable and efficient execution schedules for real-time systems result from safe and sharp WECT estimation of the systems tasks.

There are two major steps in static estimating a program WCET. In the first step timing information including the execution time of the program constructs is extracted from the program code. In the second step considering the execution sequences of the program constructs, the WCET of the whole program is estimated. In general, the main parameters affecting a program execution time could be

[1] Corresponding Author

summarized as timing information are gathered through analysis of the program control flow of the program and the behavior of the processor on which the program is supposed to be executed. Control flow analysis is mainly concerned with the analysis of loop bounds and program execution paths. Processors behaviors are affected by the behavior of the components that influence the execution times, such as caches, memory, pipelines and branch predictions. Processor behavior analysis is aimed at determining upper bounds on the execution times of instructions or basic blocks.

There are a number of tools [25] which consider these parameters when collecting timing information to estimate a program WCET. Each of these tools is aimed at a particular aspect of the program execution and has its own strengths and weakness. For instance, Bound-T [25] mainly considers the effect of loops on the program execution time. Bound_T does not support cache analysis, but estimates loop bounds automatically and detect infeasible execution paths while analyzing loop bounds. Unlike Bound_T, Florida [25] puts more emphasis on cache analysis, and performs limited data flow analysis to compute loop bounds. Chronos [25] is another known timing analysis tool which mainly considers low level details of the underlying platform or in other words the processor behavior. Chronos estimates loop bounds for limited loops by data flow analysis and requires to its users' feedback for detecting infeasible execution paths. aiT [25] estimates upper bounds for the execution times of code snippets in executables.

It is observed that each of these timing analysis tools has its own merits and pitfalls. To benefit the merits of varieties of the tools, a unified representation of the timing information provided by the tools could be advantageous. In this article a set of XML tags for representing timing information is proposed. The proposed XML tags are inserted as comment statements within the programs to facilitate the delivery of timing information amongst different timing analysis tools. Apparently, when annotating a program with XML tags, there will be no need for specific parsers to automatically extract timing information from the annotations.

The remaining parts of this paper are organized as follows: In Section II, the main concepts behind WCET analysis are described. Specifying requirements for the representation is included in Section III. Defining parts of the proposed representation and reviewing the related work are included in Section IV and V respectively. Finally in Section VI and VII, discussion and conclusions are presented.

2. Static analysis of WCET

A In static analysis the estimation of execution time is done through three different stages which are themselves affected by different factors [7]. These stages are as follows:

Control flow analysis: it analyzes the source, intermediate or object code of program and specifies the possible flows through the program. For example it determines the possible sequences of instructions that may be executed.

Processor-behavior analysis: in this stage, analysis is done on object code and target hardware to determine the timing information for execution of instruction on target hardware. As for modern processors with pipeline and cache, this part is of high importance.

Calculation: includes the combination of results from processor-behavior and control flow analysis which yields the estimation of program execution time.

2.1. Control flow analysis

The purpose of control flow analysis is to determine the possible flows of the program and to anticipate its dynamic behavior. The results of this stage include information about how to call

functions, the number of loop iteration and the dependency between if-else statements. This analysis should be done carefully.

The flow information can be extracted from source-, intermediate- or object-code of program or even the information gathered by compiler. This stage includes three different sub-stages as follows:

- Flow extraction: achieving the flow information by manual annotation or automatic approaches of flow analysis;
- Flow representation: representing the results of previous stage and integrating the results of different methods of the flow extraction; and
- Calculation conversion: converting the information of the represented flow to a form which can be used in final calculation stage.

2.2 Processor-behaviour analysis

The purpose of processor-behavior analysis is to determine the execution time for each atomic unit of the flow based on the architectural feature of the target hardware. To achieve the actual timing behavior of the program, processor-behavior analysis must be performed on object code of program. This stage also includes two different sub-stages:

- Global: determines the effects of machine-dependent factors that should be modeled over the entire global program. These factors include instruction cache, data cache, prediction of branches and translation lookaside buffers.
- Local: determines the effects of machine-dependent factors that can locally handled on instruction and neighbor's instruction. These factors include the speed of accessing memory or the overlaps resulted from pipeline.

