Partial Flow Analysis with oRange

Marianne de Michiel, Armelle Bonenfant, Clément Ballabriga and Hugues Cassé*

Université de Toulouse, Institut de Recherche en Informatique de Toulouse (IRIT)

Abstract. In order to ensure that timing constrains are met for a Real-Time Systems, a bound of the Worst-Case Execution Time (WCET) of each part of the system must be known. Current WCET computation methods are applied on whole programs which means that all the source code should be available. However, more and more, embedded software uses COTS (Components ...), often afforded only as a binary code. Partialisation is a way to solve this problem.
In general, static WCET analysis uses upper bound on the number of loop iterations. oRange is our method and its associated tool which provide mainly loop bound values or equations and other flow facts information. In this article, we present how we can do partial flow analysis with oRange in order to obtain component partial results. These partial results can be used, in order to compute the flow analysis in the context of a full application. Additionally, we show that the partial analysis enables us to reduce the analysis time while introducing very little pessimism.

1 Introduction

Critical hard real-time systems are composed of tasks which must imperatively finish before their deadline. Static WCET analysis is performed by a timing analysis tool which needs loops upper bounds. Such bounds may be given by manual annotations of programs or automatic evaluation. All feasible paths through the program have to be studied in order to extract some flow information which is used to statically bound the number of times loops are iterated.

Several approaches have been proposed in Flow Analysis about loop bounds [2,3,7,8,9,10,11,13].

The current WCET computation methods are designed to be used on a whole program. However, there are some drawbacks to this approach. First, the analyses used for WCET computation usually run in exponential time with respect to the program size. Second, when the program to analyze depends on external components (e.g. Components Off The Shelf (COTS) or libraries) whose sources are not available, the lack of information about the components prevents the WCET computation if information from the user is requested (and not provided). Partial analysis becomes then crucial to improve the time needed to compute WCET and more important the computation of the WCET itself.

* michiel,bonenfant,ballabri,casse@irit.fr

This article presents how partial analysis is possible with oRange, our tool which calculates upper bounds of loops in C programs. In section 2, we present the partial analysis . Section 3 compares classical and partial analysis provided by oRange. Finally, Section 4 gives our conclusions.

2 Partial Analysis

oRange[12] combines a) loop bound expression building of C program as in [7] with b) abstract interpretation as in many previous works [9,10,11,13]. It is used by OTAWA[1][1] in order to obtain loop bounds expressions *total* and *maxi*. For a loop L_i, $total_i$ represents the total number of iterations in the overall execution of the program and $maxi_i$ represents the maximum number of iterations for each loop startup.

2.1 Description

A partial result: When partialising a function, we build what we call a *partial result*. It is a pair made of a tree representing loop and function calls included in the partialised function and an abstract store[2] which contains the assignments of the non local variables (global, static variables and parameters being modified by the function). The tree is a parametric flow fact, it is instantiated with the call context in the full analysis. The abstract store is used to evaluate the impact of the function on the rest of the application. These two results are called summaries of the function.

Usage of a partial result: A partial result can be used to build either a partial result of another function, or a complete analysis. The pair of the partial result is combined with the caller context: the abstract store representing the caller context is instantiated with the tree and composed with the abstract store of the partial result.

Partial analysis and pessimism: The number of times we cannot determine if a branch is executed or not is higher in partial analysis because we know less of the context. Thus, we have to take into account the two possible branches and usually add indeterministic results. This leads to pessimism, but is only effective if the loop iteration number depends on at least a variable in an alternative branch.

2.2 Automatization

It is possible to choose manually functions to be partialised. We can also do an automatic partialisation. There are two options: a pessimistic option which

[1] oRange is integrated in the OTAWA tool chain dedicated to WCET computation.
[2] A set which maps each variable of the execution state to an expression which can depend on the input of the instruction (i.e. preceding instructions)

improves time computation by choosing to partialise large functions. The second option partialises only non-pessimistic functions. In order to automatize, we first select functions regarding the options. Potentially partialised functions are then classified according to their nesting level.

We define an internal weight which depends on the number of internal function calls, the weight of the function called, the number of assignments in loops, the assignments of in and out parameters. We then obtain a total weight depending on the imbrication level, the frequency of call of the function, the number of nested loops in the function... During partialisation, the total weight of a function can be re-evaluated.

Depending on its internal, total weight and the option chosen, functions are potentially partialisable.

