Static Energy Consumption Analysis in Variability Systems

Marco Couto*, Jácome Cunha‡, João Paulo Fernandes§, Rui Pereira*, and João Saraiva*
* INESC TEC/HASLab, Universidade do Minho, Portugal
‡ NOVA LINCS, DI, FCT, Universidade NOVA de Lisboa, Portugal
§ LISP - RELEASE, Universidade da Beira Interior, Portugal
marco.l.couto@inesctec.pt, {ruipereira,jas}@di.uminho.pt, jacome@fct.unl.pt, jpf@di.ubi.pt

***Abstract*—Energy consumption is becoming an evident concern to software developers. This is even more notorious due to the propagation of mobile devices. Such propagation of devices is also influencing software development: a software system is now developed has a set of similar products sharing common features.**

In this short paper, we describe our methodology aim at static and accurately predict the energy consumption of software products in such variability systems, typically called software product lines.

I. Introduction

Software development has drastically changed from the last to this new century. Software is now present in a wide variety of hardware systems, ranging from different (mobile) phones, portable computers, and consumer electronics. As a consequence, software developers have to structure their software so that it can be easily be ported to devices with different characteristics.

The way we use software systems has also changed: large wired computer main frames and personal computers, are not the main and only hardware running our software. Nowadays, we often use software systems in mobile devices, where time to execute our software is not the only concern: energy consumption, and as consequence the battery drain, is becoming an important concern in software engineering [1].

In this paper we present a methodology to reason about energy consumption in the context of software where there are many variants of a similar product. This context is nicely captured by the very active research area on Software Product Lines (SPL). Software product lines are a suitable approach also to develop software for mobile devices, where applications must be developed once for many platforms with different characteristics, and where energy consumption is becoming a software bottleneck.

We present a static analysis methodology so that we can predict the energy consumption of each product in the line. As a result, the methodology is able to statically identify products with different energy consumption profiles, and, for example, to identify the greenest product in a line.

We propose to adapt well-known static analysis techniques to the SPL realm: first, we use SPL static analysis techniques [2] to compute energy related properties from the code. Second, we combine that with the Worst-Case Execution Time (WCET) prediction approach to compute the energy comsumption of a program, instead of its execution time. We call this new technique the *Worst-Case Energy Comsumption* (WCEC). Our technique aims to predict the energy consumption in a feature-sensitive manner, instead of generating all products of a SPL and analyze them individually.

II. Static Analysis in SPL

A SPL is a software system that is able of creating different software solutions from reusable assets (i.e., code fragments, visual assets, etc.). Such solutions are called *products*. A *product* is characterized by the set of *features* that it includes/implements, which has the name of *product configuration*. A *feature* is a specific functionality that can be included in one or more *products*, and is usually defined as a code block/fragment.

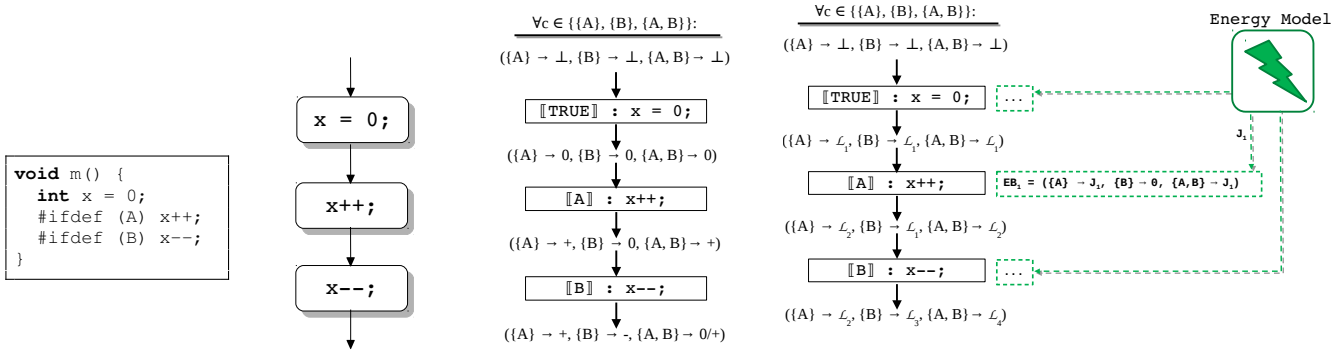
To define if a code block belongs to a specific *feature* we can use *conditional compilation*. This technique is based on associating to the code block a pre-processor instruction, an `#ifdef` Φ, where Φ is a propositional logic formula over *feature names*. The syntax of a formula is as follows:

$$\Phi ::= f \in \mathbb{F} | \neg\Phi | \Phi \wedge \Phi | \Phi \vee \Phi$$

Here, f is a *feature* name, drawn from a finite alphabet of feature names $\mathbb{F}$. This allows to indicate which *features* include that code block, or which one must exclude it.

