Worst-Case Execution Time Analysis of SCOPS Modules Based on Software Analysis Model

Jong Yong Keum[a*], Yong Suk Suh[a], Joon Ku Lee[a], Gwi Sook Jang[a], Yong Jin Seo[b], Hyeon Soo Kim[b], Je Yun Park[a]
[a]*Research Reactor Design & Engineering Div. KAERI, 150-1 Dukjin-dong, Yuseong-gu, Daejon, Korea, 305-353*
[b]*Department of Computer Science and Engineering, Chungnam Nat'l Univ., 220 Gung-dong, Yuseong-gu, Daejon, Korea, 305-764*
[*]*Corresponding author: jykeum@kaeri.re.kr*

1. Introduction

To analyze schedulability, a worst-case execution time (WCET) analysis that computes the upper bounds of the execution times for modules is indispensably required. Using WCET analysis, the longest path among various paths is searched. This paper presents analysis procedure of WCET and test and analysis results of WCET for the SCOPS (SMART COre Protection System) source modules.

2. Analysis Procedure of WCET

The analysis procedure of WCET [1] follows six steps (Fig. 1). Each step is explained in the subsequent clauses.

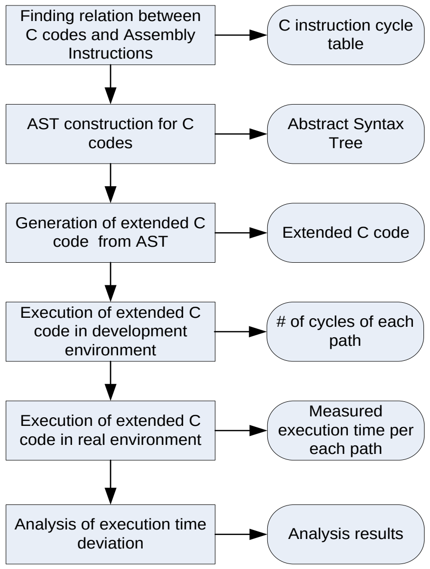
Finding relation between C codes and Assembly Instructions
C instruction cycle table
AST construction for C codes
Abstract Syntax Tree
Generation of extended C code from AST
Extended C code
Execution of extended C code in development environment
# of cycles of each path
Execution of extended C code in real environment
Measured execution time per each path
Analysis of execution time deviation
Analysis results

Fig. 1. Analysis procedure of worst case execution time

2.1 Finding relation between C codes and Assembly Instructions

C codes are transformed into assembly instructions through a compiler. Assembly instructions have operations and information to be used in an operation, and they are executed by the CPU directly. Although assembly instructions are different according to their CPU architectures, they are usually executed in the sequence shown in Fig. 2.

Using the assembly instructions corresponding to a C statement, we can calculate the cycles required to execute a C statement. Furthermore, we can compute the total cycles required to execute C modules. We performed works to classify a C statement into assembly instructions and define their execution cycles.

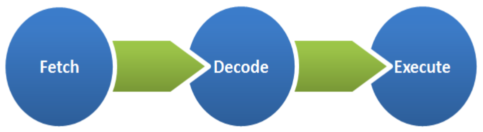

Fig. 2. Operation processes of assembly instructions

2.2 Abstract Syntax Tree (AST) Construction

The purpose for constructing AST is for extracting information to analyze a C source module. To predict the WCET, nodes of AST include operand type, and related information on an operator to analyze the C module. Most of the execution time is affected by the operand types. For example, in the case of + operator (ADD), it takes one cycle to compute an add operator with both operands of integer type. But it takes four cycles to compute an add operator with both operands of a float type. Using the operand information, we can determine the cycle counts of operators.

Nodes of AST include abstract expressions for operation cycles. For example, suppose an expression is $a>b$ and a, b are integers. A cycle string, such as "LOAD + LOAD + int_GT" is stored in a specific node field of AST. A cycle string is used to store information to know the cycle counts of any operation. LOAD means the time required to load any variable into a register from memory. And int_GT means the time required to compare integer a and integer b loaded into registers.

