Source-level power estimation and optimization for embedded systems

Daniele Paolo Scarpazza
Politecnico di Milano - Dept. of Computer Engineering, Milano, Italy, <scarpazz@elet.polimi.it>

ABSTRACT

The task of estimating properties of programs, like energy consumption and execution time, was historically performed in a number of different approaches, ranging from dynamic low-level simulation to source-level analysis. Our research group has been recently working on a novel approach, which includes a static and a dynamic analysis phase, and tries to take advantage of the best features of the two approaches, thus leading to metrics which are both accurate and fast to apply. My current effort is aimed at the following objectives: developing an automatic estimation flow, capable of applying the metrics, and the extending it to the automatic selection and application of the best eligible source code transformations.

1. DETAILS

Designing embedded systems is a complex task, often subject to constraints on speed, cost, weight, volume, energy consumption and also development cost and time-to-market. In particular, battery-powered devices (mobile phones, MP3 players, PDAs, etc.) demand a special attention to energy consumption. As a consequence, designing software for such devices requires the availability of fast and accurate design space exploration strategies.

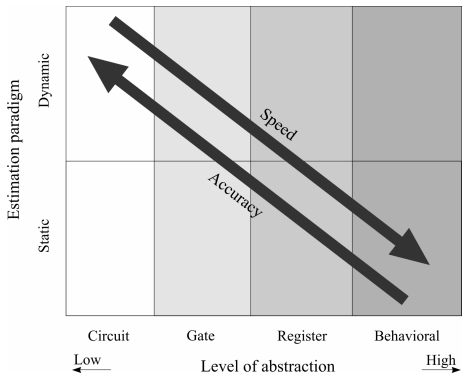

Figure 1: A simplified classification of current software power consumption estimation methods.

A number of different approaches was proposed for this analysis, ranging from gate-level to source code analyses, and relying either on a static or dynamic analysis paradigm (see fig. 1). Generally speaking, dynamic methods and methods at a low level of abstraction and proved to be more accurate than high-level and static ones, but much more computationally demanding.

The method we propose is a high-level method (since it works at the C-language source code level), and a method composed by a static phase and a dynamic phase. During the static phase, the source code is split in minimal characterizable units called *atoms*; during the dynamic phase, an instrumented version of the original source code is executed and the execution counts for each atom are recorded. This way, the efficiency of high-level static methods is combined with the accuracy of dynamic lower-level ones.

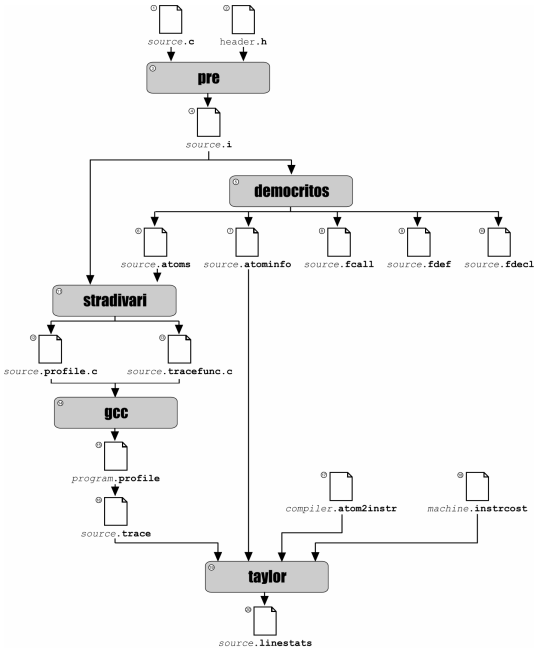

Figure 2: The current composition of our estimation flow.

The status of the project at the time I'm writing is depicted in figure 2: the original program source code is preprocessed (by `pre`, a simple script invoking `GNU cpp`) and analyzed by `democritos`, a specialized ANSI C parser we developed in `GNU bison`, able of decomposing the program in its atoms and generating a description of them. Then, thanks to this information, `stradivari` is able to generate an atom-consistent instrumented version of the original program. The instrumented program is then compiled with `GNU gcc` and run, thus yielding a complete, atom-consistent, execution trace. With the above execution trace, the atom characterization generated by `democritos`, and appropriate architectural information, `taylor` is able to provide an estimate of power consumption, execution time, binary code length, and any other additive property of the the original program. Our current task is to complete all details of `democritos`; our future objectives are the extension of the methodology to the selection and application of power-optimizing source-code transformations.