In order to statically analyze a program (or a set of programs - a SPL) we use techniques that rely on the analysis of the control/data flow graph of such program(s). Every static dataflow analysis needs to combine essentially three components: a *CFG - control flow graph* (to represent the connection between instructions), a *lattice* (to represent the values of interest for the analysis), and the *transfer functions* (responsible for simulating the execution of the program represented by the *CFG*). For SPL analysis, we need to make the analysis feature aware, so it can compute the results for all products at once. Following the approach presented in [2], this can be achieved by extending the three static analysis components. In other words, to each *CFG* node we associate the list of *features* that implement the corresponding instruction and the *lattice* element calculated by the *transfer functions* for each possible product.

Figure 1 represents a SPL example, with two *features* ($\mathcal{A}$ and $\mathcal{B}$), with the source code (Figure 1a), the basic CFG for a *product* with the two *features* (Figure 1b), and the SPL



(a) Example of a SPL (b) CFG (c) CFG for SPL Static Analysis (d) Calculating local energy bounds for each node in the CFG

Fig. 1: SPL example with program analysis and the energy estimation approach overview

static analysis *CFG* used for *Sign Analysis* (Figure 1c). This approach will be the basis of our work.

III. Static Energy Analysis in SPL: The idea

In order to achieve static energy consumption in SPL it is necessary not only to determine the behavior of all products, but also to understand how it can affect energy consumption. In such analysis the goal is to give an upper/lower energy bound for each instruction, determine how they are all related, and calculate an estimation for the worst/best case scenario (similar to worst/best case execution time estimation [3]).

In classic execution time prediction, in the first step, called *Processor Behavior Analysis*, the processor is analyzed in order to determine how it will behave when executing a certain statement. Moreover, it allows to determine how it will behave depending also on what statements were executed before.

Our goal for SPL static energy analysis is to create an abstract model of such behavior, with all the processor states that can affect energy consumption. This will be called our *prediction model*, $\mathbb{P}$. With the problem modeled that way, all the properties from static dataflow analysis will be maintained. This was already proven to work before [4]. For our purpose, we want to use this approach to model every hardware component that influences the energy consumption.

The *prediction model* $\mathbb{P}$, together with the *fixed point computation* of static analysis, will only return, for each node of the CFG, the hardware components states in the worst/best case scenario. In order to get an energy estimation, and following the WCET/BCET principle, we need to first match those states with an *energy model*, $\mathbb{E}$, where the consumption per state and instruction is specified, and then use a constraint solving technique to get an upper/lower bound estimation, as explained in [3].

In order to keep our approach feature-oriented we need to combine these techniques with SPL static analysis concepts. In other words, we need to calculate upper/lower energy bounds for every *product*. The approach to follow for that is the exact same one as presented in Section II: compute the information in each node for every *product* of the SPL, using the three static program analysis techniques.

The final result of this analysis allows us to predict how the components will behave after executing every statement, and for every valid *product* in the SPL. Figure 1d represents the basic overview of our approach, when analyzing Example 1a.

IV. Conclusion and Future Work

This paper presents ongoing work on the analysis of energy consumption in SPL. We have chosen to combine energy estimations with static program analysis and time estimation techniques, and the precision and correctness of our approach will obviously be dependent of the quality of our behavior analysis. In other words, the more accurate the *lattice* and *transfer functions* are, the more trustworthy the results will be. In order to prove our concept we will first develop our prediction model using only a subset of all hardware components, and represent the basic behavior of them.

Even if we can model every component into a combination of *lattice* and *transfer functions*, we still need an energy model that can be used together with it. As we said before, we assume that such a model can be provided by hardware manufacturers in the future, but for proof of concept we will design a simpler one that can be used with the simpler prediction model.

References

[1] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *Proc. of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. ACM, 2014, pp. 22–31.

[2] C. Brabrand, M. Ribeiro, T. Tolêdo, and P. Borba, "Intraprocedural dataflow analysis for software product lines," in *Proc. of the 11th Annual International Conference on Aspect-oriented Software Development*, ser. AOSD '12. ACM, 2012, pp. 13–24.

[3] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[4] A. Møller and M. I. Schwartzbach, "Static program analysis," May 2015, department of Computer Science, Aarhus University.