2.3. Calculations

The purpose of this stage is to calculate the WCET estimate for the program based on the information gathered from the previous two stages. There are generally three approaches for this stage:

- Tree-based: In this approach the syntax tree of program is considered as a tree structure. By bottom-up traversing of the tree and applying different timing rules at the nodes (called “timing schema”), the estimated time can be achieved.
- Path-based: In this approach, the execution time is calculated in different feasible execution paths. Based on the results, the path with the longest time is specified and then the WCET is estimated.
- Implicit Path Enumeration Technique (IPET): In this approach, using arithmetical constraints to model the program flow and attempt to maximize the execution time of the entire program under these constraints. Calculations are done via Integer Linear Programming (ILP) formulas.

3. Requirements for proposed XML representation

The proposed representation, in addition to contain the timing information of the program code, should be able to facilitate the estimation of execution time. This section describes the structures in the program which affect the execution time estimations and contain timing information. The representation should be able to preserve this information. According to what was stated in Section II, based on the processor-behavior and control flow analysis, the structures within the program code which affect WCET analysis are elaborated on in subsection A and B. Then, the reasons why XML should be used are describe in subsection C.

3.1. Structures effective in control flow analysis

Input parameters: data received in different parts of the program as input parameter affect the program execution path and time. These parameters are divided into two groups. First are the value entered from outside of program. The limitation in this part can be alleviated by manual annotations in code. That is because these data are created outside the program and their real values are only specified during the execution of the program. The second group includes the internal values which are determined by processing and calculating the values of variables based on static analysis [7].

Loop bound and depth of recursive function call: the loop bound is important factors in estimation of program execution time. However, determining a bound for loop is problematic. Of course, if this is done without error, annotating the loop iteration bound in program code can solve the problem. This problem also exists for the depth of recursion call. Of course, Tried to estimate these bounds automatically [7] [23].

Infeasible paths: there are different execution paths in a program code. An infeasible path is an execution path allowed by the static structure of the program, but not possible when the semantics of the code is taken into account [7]. Determining these paths affects the specification of the longest path and therefore the estimation of WCET [24][7].

Conditional and branching statements: In conditional statements, based on the result of conditional expression evaluation, one of the two possible paths of if-then and if-else will be executed. It is obvious that the time of execution of the whole block depends upon the selection of the path. In switch-case statements the time of execution is depend on our selection too.

3.2. Factors effective in processor-behavior analysis

As mentioned in Section II, in order to calculate the execution time of each path, the execution time of each instruction along that path should be specified exactly. This time will not be the exact sum of the execution time of instructions on this path, because the features of modern processors help some of these instructions to be executed simultaneously and the time of their execution overlaps each others. The processor-behavior part of WCET analysis is depending on the features of processors [22].

Execution time of an individual instruction: each instruction, in programming languages depending on the machine upon which it is executed, has its own time of execution which is independent of other instructions. The execution time for each instruction should be specified in advance.

Pipeline: One of the features of modern processors is pipeline. A pipeline, by simultaneous executing of instructions in different stages can accelerate the instructions execution, these stages are instructions fetch, decoding, executing and writing the results.

Speculative Execution: Speculative Execution depends on the pipeline. Branch instructions will cause a stall in pipeline. In each piece of a program code there are many conditional statements and the goal is to determine the instructions which follow this branch. It should be mentioned that just some of these predictions can be true completely [7].

Cache: Another feature of modern processors is cache memory. In a computer system the main memory is RAM which is so slow in comparison to the processor speed. The cache memory is faster of RAM which stores copies of the blocks from the most frequently used main memory locations. The modern processors have two independent caches: an instruction cache to speed up instruction fetch, a data cache to speed up data fetches [7][17].

