Profile Guided Management of Code Partitions for Embedded Systems

Shukang Zhou Bruce R. Childers Naveen Kumar
Department of Computer Science, University of Pittsburgh
Pittsburgh, PA 15260
{zhou, childers, naveen}@cs.pitt.edu

Abstract

Researchers have proposed to divide embedded applications into code partitions and to download partitions on demand from a wireless code server to enable a diverse set of applications for very tightly constrained embedded systems. This paper describes a new approach for managing the request and storage of code partitions and we explore the benefits of our scheme.

1. Introduction

As embedded systems have become very diverse, there is a niche of such systems that have very tight constraints for memory, power, network capabilities, and cost, such as smart cards. We envision that in the near future they will be running non-trivial applications and be able to make wireless network connections at any time. As an example, we can imagine a smart card using a heavy-weight protocol like RSA to authenticate a user with the network or a server. Due to constrained memory on a smart card, it may be infeasible to have the entire program loaded on the card at any one moment. To address such memory constraints, researchers have proposed to partition applications into pieces and to download code partitions on demand via a wireless link between the client and the server [4]. As wireless bandwidth is limited and radio transmissions have high power consumption, the downloaded code partitions should be cached in a Code Partition Buffer (CPB) for future use. In this work, we investigate low overhead ways for managing the CPB at the software level to minimize wireless transfers. We propose a novel CPB management scheme that is light-weight and efficient and we provide insight into why such a scheme is beneficial.

2. Motivation

Applications for smart card devices often have small code working sets. For example, we experimented with several benchmarks from MediaBench and MiBench and found that the most frequently executed 4 KB of code accounts for 98.3% of the dynamic instruction count. Such a small working set shows that a small buffer can effectively capture the dynamic code footprint of these applications. However, the code must be intelligently managed to assure that frequently executed code in the working set is not replaced by infrequently executed code.

CPB management tries to keep active code in the buffer and to avoid including code which would not be executed in the near future. Similar to cache block and page replacement, we could use Least Recently Used, Least Frequently Used, and other policies to manage the CPB. However, such policies are expensive for a software managed CPB due to the need for instrumentation code to maintain usage information.

Applications for smart card devices are special ones: Their behaviors are often relatively simple, and their code properties are stable across different inputs. It is well known that most execution time is spent on a small part of the program code. And recent studies further show that much time is spent on a set of frequently executed *traces*. A trace is a dynamic sequence of executed basic blocks that directly capture dynamic control flow. Hazelwood [1] showed that regardless of data inputs for SPEC2000 programs, code traces that account for roughly 85% of the dynamic instruction count are repeated during successive executions. Hughes [2] found a similar trend for multimedia applications, where instructions per cycle and composition of instructions stay roughly constant at frame granularity, while the execution time of each frame can vary significantly.

Resource constraints on smart cards and the expected stable behavior of their applications lead us to believe that complicated run-time CPB management policies (e.g. LRU) are not necessary and too expensive. This motivates the use of static profiling information to guide CPB management at run-time, which has very low run-time overhead while still achieving good performance and minimizing wireless transfers.

3. Proposed Scheme

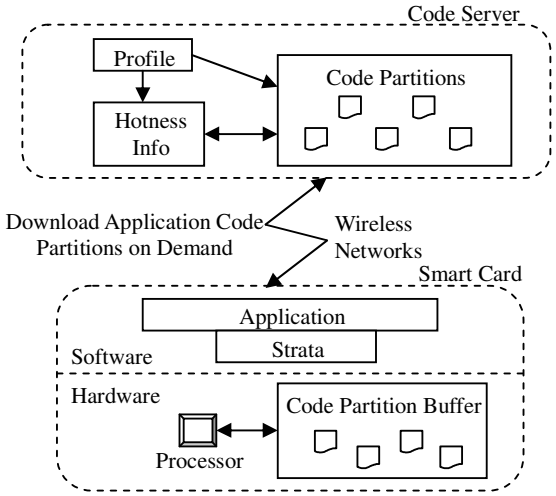

Figure 1: Functionality for Code-Partitioned Embedded Systems

Our proposed solution assigns code partitions different priorities in the CPB based on their frequency of execution (*hotness*) determined from program profiles. The functionality of our proposed solution is shown in Figure 1. We use static profiling to identify the hotness of each piece

of code and classify the code into three categories: (1) *long-lived hot code*, (2) *intermittently* or *short-lived hot code*, and (3) *cold code*. The hotness information is kept in the server.

In our scheme, a lightweight software layer, Strata [3], is included in a smart card's operating system. It controls an application's execution and manages the CPB. Here, the CPB is partitioned into three parts: (1) *persistent buffer*, (2) *temporary buffer*, and (3) *transient buffer*, which hold different code partitions based on hotness.

