Parametric WCET Analysis

Stefan Bygde

March 16, 2007

Contents

1 Research Area 3

2 Research Methodology 3

3 Research Overview 3
   3.1 Flow Analysis ........................................... 4
   3.2 Low-level Analysis ...................................... 4
   3.3 Calculation Phase ....................................... 5
   3.4 Relation to my research ................................. 5

4 Conferences 6

5 Literature 7

6 Research Groups 8

7 Research 9
   7.1 Detailed description ..................................... 9
      7.1.1 The Polyhedral domain ............................. 9
   7.2 Hypothesis .................................................. 10
   7.3 Research Questions ....................................... 10

8 Activities and Expected results 11
   8.1 Improving the analysis of SWEET ....................... 11
   8.2 First Prototype ........................................... 12
   8.3 Implementation into SWEET .............................. 12
9 Time Plan

9.1 Milestones
1 Research Area

My research area is parametric Worst Case Execution Time (WCET) analysis. WCET analysis is about getting information about the worst-case execution time for a program on a certain platform for any possible input. Static WCET analysis is an automatic way to derive a safe upper bound for the worst-case execution time of a program by analyzing its source code and the hardware of the target platform. In many cases it is useful to have such an estimation of the WCET, especially for hard real-time systems. Normally, the WCET is estimated by measurements of the executing program (dynamic analysis), which gives an underestimation of the actual WCET (because it is not possible to be sure that the worst case has been measured). In static analysis on the other hand, the estimation of the WCET should always be equal or greater than the actual WCET (of course, assumed that the analysis tool is correct). In classic static WCET analysis, the upper bound of the execution time is given as an absolute value (for instance, in milliseconds) whereas parametric WCET-analysis aims to derive the WCET-bound as an expression of some undetermined variables, which later can be instantiated. The techniques used for this is quite general static analysis techniques and could possibly be used in other applications than WCET analysis.

2 Research Methodology

The first step to examine this area is to implement a fast prototype to find out how well some suggested theoretical ideas of parametric WCET analysis would work out in practice. The prototype will hopefully reveal the important issues and problems that have to be solved. These issues and problems should then be carefully examined and some theory and/or techniques should be developed. Finally, if the methods/ideas developed seem to work, the plan is to do a more serious implementation into an existing research-prototype developed at MdH.

3 Research Overview

There are in general two ways of estimating the WCET of a program; dynamic and static analysis. Dynamic analysis is done by measurements on the running software, there are many techniques and tools for doing that.
Measurements take a lot of effort to do (setting up the gear and so forth). By dynamic analysis one typically obtains an under-estimation of the "real" WCET, because it is hard (not to say impossible) to force the program to its worst case behaviour. On the other hand, we have static analysis which is an automatic process that derives a safe upper bound of the actual WCET. By a safe upper bound we mean an estimation that is guaranteed to be at least as high as the real WCET. However, the estimation should be as tight as possible, meaning that we aim to get as close as possible to the real WCET without compromising the safety. To obtain such an estimation one has to take both software and hardware into consideration. Usually this analysis is divided into different phases called flow analysis, low-level analysis and calculation.

3.1 Flow Analysis

The most important impact on the execution time of a program is of course the software. Execution typically spends most time in loops and finding (safe) upper bounds of the execution bounds of loops is essential for obtaining a good estimation of the WCET. The flow analysis phase analyses the source code (or possibly some intermediate format or even the object code), to find bounds on the execution counts for all program points. This is non-trivial, in fact, finding exact flow-information is in general uncomputable, so one typically has to introduce some abstraction to make it computable and in that of course loose some precision. Research problems on this level is mainly static analysis techniques such as slicing (removing variables and parts of programs which doesn't affect the program flow), abstract interpretation (general framework for abstracting program semantics) etc.

3.2 Low-level Analysis

Finding the flow facts of a program is not enough to derive a concrete WCET estimation, programs can behave differently on different platforms, especially on modern platforms where you typically find cache memories, pipelines, branch prediction etc. These construct improves the average performance but at the same time it introduces unpredictability and it can in some cases yield very complex worst-case behaviours. The low-level analysis phase tries to find the worst-case behaviour for the atomic instructions of the hardware using a timing model. The timing model is a abstract model of the hardware.
(including caches, pipelines etc.) which focuses on timing aspects. The model has always to assume the worst possible case (e.g. assuming a cache-miss if it cannot without doubt infer a cache hit) in order to be safe. The output of the Low-level analysis is worst case behaviour of the atomic parts of a program. Research on this level includes modeling of hardware, predictability and so forth.

3.3 Calculation Phase

The calculation phase combines the results from the flow analysis and the low-level analysis to derive a concrete WCET estimation. There are different techniques to achieve this. A simple example is the Implicit Path Enumeration Technique (IPET) which is simply to take the execution bounds derived in the flow analysis, let’s call them $c_i$ where $i$ is a program point, and the worst-case execution time of atomic parts of the program $x_i$ and then maximize the sum

$$\sum_{i \in P} p_i * x_i$$

subject to some constraints obtained from the flow analysis. This sum will surely include the worst-case path through the program, however, while the path is certainly possible to take structurally, it might be impossible to take due to semantic constraints which is not taken into consideration by this method. Thus the sum may correspond to a path through the program which is not actually a valid path (i.e. it will not occur in practise) but it will certainly be at least as long as the real worst-case path and therefore we will obtain a safe over-estimation of the actual WCET. To tighten the estimation the flow analysis can provide some info on infeasible paths which is paths through the program which is structurally possible to take but impossible to take in practise.