Instruction Level Parallelism (ILP): Most of the processors are able to execute different instructions in a parallel way. This issue also makes problems for stimulating the time of program execution. In order to find those instructions that can be performed in a parallel way, we need to determine the

dependency among instructions before the run time. This dependency can be data or control dependency [22].

It is important to show the dependency which leads to reduction of execution time in the purposed structure in comparison to the execution time when these instructions are executed separately.

3.3. The reasons for using XML

XML is a pervasive and universal format which is used to show the structural information extensively. This document naturally has a hierarchical structure suitable for representing the program code structure. The XML-based representation is so easy to understand and manipulate by simple tools, because it is so flexible and is supported extensively.

The advantages of a representation which is based on XML for program code [23] are:

- Explicit code structure: The XML is naturally structured and can be used in tree-shape model for representation.
- Powerful Querying capability.
- Extensible representation: The program code which is in the form of plain-text cannot be extended easily. Adding data and new codes in plain-text damage the code structure and entails changing in parser, while in XML new data can be inserted easily and that's because of its extensible capacity.
- Flexible formatting.
- Cross referencing
- Extensive support

Since we are all familiar with XML and its uses, we avoid describing its structure and the related experiences in using this structure will be discussed in section V.

4. Defining the components of purposed representation

Statements: for back-annotation [26], the execution time of statement will be assigned as the statement attribute.

```
<StatementK Line=n Time=n >
   Statement
</ StatementK>
```

The input parameter: The maximum and the minimum value of parameter will be added as an element.

```
<StatementK Line=n Time=n >
    Statement
       <VarNameK Min=n Max=n>
          VariableName
      </VarNameK>
...
</ StatementK>
```

Call statements: In these statements, the execution time of a method will be varied depending on the parameters. In call statements instead of absolute execution time we should use the worst execution time.

```
<CallStatementK Line=n WTime=t>
```

```
    Statement
</CallStatementK>
```

Block: Block is a set of statements which in final calculation, the executing time of block is considered instead of the execution time of statement independently. A block is considered as an element in which the timing information will be used as its attribute. Also, the approaches that applied IPET calculation use number of iteration of a block. therefore, for each block, number of iteration of block will be assigned as the statement attribute.

```
<BlockN … NoOfItr= n>
 <StatementK …  >
      …
    </StatementK>
 …
</BlockN>
```

According to the above definition various blocks will be made:

1. The simple block:

In a simple block there are some statements with specified execution time. The simple block execution time is obtained by adding up the execution time of statement in each block.

```
<SimpleBlockN  TotalTime=n … >
    <StatementK …  >
      …
    </StatementK>
 …
</SimpleBlockN >
```

2. Loop Block: for context-sensitive loop, the WCET of loop for each execution of loop may be different. Therefore, for each execution of loop, information is an element for the loop and includes the followings:

- Worst execution time for an iteration
- The maximum number of iterations

```
<loopBlockK WExeTime=n MaxItr=n..>
    <ExectionK   WExeTimePIt=n  MaxItrPIt=n >
         Loop Call Statements
    </ExectionK>
  …
</LoopBlockK>
```

3. The conditional statement block: each of these kinds of blocks consists of two blocks: one related to then-part and the other to else-part. The execution time of each of these blocks is expressed as its attribute. The total execution time of the conditional statement is considered as an attribute for if block element and is in the forms of worst, best, and total.

```
<IfBlockK BTime=n  WTime=n TotalTime=n … >
  <ThenBlockK Time = n … >
     …
  </ThenBlockK>
```

```
  <ElseBlock Time=n >
    …
  </ElseBlockK>
</IfBlockK
```

4. Switch statements block: In these blocks each case has its own execution time which is expressed as an attribute for the element of case block. The total execution time of the conditional statement is considered as an attribute for Switch block element and is in the forms of worst, best, and total.

```
<SwitchBlockK  BTime=n WTime=n  TotalTime=n>
 <CaseBlockK Time=n>
   …
 </CaseBlockK>
 …
</SwitchBlockK>
```

5. The block of statements affected by the low-level factors: As stated above, factors like modern processors made the simultaneous and fast execution of many statements. These statements are adjacent each other. Therefore by specifying them one can place them in a single block (HLETA). The time of this block will be considered instead of the sum of the execution time of statements.

```
<HLETABlockK Time = n NoOfItr= m >
 <StatementK …>
 …
</HLETABlockK
```

The parameters of methods: At the start of each method, its timing information should be specified. Therefore, a block has the following information.