2.3 Generation of Extended C Code from AST

Each node of AST includes a cycle string and type information for an operation to predict WCET. Additionally, because nodes include C source codes, each node is traversed sequentially and node information is printed. Finally, we can print an extended C module and find WCET.

There are various execution paths in a C module. To find execution paths in the module, probes should be inserted in an appropriate position of AST. Using these probes, we can know which paths are executed in a module.

2.4 Execution of Extended C Code in Development Environment

Extended C codes include codes to measure the execution time and determine the execution paths. It is possible to predict the execution time of a specific path of extended C codes with test cases. We must execute a specific function to measure WCET per function and observe the results. In particular, to control the execution paths, we must insert extended probes into the C codes, observe an execution path, and prepare test cases to execute different paths in a module.

The test environment (Fig. 3) in a simulation mode was constructed to measure WCET of extended C module. A test driver is needed to call a specific module of SCOPS codes. This test driver includes codes to prepare test data and codes to call the related module.

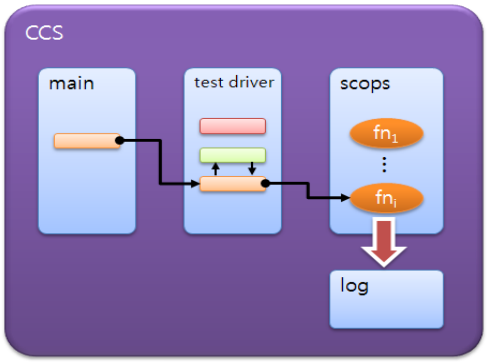

Fig. 3. Test environment to measure WCET in a simulator

In Fig. 3, when related modules are executed with a test data, information to indicate whether a *true* path or a *false* path of a predicate in the module passed and execution cycle counts of the path are recorded in the log.

2.5 Execution of Extended C Code in a Real Environment

In the 2.4 clause, WCET of the extended C modules in a simulator was measured. But this WCET is only a predicted value. The execution time of the C modules in a real test environment is measured as in Fig.4. This environment is similar to Fig. 3 except that the C modules are executed on a real processor board, not in a simulator, and the execution time is measured using an oscilloscope.

2.6 Analysis of Execution Time Deviation

WCET to be measured using a software analysis model is the predicted value of the execution time. To demonstrate the validity of this predicted value, we must show its deviation from the real execution time. If such validity is demonstrated, we will be able to use a software analysis model to analyze the deterministic features and response time of the software.

To find the deviation of the software analysis model from real execution time, Equation 1 is used. In Equation 1, t(Software) means the predicted value of the software analysis model and t(Board) means the real value from the target processor board.

$$\text{deviation} = \frac{\text{t(Software)} - \text{t(Board)}}{\text{t(Board)}} \times 100 \quad \text{(Equation 1)}$$

Applying t(Software) and t(Board) to Equation 1, we obtain 75.9% on average. Through the deviation analysis, we showed t(Software) is the upper bound of t(Board). Therefore, when a real program is executed on a target processor board, the real execution time is lower than the execution time from the software analysis model.

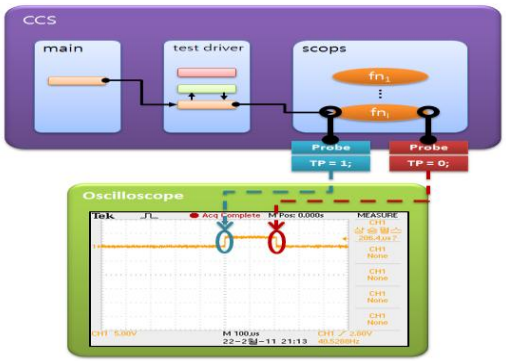

Fig. 4. Test environment to measure WCET on a real processor board

3. Conclusions

The test cases in this paper are only for a unit test and do not sufficiently reflects the system context. In other words, test cases are those that meet the branch coverage. Therefore, we cannot assure that these execution paths searched using these test cases are real execution paths searched in view of system context. To precisely measure WCET, we need a set of test cases to reflect the real system context. We will also find the execution paths of a real system and methods considering pipelining, cache, and so on.

REFERENCES

[1] Korea Atomic Energy Research Institute, "Final Report on Real Time Analysis of SCOPS test software," October 2011