Dynamic Code Overlay of SDF-Modeled Programs on Low-end Embedded Systems

Hae-woo Park Kyoungjoo Oh Soyoung Park Myoung-min Sim Soonhoi Ha

School of EECS, Seoul National University, Seoul, Korea
{starlet, kjoh, soy, min, sha}@iris.snu.ac.kr

Abstract

In this paper we propose a dynamic code overlay technique of synchronous data-flow (SDF) –modeled program for low-end embedded systems which lack MMU-support. With this technique, the system can utilize expensive SRAM memory more efficiently by using flash memory as code storage. SRAM is divided into several regions called overlay slots. A data-flow block or a cluster of data-flow blocks is loaded into the corresponding overlay slot on demand at run-time. Which blocks are clustered together and which overlay slots are allocated to the clusters are statically decided by the clustering and placement algorithm. We also propose an automatic code generation framework that generates the C-program code, dynamic loader and linker script files from the given SDF-modeled blocks and schematic, so we can run or simulate the program immediately without any additional coding effort. Experiments report that we can reduce the SRAM size significantly with a reasonable amount of time overhead for several real applications.

1. Introduction

A typical memory architecture for low-end embedded systems consists of ROM (i.e. NOR flash memory, NAND flash memory or mask ROM) for bootstrapping and SRAM for working memory. Depending on where to put the code, there are various memory architectures [2].

In this paper we are interested in the architecture where the code is also stored in NAND flash memory. Because of long access latency and sequential access requirement, a NAND flash memory cannot be used for code storage of execute-in-place (XIP) applications. Therefore the shadowing technique [3] is used to copy the whole code into SRAM at boot time. This architecture gives the best performance at run time while it slows down the boot process. But it has a serious drawback: SRAM should be big enough to store the whole code as well as the working data. Since SRAM is a cost and power bottleneck, reducing the SRAM size is a major concern for cost-sensitive low-end embedded system design.

In this paper, we propose a code overlay technique without compiler assistance, based on synchronous data-flow (SDF) –modeled programs [1]. Data-flow programs have been successfully used for specifying multimedia applications. A key advantage of an SDF program is that the schedule can be determined at compile-time. In addition, the proposed technique can find the optimal copy overhead by compile time analysis.

2. Related Work

Overlay technique has been widely used in computer systems where virtual memory system could not be used.

Park *et al.* [2] proposed a compiler-assisted demand paging technique. They presented a code clustering algorithm taking advantage of page-based load operation in NAND. Compared with that approach, our proposed technique utilizes the schedule information of the given SDF-modeled program so that the overlay cost is minimized.

Recently, various compiler techniques that exploit software-exposed speed-differentiated memory (a.k.a. scratch-pad memory) have been proposed [4] [5]. Most techniques for scratch-pad memory analyze the program at compile time and locate some parts of it on the scratch-pad memory to save energy or to improve performance. Those techniques differ from ours in that all programs need not be copied into SRAM before execution.

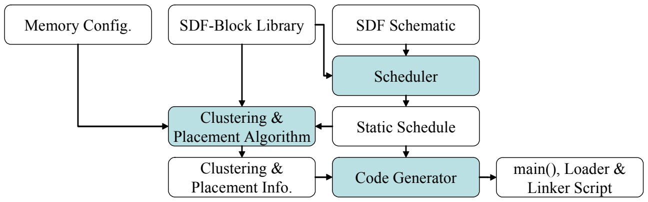
3. Proposed Code Generation Framework

The overall flow of the proposed code generation framework is shown in Figure 1. An SDF-modeled program composed of well-defined functional blocks is fed into the framework as input. We assume that each function block is written in C code, and the code size of the block is given from the block library. We first determine a static schedule of the SDF program graph using existent SDF scheduling algorithms.

In the proposed approach, execution buffer space in

3-9810801-0-6/DATE06 © 2006 EDAA

SRAM is divided into regions called overlay slots. A data-flow block or a cluster of blocks is loaded into the overlay slots on demand at run-time. The clustering and placement algorithm puts together data-flow blocks into *cluster*s, and determines where each cluster is placed in NAND and SRAM so that the overall run-time copy overhead is minimized and/or the required SRAM size is minimized. The optimal clustering and placement decision is delivered to the code generator. The proposed clustering and placement algorithm is based on genetic algorithm which gives a good result in reasonable time while a general branch-and-bound algorithm consumes too much time.