- The capability of encoding parameters and their values in the form of maximum and minimum.
- The total execution time for the method

```
<MethodTimeInfoBlockK  TotalTime=n>
   <ParameterBlockK  NumberOfParameters=n>
       <ParamK Min= n Max=n >
          ParameterName
       </ParamK>
         …
   </ParameterBlockK>
</MethodTimeInfoBlockK>
```

This parameter could be used for parametric WCET analysis[27][28].

Of course in recursive call the execution time of the method is calculated based on its total execution time and the depth of recursion. Therefore, the information block of these methods needs to be different from that of other methods.

```
<RecMethodTimeInfoBlockK TimePCall=n
               CallDepth=n TotalTime=n >
<ParameterBlockK  Parameters=n>
  <ParamK Min= n Max=n >
       ParameterName
```

```
    </ParamK>
        …
</ParameterBlockK>
</RecMethodTimeInfoBlockK>
```

Infeasible execution paths: For each path with an infeasible path, an attribute is added to each block or element on the path which shows its infeasibility.

```
<BlockN Infeasible Path=True>
 …
</BlockN>
```

5. Related works

As mentioned in Section I, the most of WCET analysis approaches first extract the information and then the calculation is performed. Using a representation of program that can preserve the timing information in itself is suitable for these approaches. [11] and [6] are examples of these approaches. Different tools and approaches have used the intermediate representation for two purposes: first, to bring about the facilities for extraction of timing information and make manual annotation of the program code second, to use the output of the tool that producing this intermediate representation as the input for another tool.

New Intermediate Code (NIC) is one of these intermediate representations. The program code with C language is translated into NIC [1]. NIC has been used as an intermediate representation in WCET analysis framework [17]. Facilitating the extraction of information, making communications between different tools and visualization of results are among the important functions of this intermediate representation.

Textual Code Description (TCD-Code) is another intermediate representation which has a structure similar to XML and has been used in some research projects. Although TCD has characteristics suitable for WCET analysis, it has also been used for inter-tool communication.

In [7] XST that is similar to XML has been used to represent syntax tree, extract information, and analyze WCET. Besides, the results are shown using an explorer that can be manipulated.

Extensible Annotation Class (XAC) is another intermediate representation which has been used for preserving information in program analysis tools [13]. XAC has two important goals: portability and extendibility to create the ability to preserve extra information needed for analysis tools. In [24] MAD, which is based on XML, has been used to express the characteristics of modern processors, statement meanings. MAD was used in [15] to represent the feature of modern processor and the semantics of instructions and etc. XTC, designed based on XML, has been used as a sharing mechanism between aiT and SymTA /s [14]. aiT produces a visible XML-based output.

6. Discussion

The proposed representation is discussed in two parts here. First, from their ability in preserving timing information and calculation of execution time is discussed. Next, the usefulness of the representation in syntactic and semantic analyses for the WCET analysis is dealt with.

The first topic in WCET analysis is analyzing program code at the level of source code (high level), object code, or machine code. XML-based representation is possible for source or intermediate level of program code representation.

Extraction of timing information from program code is done automatically or using manual annotation. As stated in [16] despite the advancements in automation of information extraction, the use

of manual annotation is unavoidable. Adding manual annotation to plain-text makes changes in the parser for syntactic analysis [23]. Moreover, the language of annotations should be used independent from tools and methodology and should have the capacity of being added at all levels of program code representation [16]. All these necessities are satisfied by the XML-based representation of the program codes. As stated in Section II, a structure should be considered for representation of the extracted timing information. Considering the proposed structure, discussed in Section IV, XML-based representation is a suitable structure for necessities mentioned in subsection A and B of Section 3. Based on the proposed representation, calculation (Section II) will be possible for all calculation methods.

