An Assembly–Level Execution–Time Model for Pipelined Architectures

ABSTRACT

TThe aim of this work is to provide an elegant and accurate static execution timing model for 32-bit microprocessor instruction sets, covering also inter–instruction effects. Such effects depend on the processor state and the pipeline behavior, and are related to the dynamic execution of assembly code. The paper proposes a mathematical model of the delays deriving from instruction dependencies and gives a statistical characterization of such timing overheads. The model has been validated on a commercial architecture, the Intel486, by means of timing analysis of a set of benchmarks, obtaining an error within 5%. This model can be seamlessly integrated with a static energy consumption model in order to obtain precise software power and energy estimations.

1. INTRODUCTION

The peculiarities of target application fields of embedded computing (e.g., mobile systems), typically pose stringent area and energy constraints. The current trend towards high–levels of integration up to system–on–chip, is exacerbating the need of taking into account power requirement during the early stages of the design as well as throughout the entire verification flow. In addition, the *penetration* of software within the typical hardware/software architectures used in embedded systems, is steadily gaining importance but unfortunately efficient power aware compiling and estimation techniques are still a research topic not yet mature for the EDA arena. Previous approaches [8][9][7] propose a characterization of the power consumption of a given microprocessor based on the measurement of the average current absorbed by the core during the execution of long sequences of the same machine instruction. Power figures are then associated with assembly instructions, leading to an abstraction from architectural details of the microprocessor. Such approaches, though, still suffer a lack of generality since a new set of measurements is needed when a different processor is analyzed. A more general approach, proposed in [1], abstracts from the architectural level by determining a set of *functionalities* and by decomposing the computational activity of each instruction in terms of these functionalities. According to this model the energy absorbed by each instruction is computed as the weighted sum of the contributions of the functionalities. A tuning phase, based on a limited set of experimental data, allows associating to each functionality an average current absorption per clock cycle. It is worth noting that the overall energy consumption is strongly dependent on the number of cycles taken for the execution of assembly instructions. In [1] the timing is assumed to coincide with the nominal value reported in the processor data–sheets. This timing data, being purely static, are a sound starting point for a general energy model but disregards the delays introduced by the interlocks arising from a pipelined execution of the code. Limitations deriving from a static analysis have been studied in [6] and a solution, based on the models proposed in [8][9], has been presented. The extended approach, though, does not address the problem of the lack of generality. The aim of this work is to cope with the above limitations, providing a model capable of describing timing overheads due to inter–instruction effects in a formal and general way. The advantages of a *static* model with respect to a dynamic, simulation–based, approach are evident. The proposed strategy is based on a dynamic characterization—to be performed once and for all—of a given instruction set aimed at producing statically usable figures. To this purpose a sound and formally consistent statistical model has been developed and verified both theoretically and by comparison against actual timing measures. The methodology and related models are being implemented in a co–design flow that will enable accurate and efficient software power estimation [5]. This paper is organized as follows: Section 2 suggests a possible strategy to extend the framework described in [1] and details the mathematical model along with its statistical properties; the tuning and validation methodologies adopted are described in Section 3, where the experimental results obtained are reported. Eventually, some conclusions are summarized in Section 4.

2. PROPOSED MODEL

This section introduces the extension of the previously developed model [1], to cover also inter–instruction effects. For the purpose of producing a widely applicable *static* estimation of the timing overheads related to the interaction between instructions, a taxonomy of a generic instruction