The Journal of Systems and Software 83 (2010) 1051–1075

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Software metadata: Systematic characterization of the memory behaviour of dynamic applications ☆

Alexandros Bartzas [a], Miguel Peon-Quiros [b,*], Christophe Poucet [c,d], Christos Baloukas [a], Stylianos Mamagkakis [c], Francky Catthoor [c,d], Dimitrios Soudris [e], Jose M. Mendias [b]

[a] *ECE Department, Democritus Univ. of Thrace, 67100 Xanthi, Greece*
[b] *DACYA/UCM, 28040 Madrid, Spain*
[c] *IMEC vzw, Kapeldreef 75, 3001 Heverlee, Belgium*
[d] *ESAT, K.U. Leuven, 3001 Heverlee, Belgium*
[e] *ECE School, National Technical Univ. of Athens, 15780 Zografou, Greece*

ARTICLE INFO

Article history:
Received 12 September 2008
Received in revised form 31 December 2009
Accepted 1 January 2010
Available online 13 January 2010

Keywords:
Software metadata
Profiling
Analysis
Embedded systems
Dynamic data types
Optimization

ABSTRACT

Development of new embedded systems requires tuning of the software applications to specific hardware blocks and platforms as well as to the relevant input data instances. The behaviour of these applications heavily relies on the nature of the input data samples, thus making them strongly data-dependent. For this reason, it is necessary to extensively profile them with representative samples of the actual input data. An important aspect of this profiling is done at the dynamic data type level, which actually steers the designers choice of implementation of these data types. The behaviour of the applications is then characterized, through an analysis phase, as a collection of *software metadata* that can be used to optimize the system as a whole. In this paper we propose to represent the behaviour of data-dependent applications to enable optimizations, rather than to analyze their structure or to define the engineering process behind them. Moreover, we specifically limit ourselves to the scope of applications dominated by dynamically allocated data types running on embedded systems. We characterize the *software metadata* that these optimizations require, and we present a methodology, as well as appropriate techniques, to obtain this information from the original application. The optimizations performed on a complete case study, utilizing the extracted software metadata, achieve overall improvements of up to 42% in the number of cycles spent accessing memory when compared to code optimized only with the static techniques applied by GNU G++.

1. Introduction

System-level design and optimization of embedded systems is a highly challenging task. Especially since embedded systems are becoming more and more complex, from both the hardware as well as the software perspective (Sangiovanni-Vincentelli, 2007). Nowadays, it is feasible to build a system-on-chip (SoC) accommodating a multitude of analogue and digital components. However, the development process of software components is the one that consumes the *majority* of the time-to-market and cost budget (Bernstein et al., 2004; Graaf et al., 2003; Frakes and Kang, 2005). Over the last few years, the main focus in the design of embedded systems has been to provide good performance and at the same time achieve low-power consumption. To achieve optimal results, a good coordination between hardware and software design is required. Therefore, memory-intensive applications running on embedded platforms (e.g., multimedia) must be closely linked to the underlying operating system (OS) and hardware. Putting all this together, it is clear that developing a complete, working system is an integration nightmare (Sangiovanni-Vincentelli, 2007).

Additionally, it is common practice to develop embedded platforms with extensive use of on-chip memory subsystems (i.e., caches and scratchpads) to improve the performance of new demanding applications. Such an overview of current embedded platforms is presented in Wolf (2004). Furthermore, the existing optimization methodologies mostly rely on purely compile-time information. As a result, most optimization research for embedded systems is focused on the effect of static data allocation and access

☆ This paper has been supported through a research grant from HiPEAC NoE http://www.hipeac.net.

* Corresponding author. Tel.: +34 615607843.

E-mail addresses: ampartza@ee.duth.gr (A. Bartzas), mikepeon@gmail.com (M. Peon-Quiros), poucetc@imec.be (C. Poucet), cmpalouk@ee.duth.gr (C. Baloukas), mamagka@imec.be (S. Mamagkakis), catthoor@imec.be (F. Catthoor), dsoudris@microlab.ntua.gr (D. Soudris), mendias@dacya.ucm.es (J.M. Mendias).

scheduling, decided at compile-time (Avissar et al., 2001; Benini and Micheli, 2000).

However, in modern consumer embedded systems, such as PDAs, smartphones and portable game platforms, it is common to have a number of complex dynamic applications (e.g., web browsing Bellavista et al., 2002, 3D rendering Nadalutti et al., 2006 or signal processing) running simultaneously. The latter are not amenable to pure static compile-time optimizations. The number, as well as the exact combination, of these applications running on modern embedded systems cannot be decided at design-time as this information depends on the input arriving from the user and the environment at run-time. The dynamicity of each application task, in turn, also depends on the user input, environment input and the interactions amongst the applications themselves as well as with the OS.