The need of a common format for program code was considered by Gustafsson [9]. As stated, another purpose of this article is to clear the way for flow analysis and extraction of its information based on XML-based intermediate representation. Plain-text codes are not suitable for syntax and structural analysis [23], while WCET analysis needs syntax and semantic analysis. A XML-based representation of program code as a hierarchical structure can be useful in this regard. In what follows, the research projects which have used XML-based representation for program analysis and extraction of syntactic and semantic structures of program code are discussed. The focus of the article is on applications that can be extended for WCET analysis.

JavaML2 is one of the intermediate representations of Java program codes based on its syntax tree and is used for analyzing programs and specifying the structures used [2]. JavaML2 is an XML-based representation of Java program code. In [20] JavaML is used for reverse engineering of Java code to UML. srcML is another XML-based representation that is a representation of C++ program codes[19]. The purpose of providing this representation is to analyze program code in C++ statically. XMLizer is a tool which makes converting program codes in Pascal and Java languages into XML, based on their syntax tree [18].

Many approaches and tools of WCET analysis use graphs [21] that can be converted to XML. GLX prepares the XML-based representation for description of graphs [22].

Based on what was stated above, the proposed representation can be extended in a way that can be used for syntactic analysis and determination of the program structure.

7. Conclusion and Future Research

In this article the use of XML-based representation of program code as a standard for exchanging timing information amongst timing analysis tools is proposed. In general, the main parameters affecting a program execution time could be summarized as timing information are gathered through analysis of the program control flow of the program and the behavior of the processor on which the program is supposed to be executed. Timing analysis tools can perform on the representation and annotate the timing information resulted within the program. The annotated code could be simply applied by other tools for relatively more accurate estimation of the worst case execution times.

This representation will be developed in further research projects in a way that can do the analysis directly on the offered representation. Further advancements will be made to the representation to make encoding the architectural information therein possible and to alleviate the weakness in storing the time information in blocks related to structures affected by low-level factors.

8. References

[1] A Retargetable Compiler for ANSI C website. URL: http://www.cs.princeton.edu/software/lcc/
[2] Aguiar A., David G., Badros G., “JavaML 2.0:Enriching the Markup Language for Java Source Code”, XML: Aplicacoes e Tecnologias Associadas (XATA 2004), Porto, Portugal, 2004.

[3] Bernat G., Colin A., Petters S. M., "pWCET: a Tool for Probabilistic Worst-Case Execution Time Analysis of Real-Time Systems", Technical Report YCS-2003-353 , Department of Computer Science, University of York, UK, February 2003.
[4] Chen K., Malik S., August D., "Retargetable static timing analysis for embedded software", Proceedings of the 14th international symposium on Systems synthesis, Canada, 2001.