3.4 Relation to my research

My research will focus on the flow-analysis part, but will also include the calculation phase. As mentioned above, yielding exact information about a program is in general uncomputable so a framework for abstracting the semantics of programs was proposed in the seventies by Cosout & Cosout [CC77]. Using abstract interpretation and a parametric IPET technique, my aim is to try and obtain the WCET estimation as a expression in some
unknowns rather than as a constant value. The unknowns will in the first attempt correspond to program (input) variables, and the WCET estimate will thus be given in terms of them. The technique will mainly rely on different techniques which has been used in other applications.

4 Conferences

This project is somewhat overlapping with different communities. While the context of the project is to obtain a parametric WCET formula for a piece of code in embedded real-time systems, the techniques and research is quite general and could be used in many other applications as well. Therefore conferences in static analysis in general are the most interesting conferences and those of embedded systems and real-time system are of less interest.

SAS

One of the most important conferences in program analysis in general is the Static Analysis Symposium (SAS), topics in this conference is limited to static analysis including topics as: abstract interpretation, control flow analysis, model checking etc. In this conference there are mainly theoretical and general approaches presented.

TACAS

Another conference related to WCET analysis is the Tools and Algorithms for the Construction and Analysis of Systems (TACAS) which is more focused (as the name suggests) on tools and algorithms. The conference is quite general and is meant to find similarities between different communities. Topics include but are not limited to for instance: formal methods, software and hardware verification, static analysis, programming languages and real-time systems.

SCAM

Another conference which focuses on static analysis which is possibly more applied than SAS is the conference on Source Code Analysis and Manipulation (SCAM) conference. Some topics: program transformation, abstract
interpretation, program slicing, source level software metrics and decompila-
tion.

**WCET**

There is also a very focused international workshop on WCET under the Euromicro Conference on Real- Time Systems (ECRTS) conference. The topics in this conference is limited to timing analysis and accepts submissions that deals with, for example: Flow analysis for WCET, Low-level timing analysis, modelling and analysis of processor features, Calculation methods for WCET, Strategies to reduce the complexity of WCET analysis, Timing analysis through measurement, Probabilistic analysis techniques and others.

**5 Literature**

**Abstract Interpretation**

The topic of static analysis, abstract interpretation and WCET analysis is quite big, there are several conferences with related material and so forth. This section will be focusing on central literature in my particular work. Since my work will be centralized around abstract interpretation, there is one important seminal paper, that is the introduction paper by Cousots in 1977. It has the not-so-short title ”abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints” [CC77]. This has generally been accepted as a subject and as could be seen above; many conferences has this as a topic. One important topic in abstract interpretation is choosing a suitable abstraction for the semantics, such an abstraction is commonly referred to as an *abstract domain*. Important such domains has been proposed in [CC77], [CH78], [Gra89].

**Program Analysis**

When it comes to static program analysis in general there is a very important book which attempts to unify most static analyses and showing similarities between them. It is an essential source of knowledge for beginners and researchers in static analysis. The title is ”Principles of Program Analysis” by Nielson, Nielson and Hankin [NNH05].
Parametric WCET Analysis

There has not to our knowledge been much research on this specific subject. My research will be heavily based on Björn Lispers paper "Fully automatic, Parametric Worst-Case Execution Time Analysis" [Lis03]. The ideas in this paper will be further investigated, implemented and evaluated. Inspired by Lispers paper are two master theses developed at Saarland University [Hum06][Alt06]. These theses used many ideas from [Lis03] but used a little bit different approach. This literature will be very central to this work.

6 Research Groups

There are some research groups that actively works with static WCET analysis.

The compiler design lab of Saarland university

Parts of this lab has timing research as interest and some interesting publications has been written by this group.

Patrick Cousots group on abstract interpretation

Patrick Cousot, one of the founders of abstract interpretation is still active in the research of abstract interpretation. His research group has a very informative web page with alot of pointers to publications involving abstract interpretation.

AbsInt

This is a company rather than a research group. As the name of their title hints, they do static analysis based on abstract interpretation. They have developed a tool called aiT, which is a static analyzer. The tool focuses on hardware timing and performs abstract interpretation over the states of caches and pipeline.
7 Research

7.1 Detailed description

Abstract Interpretation is a framework for program analysis. It is a general theory for abstracting program language semantics. We define the collecting semantics of a program as a semantics that captures all possible states that may occur during any execution of a certain program. Clearly, this semantics is uncomputable since it would have solved the halting problem if it computable. By using abstract interpretation we abstract the semantics to make it computable, of course we will loose precision by doing this so we have to decide on which properties we are interested in. By the collecting semantics we would have derived sets of states at each program point, in abstract interpretation we instead derive an abstract state for each program point. The abstract state describes some invariant of the program variables at a program point. An abstraction like this is called an abstract domain. Examples of abstract domains are the following:

\[ x \in \{ v | a \leq v \leq b \} \text{ for some } a, b \in \mathbb{Z} \]  
\[ x - c \equiv 0 \pmod{m} \text{ for some } c, m \in \mathbb{N} \]  
\[ \sum_{x_i \in \text{Vars}} c_i x_i \leq m \text{ for some } c \in \mathbb{Z}^{\text{Vars}} \]  

where Vars denotes the set of variables in a program and \( x \in \text{Vars} \). Equation 3 is will be an important domain in my research, we will therefore describe it in more detail.