Levels are determined by the imbrication level of inside function calls where functions are either partialisable themselves or not.

3 Results

In table 1 we present a comparison between classical full and partial analysis on the debie application (WCET'07 challenge[5]) and its functions/sub functions. Experiment has been done on a Core2 Duo Processor T7200 2GHz.

			Classical			Partial		
Program	L_C	L_N	% of B	time	C	% of B	time	partialisation
class	2	2	100%	15.885s	6	100%	14.529s	none
hw_if	3	3	66%	16.037s	16	66%	14.481s	none
measure	12	6	25%	26.394s	124	25%	25.062s	none
tc_hand	9	4	78%	6m42.573s	127	78%	1mn24.861s	2(2 levels)
harness	2379	43		>11h	50171	87%	46mn50.068s	55 (8 levels)
telem	4	4	50%	15.981s	9	50%	14.677s	none
health	85	11	86%	92m12.706s	1344	86%	28.014s	11(8 levels)
debie	2390	1		>11h	50269	88%	47mn3.728s	56 (11 levels)

Table 1. Classical versus Partial Analysis

L_C: number of loop calls from any file
L_N: number of loops directly into the file
B: number of loops bounded
$\%B$: % of B according to L_C
C: number of function calls

Thanks to partialisation, there is a real gain of time computation. Fortunately, the number of loops bounded is identical or greater.. In term of pessimism, tc_hand and health could have obtain less accurate loop bounds because of partialisation, but in these cases loop bounds do not depend on alternative branches for these cases (see 2.1). In most cases, we think partialisation introduces pessimism.

4 Conclusion

oRange, which computes flow facts information (especially loop bounds), can be used to obtain partial results of functions and can use them. The main achievement of our approach is that partial evaluation may be processed for a function independently of any call, partial results are then combined with each call context in a complete evaluation. Experimental results show that the computing time to get the flow facts can be greatly improved. oRange automatic analysis supports options allowing pessimism reduction.

One of the main usage of partialisation is to obtain COTS summaries and to be able to use them without analyzing the entire library as we have done in [6]. This study has been partially funded by the European Community under the Merasa [4] project.

References

1. Otawa. http://www.otawa.fr.
2. Bound-t tool. http://www.tidorum.fi/bound-t/, 2005.
3. ait tool. http://www.absint.com, 2007.
4. Merasa. http://ginkgo.informatik.uni-augsburg.de/merasa-web/, 2007.
5. Wcet project. http://www.mrtc.mdh.se/projects/wcet/, 2007.
6. Clément Ballabriga, Hugues Cassé, and Marianne De Michiel. A generic framework for blackbox components in wcet computation. In *9th Intl. Workshop on Worst-Case Execution Time Analysis, WCET 2009, Dublin, Ireland*, 2009.
7. Joel Coffman, Christopher A. Healy, Frank Mueller, and David B. Whalley. Generalizing parametric timing analysis. In Santosh Pande and Zhiyuan Li, editors, *LCTES*, pages 152–154. ACM, 2007.
8. Christoph Cullmann and Florian Martin. Data-flow based detection of loop bounds. In *7th intl. workshop on worst-case execution time (wcet) analysis, pisa, italy*, 2007.
9. Andreas Ermedahl, Christer Sandberg, Jan Gustafsson, Stefan Bygde, and Björn Lisper. Loop bound analysis based on a combination of program slicing, abstract interpretation, and invariant analysis. In *7th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis, Pisa, Italy*, 2007.
10. Martin Kirner. Automatic loop bound analysis of programs written in c. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2006.
11. Paul Lokuciejewski, Daniel Cordes, Heiko Falk, and Peter Marwedel. A fast and precise static loop analysis based on abstract interpretation, program slicing and polytope models. In *Cgo '09: proceedings of the 7th international symposium on code generation and optimization*, Washington, DC, USA, 2009.
12. Marianne De Michiel, Armelle Bonenfant, Hugues Cassé, and Pascal Sainrat. Static loop bound analysis of c programs based on flow analysis and abstract interpretation. In *RTCSA*, pages 161–166. IEEE Computer Society, 2008.
13. Adrian Prantl, Jens Knoop, Raimund Kirner, Albrecht Kadlec, and Markus Schordan. From trusted annotations to verified knowledge. In Niklas Holsti, editor, *WCET*, volume 09004 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.