[5] Ermedahl A., "A Modular Tool Architecture for Worst-Case Execution Time Analysis", PhD dissertation, Dept. of Information Technology, Uppsala Univ. Uppsala, Sweden, 2003.
[6] Ferdinand C., Heckmann R., Langenbach M., Martin F., Schmidt M., Theiling H., Thesing S., Wilhelm R., "Reliable and precise WCET determination for a real-life processor". In Proceedings of the First International Workshop on Embedded Software, LNCS 2211, p469{485. Springer, 2001.
[7] Gustafsson J., "Analysing Execution-Time of Object-Oriented Programs using Abstract Interpretation", PhD thesis, Uppsala University, Uppsala, Sweden, May 2000.
[8] Gustafsson J., Ermedahl A., Lisper B., "Towards a flow analysis for embedded system C programs", In Proc. 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2005), February 2005.
[9] Gustafsson J., Ermedahl A., Lisper B., Sandberg C., Källberg L., "ALF - A Language for WCET Flow Analysis" Proc. Ninth workshop on Worst-Case Execution Time Analysis (WCET'09), Dublin, Ireland, June 2009.
[10] Healy C. , Sjodin M., Whalley D. B., "Bounding Loop Iterations for Timing Analysis", In Proc. IEEE Real-Time Technology and Aplications Symposium, pages 12–21,1998.
[11] Healy C., Sjodin M., Rustagi V., Whalley D., Engelen R., "Supporting timing analysis by automatic bounding of loop iterations", Real-Time Systems, 18(2/3):121{148, May 2000.
[12] Holt R., Winter A., Schürr A., "GXL: Towards a Standard Exchange Format", Working Conference on Reverse Engineering, 2000.
[13] Hu E., Bernat G., Wellings A., "A Static Timing Analysis Environment Using Java Architecture for Safety Critical Real-Time Systems", Proc. of the 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems WORDS-2002, p 77–84, January 2002.
[14] Kästner D., Wilhelm R., Heckmann R., Schlickling M., Pister M., Jersak M., Richter K., Ferdinand C.,"Timing Validation of Automotive Software", ISoLA 2008: 93-107, 2008.
[15] Keutzer K., Malik S., Newton A., Rabaey J., Sangiovanni-Vincentelli A.,"System Level Design: Orthogonolization of Concerns and Platform-Based Design". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19(12), December 2000.
[16] Kirner, R., Knoop, J., Prantl, A., Schordan, M. and Wenzel, I. "WCET Analysis: The Annotation Language Challenge". In Post-Workshop Proceedings of the 7th International Workshop on Worst-Case Execution Time Analysis, Pisa, Italy, July 3, 2007.
[17] Lundqvist T., "A WCET Analysis Method for Pipelined Microprocessors with Cache Memories", PhD thesis, Dept. of Computer Engineering, Chalmers University , Sweden, 2002.
[18] McArthur G., Mylopoulos J.,Ng S. "An Extensible Tool for Source Code Representation Using XML", Working Conference on Reverse Engineering, 2002.
[19] Michael L. Collard, Huzefa H. Kagdi and Jonathan I. Maletic, "An XML-based Lightweight C++ Fact Extractor" International Workshop on Program Comprehension, 2003.
[20] Russell C., Dewar R., "XML Encoded Reverse Engineering of Java to UML", Technical Report HW-MACS-TR-0007, 2003.
[21] Sandberg C., Ermedahl A., Gustafsson J., Lisper B., "Faster WCET flow analysis by program slicing", ACM SIGPLAN Notices, v.41 n.7, July 2006.
[22] Sandström P., "A look at Execution Time Analysis and Measuring Interrupt Latency", Master Thesis, Mälardalen University, 2000.
[23] Simic H., Topolnik M., "Prospects of encoding Java source code in XML" In Proc. of the 7th International Conference on Telecommunications, Zagreb, Croatia, 2003.
[24] Suhendra V., Mitra T., Roychoudhury A., Chen T., "Efficient Detection and Exploitation of Infeasible Paths for Software Timing Analysis", Design Automation Conference, 2006 43rd ACM/IEEE, 2006.
[25] Wilhelm R., Engblohm J., Ermedahl A., Holsti N., Thesing S., Whalley D., Bernat G., Ferdinand C., Heckmann R., Mitra T., Mueller F., Puaut I., Puschner P., Staschulat J., Stenström P. "The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools" in ACM Transactions on Embedded Computing Systems, Vol. 7, No. 3, Apr 2008, pages 1-53.
[26] Lokuciejewski P., Marwedel P., Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems. Springer, November 2010.
[27] Bygde S, Ermedahl A., Lisper B, An efficient algorithm for parametric WCET calculation, Journal of Systems Architecture Volume 57, Issue 6, June 2011.
[28]Marref A, Evolutionary techniques for parametric wcet analysis. In Tullio Vardanega, editor, WCET, volume 23 of OASICS, pages 103–115. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.