Extended Control Flow Graph Based Performance Optimization Using Scratch-Pad Memory*

Pu Hanlai, Ling Ming, Jin Jing
National ASIC System Engineering Technology Research Center
Southeast University, Nanjing 210096, China
{pessia, trio, dsa}@seu.edu.cn

Abstract

This paper presents an exploration approach for the researcher to choose the suitable size of Scratch-Pad memory (SPM) for maximal performance improvement of a specified application. The approach uses an extended control flow graph (ECFG) to describe the application and provides a solution to reduce the additional overhead of moving nodes to SPM. Experiments achieves on average 11% increase in performance compared to the previous approaches and 44% decrease in the application's runtime compared to none SPM environment.

1. Introduction

As an important architectural consideration, SPM fills up the increasing gap between high frequency of embedded microprocessor cores and low access speed of off-chip memories which results in a heavy fall of chip performance. The decision about a suitable size of SPM has become an important problem in SoC design. In none cache system, the general solution tries to move the most frequently accessed parts of the application to SPM and establish the relationship between the SPM size and the application's performance. The relationship helps the researcher to choose the suitable size. This method has been adopted in all the previous researches[2][3][4][5] in none cache system. To our knowledge, only Steinke et al[4] analyze the instructions of the application and divides each function into several nodes according to the normal control flow graph (CFG). They adopt knapsack algorithm to select suitable nodes. Unfortunately, they ignore the relationships of nodes which have considerable influence on space utilization of SPM. The negative effect is caused by the strict limitations of addressing space of some ARM instructions[1], especially such as branch (B/BL) instruction and data-loading (LDR) instruction. Commonly, the huge address gap between SPM and off-chip memory blocks single branch instruction's free jump. It should be replaced by a macro consisting of two branch instructions and one data-loading instruction. Similarly, some data-loading instructions in SPM need additional space to save their operating address. All of these lead to an increase in size of nodes moved to SPM. However, if two related nodes are both selected, their total size is likely reduced because of the decrease in the number of long branch macros and additional space of data-loading instructions. Therefore, the neglect of the relationships of nodes must decrease the space utilization of SPM and lead to a minor decision about the SPM size.

1 This research was sponsored by National Natural Science Foundation under grant 60176018.

2. Proposed Exploration Approach

To collect the application's runtime information, our target architecture model is built based on ARMulator (ARM7TDMI)[1], including three parts, SPM, off-chip memory (SDRAM) and instruction FIFO (IF). IF is used to fetch instructions saved in SDRAM via burst read operation. The diagram in Fig. 1 shows the work flow of our approach.

2.1. ECFG Generation

ECFG is built directly from the application's execution file and independent of input data. Besides detail information of the normal CFG node, each ECFG node gives a description of its call graph and the global data variables accessed by it. ECFG transforms the whole application into a directed graph consisting of nodes and relationships.

2.2. SPM Allocation Algorithm

A refined knapsack algorithm was adopted to solve the problem during the process of SPM allocation. The set $I(i)$ is defined: $I(i) = \{I_j\}, \forall j \in [1,i]$. If n_j has been selected,

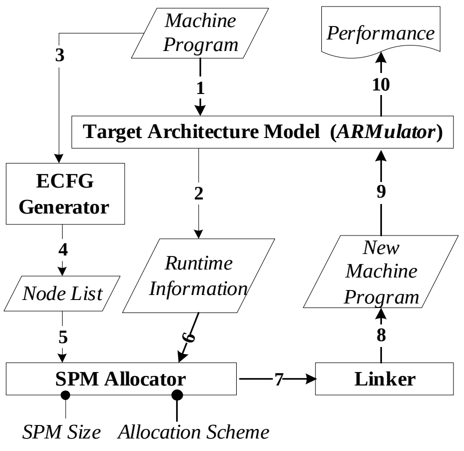

Figure 1. Work flow

Benchmark	T	T_p	T_o	R_{op}	R_o
Dijkstra	44.7	28.0	15.9	-43%	-64%
Jpeg2bmp	191	141	121	-14%	-37%
Gunzip	15.7	12.1	11.4	-6%	-27%
MP3player	379	291	282	-3%	-26%
Bubblesort	140	87.5	87.3	-0.2%	-38%
CRC32	9.21	2.71	2.71	0	-71%
Average				-11%	-44%