Memory Config.
SDF-Block Library
SDF Schematic
Scheduler
Clustering & Placement Algorithm
Static Schedule
Clustering & Placement Info.
Code Generator
main(), Loader & Linker Script

Figure 1: Overall flow of the proposed framework

Then our framework generates the main() function by stitching the codes of blocks according to the schedule sequence. Before calling a block, the *loader* function is called to load the cluster which includes the block from NAND flash memory to SRAM if needed. We enforce main() function to be always resident in SRAM and make it supervise the control transfer between blocks. The *linker script* file is also automatically generated. This file informs the linker of the clustering and placement information.

After all, the framework builds the executable binary with the generated files.

4. Experimental Results

We implemented the proposed code generation framework in PeaCE environment [6], and use ARM developer suite (ADS) as compiler and simulator with the following parameters: ARM7TDMI, 35ns NAND access time, and 512/2048 byte NAND page size. Applications we have used for experiments and their characteristics are as shown in Table 1.

Table 1: Test applications

Application	Number of Blocks	Largest Block Size (Code)	Total Block Size (Code)
butterfly	17	144	828
cd2dat	12	376	2,004
JPEG encoder	7	1,156	2,992
H.264 decoder	32	20,872	59,636

Figure 2 show the total running cycles of each application as the overlay region size and NAND page size vary. The graphs show that the total cycle increases as the overlay region size decreases because of the increasing copy overhead. The result shows that the result shows that we can save on average significant amount of SRAM with relatively small performance degradation.

5. Conclusion

In this paper, we presented an automatic code generation framework for dynamic code overlay. The proposed framework generates not only the program code, the loader, and the linker script automatically from the data-flow program graph, so the programmer may be ignorant of the overlay technique.

The framework has been tested on well-known applications. Experiments report that the SRAM size used for code region can be reduced significantly with a reasonable amount of performance degradation.

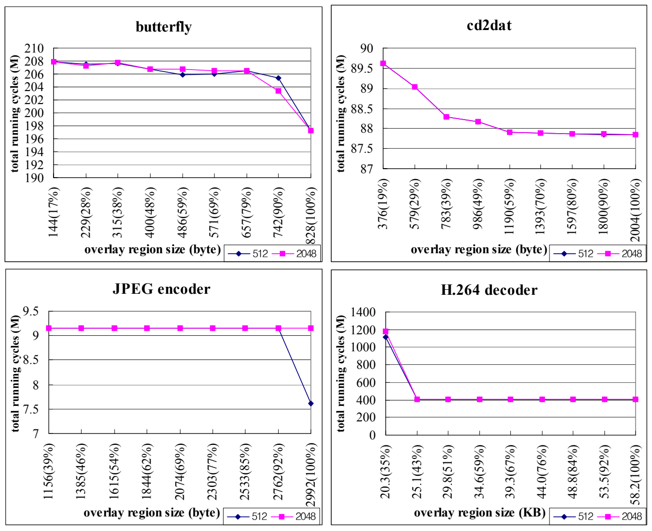


Figure 2: Total Running Cycles

Acknowledgment

This work was supported by National Research Laboratory Program (No. M1-0104-00-0015), Samsung Electronics, IT leading R&D Support Project funded by Korean MIC, and ITSoC Project.

References

[1] Edward A. Lee and David G. Messerschmitt., Synchronous Data Flow. IEEE Proceedings, 1987.

[2] Chanik Park *et al.*, Compiler Assisted Demand Paging for Embedded Systems with Flash Memory. Fourth Int. Conf. on Embedded Software (EMSOFT04), Pisa, Italy, 2004.

[3] Jean Chao *et al.*, Cost Savings with NAND Shadowing Reference Design with Motorola^{TM} MPC8260^{TM} and Toshiba^{TM} CompactFlash^{TM}., July 2002.

[4] Oren Avissar and Rajeev Barua., An Optimal Memory Allocation Scheme for Scratchpad-Based Embedded Systems. IEEE Transactions on Embedded Computing Systems, 1(1):6-26, 2002.

[5] Manish Verma *et al.*, Dynamic Overlay of Scratchpad Memory for Energy Minimization. In Proceedings Int. Conf. HW/SW codesign and system synthesis, pp. 104-109, 2004.

[6] http://peace.snu.ac.kr