The problem then resides in optimizing the design of such systems using as much design-time information as possible, but leaving enough freedom to accommodate the dynamic aspects of their behaviour without resorting to worst-case bounded solutions. In order to tackle this problem, a new methodology that takes into account these variations in behaviour needs to be developed. This methodology will require extensive information about the static and dynamic characteristics of the applications.

Unlike the hardware metadata that is already standardized in the IP-XACT standard (The SPIRIT Consortium), there is not a standard representation, not even a definition, of software metadata to represent the characteristics of the dynamic data access behaviour of applications subject to varying inputs. In this paper, we propose a uniform representation of the dynamic data access and allocation behaviour of the applications, which we will define as *software metadata*. Additionally, we propose a systematic methodology to obtain this metadata from dynamic applications, because the study of applications that mainly use static data has already been well characterized and can be even done in an analytic way. Good overviews are presented in Panda et al. (1998). Instead, the behaviour of the applications that we target is dependent to a great degree on the nature of the specific input data stream, the behaviour of the user as well as the environment. The proposed metadata representation as well as the proposed profiling and analysis methodologies form a framework providing a system level representation of the behaviour of dynamic embedded software applications. The focus and contributions of this paper are situated in:

(1) Defining a uniform representation for the application metadata information as well as a methodology to extract it from dynamic applications.
(2) Introducing profiling and analysis techniques as a concrete implementation of this methodology. These profiling techniques improve on the current state of the art and may also be used for other general purpose profiling work.
(3) Illustrating how this metadata can be used by different system level methodologies and their relevance to the targeted memory management optimizations.

The rest of the paper is organized as follows. First, we give a motivational example and in Section 3 we discuss the related work. Next, we introduce the metadata representation as well as present the overall methodology required to gather this metadata in Section 4. The feasibility of software metadata extraction is presented in Section 5, where we introduce the specific method we use for obtaining the profiling information, and in Section 6, where we discuss the required analysis for turning this profiling information in desired data. Then, in Section 7, we show one case study that employs this metadata for various optimizations that concern the dynamic data energy consumption and memory footprint. Finally, conclusions are drawn and future work is outlined in Section 8.

2. Motivational example

Let us assume that three different groups (A, B and C) need to apply their new optimization methodologies. The first task they encounter is the characterization of the behaviour of the application(s) they wish to optimize. Each group will need to allocate some time to profile, run and analyze it. The "conventional" way to perform this task is illustrated in Fig. 1a. There, all of the groups perform the same steps – profiling and analysis (in different levels of granularity). Moreover, the information produced by each of the groups will not be suitable for the other groups if they do not share a common representation.

A new design flow would allow the information sharing of profiling and analysis data among different research groups (as it is shown in Fig. 1b). With a common representation for the software metadata of applications, the three independent groups would benefit from the characterization work performed by the others or even by a different group that worked previously on the same application. Additionally, the time required to perform the "global" profiling (able to cover the needs of the optimization methodologies) and analysis work is *much* less than the addition of the individual efforts. Let us assume that $f()^{\cdot}$ is the effort/time of performing profiling and analysis for one specific methodology, then $f(metadata) \ll f(A) + f(B) + f(C)$. Moreover, the fact that the relevant information is included in any analysis and that it has a common format allows saving time and applying it on the actual optimization work. Once the information is extracted, the rest of the teams will *not need to invest any time* on profiling and characterization.

3. Background and related work

Up until today, a lot of research has been performed in memory analysis and optimization techniques for embedded systems to reduce their power dissipation and increase performance (Panda et al., 2001; Benini et al., 2000). Traditional optimizations for embedded systems use compile-time, manifest information. The source code is completely transformed to a specific standardized form such that the analysis can easily happen (Catthoor et al., 2002). For modern dynamic applications this is no longer possible, as the dynamicity of behaviour due to the input dynamics cannot be captured by source code analysis alone.

On the aspect of profiling, most tools work directly on the binary application without requiring source code instrumentation. For example, `prof` (Graham et al., 1982) uses debugging information to find out the number of function calls and the time spent in each of them. However, it is not designed to provide insights for optimizations according to memory access patterns. More recent tools such as Valgrind (Nethercote, 2004) are able to look at the memory accesses and use this information to provide consistency checks for the executed programs. Valgrind allows tracing on which line in the program an access occurs, but cannot give a semantical analysis of which variable it is that was actually accessed. Even though the aforementioned tools do not require code instrumentation, they require a recompilation of the sources to include custom debugging information, hence requiring the source code to be present. Though it is possible to use Valgrind without recompilation, this means that the information can only be tied to the specific executable, and not to the source code or the variables therein. A framework that is able to perform link-time program transformations and instrumentation is presented in Put et al. (2005). The main target of that work is code-size reduction.

In Eeckhout et al. (2003) the authors explain how to choose input data sets to conduct the profiling phase of the development of a microprocessor in an efficient manner, by reducing the number of