7.1.1 The Polyhedral domain

This domain was introduced in [CH78] by Cousot and Halbwach. It abstracts states of integer-valued variables by enclosing the possible values inside a convex polyhedron in an \( n \)-dimensional space where \( n \) is the number of program variables. The possible states correspond to integer-points inside a certain convex polyhedron (the polyhedron is derived by the abstract semantics). If \( x \) and \( y \) are program variables, we can have an abstract state look as in Figure 1. In a deterministic and terminating program we can derive an upper bound for the execution count of a certain program point by counting the integer points inside the polyhedron. There are known techniques for doing this efficiently. The reason is that in a terminating, deterministic program,
every time a program point is visited during run-time it will be visited by a new state (otherwise it would never terminate), so we can be sure that we have bounded the execution count by counting this over estimation of possible states.

7.2 Hypothesis

"Parametric WCET Analysis gives more information about a program than ordinary static WCET analysis and can potentially give tighter estimations"

7.3 Research Questions

Some concrete questions that has to be answered is the following:

Complexity

Abstract interpretation using the polyhedral domain has a exponential complexity in the number of variables considered. A reasonable question is: is
it feasible to use these techniques in practise? Do the worst-cases actually appear in this context? If infeasible, can we simplify the techniques or use other techniques? There are alot of abstract domains, and switching to a simpler (not as accurate) domain we could reduce complexity to the cost of precision. Another solution could be to use aggresive slicing algorithms to reduce the number of variables to take under consideration.

Linear Restrictions
Several of the techniques, including the polyhedral domain, consider linear relationships between variables. However, in a program there might be much more complex relationships than linear ones. The question is: how much precision do we loose by only considering linear relationships? There are other domains that can handle other, more complex relationships. Can we add more domains without increasing the complexity too much (see previous point).

Low-level code
The analysis is supposed to work on an intermediate or binary level, not just sourcecode level. There are several problems involved with this. For instance, integers are represented by bit-strings which can be interpreted in a non deterministic way (i.e. as signed or unsigned). The techniques used above does not provide a solution on how to deal with low-level code and we have to provide a solution.

8 Activities and Expected results

8.1 Improving the analysis of SWEET
First part of the research does not directly involve the previously mentioned parts but is still in the area of abstract interpretation. The analyses in SWEET relies on the abstract interval domain (see equation 1) which in many cases suffices for deriving a good WCET bound. As master thesis I added the congruence domain (equation 2) to SWEET, with the intention to make the analysis results more precise. First part of my research is to use the interval domain and the congruence domain together and use them together to refine the result maximal, in terms of abstract interpretation this
is to say that we use the reduced product of the two domains. To be able to implement the congruence domain we had to add support for low-level code and bit-operations. We will submit the first paper to SAS in 2007. If all turns out well, we will present the paper in Denmark in early fall 2007. Expected results: New theory concerning congruence analysis on low-level.

8.2 First Prototype

The second phase of the research is to implement a first quick prototype of the ideas mentioned in Section 7.1. For most of the techniques we will be using there are free libraries. Thus, the implementation of the prototype will not require too much effort. After the prototype have been implemented, the issues that has to be further investigated will become obvious. In this phase we expect to get answers to the research questions we stated in Section 7.3. They will be answered by investigations, examples. From this we will try and develop consistent theory on how to develop these kind of analyses. Expected results: A more clear view on the problem, a platform for testing ideas, theory and methods for abstract interpretation on a low level.

8.3 Implementation into SWEET

If we can solve the stated problems and the analysis seems to work fine, we will investigate the possibilities of having the analysis implemented into SWEET. That is if SWEET easily can be modified to support this functionality. SWEET currently has a very low-level memory representation and it is not always clear on what should be interpreted as a variable. We consider this to be a general problem and we will try and find a solution for it if at all possible. Expected results: An implementation into SWEET.

9 Time Plan

9.1 Milestones

2007

- Submit paper about the congruence domain
- Start implementing prototype
• Take 10-15 credits in courses

2008
• Licentiate Proposal
• Finish prototype implementation
• Start evaluating issues of approach
• Take 15-20 credits of courses
• At least one publication

2009
• Licentiate Defence
• Start implementation into SWEET
• Take 10-15 credits of courses
• At least one publication

2010
• PhD Proposal
• Finish SWEET implementation
• Take 10-15 credits of courses
• At least one publication

2011
• PhD Defence
References