Table 1. Performance [10^6cycles] comparison between the previous approach (T_p) and our approach (T_o). SPM size is 2Kbyte. $R_{op} = \frac{T_o - T_p}{T_p} \times 100\%$, T **is the program's original runtime, and** $R_o = \frac{T_o - T}{T} \times 100\%$

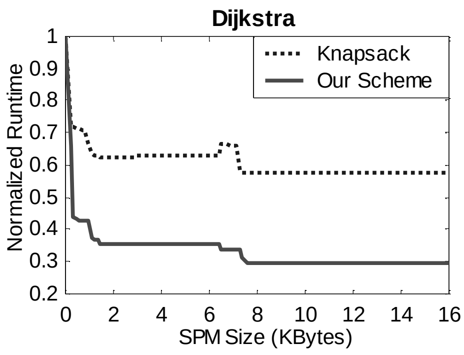

Figure 2. Dijkstra: Runtime comparison, SPM size(Byte) =$128 \times n$, $1 \leq n \leq 128$

$I_j = 1$, otherwise it is zero. $I(0)$ means none is selected. $P(n_{i+1}, I(i))$ and $S(n_{i+1}, I(i))$ are the effect on runtime and size caused by moving n_{i+1} to SPM while some nodes has been in SPM, respectively.

The algorithm works as follows: (1) Sort all nodes descendingly by their priority, which is defined as: $priority(n_i) = \frac{P(n_i, I(0))}{S(n_i, I(0))}$. (2) Check each node and calculate the total saved runtime (TP) until SPM is filled or no node is remained. The precondition of the selection is, $S(n_{i+1}, I(i)) + TS(i) \leq TS_{SPM}$. TS_{SPM} is the specified SPM size.

$$TP(i+1) = max \begin{cases} TP(i) + P(n_{i+1}, I(i)), & n_{i+1} \text{ is selected} \\ TP(i), & \text{otherwise} \end{cases}$$

3. Experiment and results

In Table 1, it clearly shows that our approach achieves on average 11% increase in performance compared to the previous approaches and 44% decrease in the application's runtime compared to none SPM environment.

Fig. 2 shows that the application's runtime optimized by the previous approaches using simple knapsack algorithm is not always decreasing while the SPM size increases. The reason is that the negative effect caused by the imported instructions exceeds the obtained profit of the selected nodes. However, our approach avoids this bad situation and brings a more decrease of runtime.

4. Conclusions

We presents an exploration approach to decide the suitable size of SPM in SoC design. The special analysis of the relationships of nodes of the application leads to significant performance improvements compared to the previous researches.

References

[1] ARM. http://www.arm.com.

[2] J. Sjödin, B. Fröderberg, and T. Lindgren. Allocation of global data objects in on-chip ram. *Proc. Workshop on Compiler and Architectural Support for Embedded Computer Systems*, December 1998.

[3] O. Avissar, R. Barua, and D. Stewart. An optimal memory allocation scheme for scratch-pad based embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 1(1):6–26, November 2002.

[4] S. Steinke, C. Zobiegala, L. Wehmeyer, and P. Marwedel. Moving program objects to scratch-pad memory for energy reduction. Technical Report 756, University of Dortmund, Dept. of CS XII, October 2001.

[5] Stefan Steinke, Lars Wehmeyer, Bo-Sik Lee, and Peter Marwedel. Assigning program and data objects to scratchpad for energy reduction. Paris, France, March 2002. Design, Automation and Test in Europe Conference (DATE).