Before execution starts, Strata downloads long-lived hot code and stores it into the CPB because this code has the highest priority. This code is stored in the persistent buffer, which means the code is never replaced. Strata then starts program execution. If any code to be executed is not in the CPB, Strata sends a request to the server for the missing code. The server sends back the requested code with the hotness information for that piece of code. The hotness information determines where Strata stores the code. Intermittently or short-lived hot code is stored in the temporary buffer and cold code is stored in the transient buffer. The code in these two buffers can be replaced by newly downloaded code which will be stored in the same buffer. Replacement in the same buffer uses a simple circular management scheme to keep the overhead minimal.

The server can adapt the dynamic behavior of an application by adjusting the hotness classification if any relatively cold code is requested too often. The smart card can be adaptive, too. If code misses happen too often, the smart card can change the size of the buffers to adapt to program behavior.

4. Preliminary Results

To validate our proposal, we investigated dynamic behavior of the MediaBench and MiBench applications. One of our preliminary results is shown in Table 1, which shows the benefit of dividing the CPB for different code partitions. We found that a significant proportion of instructions (27.5% on average) are executed only once. Hence, we need to be careful that these instructions do not evict needed code. In our approach, such code will be put in the transient buffer to avoid replacing hot code. Traditional policies such as LRU may replace hot code that will be executed in the near future with cold code and our policy avoids this problem.

Use Domain	Benchmark	Executed Once Insn #	Total Executed Insn #	%
Telecom	ADMCPdec/enc	401/401	920/944	43.6/42.5
	GSMdec/enc	1295/1290	4759/8007	27.2/16.1
Image	unEPIC/EPIC	2064/2484	6679/8159	30.9/30.4
	JPEGdec/enc	3071/3034	8226/9949	37.3/30.5
Video	MPEG2dec/enc	1526/5785	8412/22201	18.1/26.1
Security	BLOWFISHdec/enc	311/307	1833/1830	17.0/16.8
Control	QSORT	621	3045	20.4
Average		**---**	**---**	**27.5**

Table 1: Executed Once Instructions

5. Discussion

Due to the simplicity of the proposed CPB management scheme, it has a low overhead. We also believe it is efficient because the dynamic property of code is exploited by profiling. In addition, there are a number of interesting issues related to this scheme, described below.

Large Local Memory. Some other embedded systems, like PDA's and mobile phones, have a larger local memory than a smart card. Even if the local memory is large enough to hold the whole program (e.g., as in a PDA), sending the code partitions on demand could still be beneficial when the executed code is a small portion of the whole program. One of our experiments showed that the average amount of code executed by MediaBench and MiBench applications is only 11.1% of the total static code size (the range is from 2.2% to 31.9%).

Security. The transmissions between the server and the smart card might leak application execution information. Zhang [4] proposed a tamper-resistant partitioning method that can be easily incorporated in our scheme. The transmission of long-lived hot code is insensitive to input, and when most execution time is in this code, little information is leaked. Furthermore, code can be encrypted by the server and decrypted by the smart card.

Partial Computing at Server Side. The server can perform partial compilation, optimization, or execution before sending code to the smart card. This may reduce code size (transmission overhead) and execution overhead.

Code Traces Layout. Arranging the code as traces can reduce instruction cache misses and can improve application performance. In our scheme, the server can organize traces in the persistent buffer and the client can arrange the temporary buffer.

Code Pre-fetch. Based on profiling information and a smart card's request history, the server could predict future requests and pre-send the code to the smart card.

Setup Delay. Sometimes it is possible that sending long-lived hot code prior to execution may cause unacceptable setup delay. To reduce the setup delay, the code needed to start program execution may be downloaded first and the remaining code can be downloaded on demand.

6. Future Work

To investigate the benefit of our CPB management scheme, we are implementing it in Strata. Our initial results are promising and we expect that our scheme will significantly reduce wireless transmission overhead.

7. References

[1] K. Hazelwood and M. D. Smith. Characterizing Inter-Execution and Inter-Application Optimization Persistence. *Workshop on Exploring the Trace Space for Dynamic Optimization Techniques*. June 2003.

[2] C. J. Hughes, P. Kaul, S. V. Adve, R. Jain, C. Park, and J. Srinivasan. Variability in the Execution of Multimedia Applications and Implications for Architecture. *Intl. Symp. on Computer Architecture*. June 2001.

[3] K. Scott, N. Kumar, S. Velusamy, B. Childers, J. Davidson, and M. L. Soffa. Reconfigurable and Retargetable Software Dynamic Translation. *Intl. Symp. on Code Generation and Optimization (CGO'03)*. March 2003.

[4] T. Zhang, S. Pande, and A. Valverde. Tamper-Resistant Whole Program Partitioning. *Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'03)*. June 2003.