Energy-oriented dynamic SPM allocation based on Time-Slotted Cache Conflict Graph

WANG huan, ZHANG yang, MEI chen, LING ming
National ASIC System Engineering Technology Research Center
Southeast University, Nanjing 210096, China
{nanqihao, tianna1121, mei.mc121}@gmail.com, trio@seu.edu.cn

Abstract—Energy consumption has always been considered as the key issue of the state-of-the-art SoCs. Implementing an on-chip Cache is one of the most promising solutions. However, traditional Cache may suffer from performance and energy penalties due to the Cache conflict. In order to deal with this problem, this paper firstly introduces a Time-Slotted Cache Conflict Graph to model the behavior of Data Cache conflict. Then, we implement an Integer Nonlinear Programming to select the most profitable data pages and employ Virtual Memory System to remap those data pages, which can cause severe Cache conflict within a time slot, to the on-chip Scratchpad Memory (SPM). In order to minimize the swapping overhead of dynamic SPM allocation, we introduce a novel SPM controller with a tightly coupled DMA to issue the swapping operations without CPU’s intervention. The proposed method can optimize all of the data segments, including global data, heap and stack data in general, and reduce 24.83% energy consumption on average without any performance degradation.

Keywords—Time-Slotted Cache Conflict Graph; Scratchpad Memory; Energy Optimization; Virtual Memory System

I. INTRODUCTION

Reducing the energy consumption remains a major concern of the state-of-the-art SoCs. Many studies have shown that reducing the off-chip memory accessing would reduce the total energy consumption of embedded systems. However, traditional Cache, implemented to take advantage of locality of program, may cause severe cache conflict due to its capacity as well as association, and compromise the system performance and increase the energy eventually.

Some related researches [1] proposed a method to copy some portions of program, based on the static Cache Conflict Graph (CCG), to the Scratchpad Memory (SPM), and achieved promising results. However, there are two main drawbacks of their methods. Firstly, the CCG cannot be used to model the dynamic Cache conflict behavior, since most of applications have several independent execution phases, each of which has totally different memory accessing patterns. Secondly, their methods may not suit all for the data sections.

This paper introduces a novel Time-Slotted Cache Conflict Graph (TSCCG) to slice the whole program execution into several phases, and establish the conflict graph within each of those phases independently. Then the Virtual Memory System (VMS) is adopted to remap portions of data segments to the on-chip SPM according to TSCCG. Whenever the CPU requires a data, remains in globe, heap or stack segments, the Memory Management Unit (MMU) will decide the real physical address space, which could be main memory (access through Cache) or SPM. Moreover, in order to minimize the swapping overhead of dynamic SPM allocation, we introduce a novel SPM controller with a tightly coupled DMA to issue the swapping operations without CPU’s intervention. Finally, this paper introduces an Integer Nonlinear Programming (INP) to select the most profitable pages.

This paper is organized as follows. First, we briefly describe the background information and discuss the difference between our methods and prior related work (Sect.II). Then, we detail the hardware implementation and software strategies necessary to manage the SPM at run-time (Sect.III). Subsequently, we establish the energy model of our scheme, and implement an INP to find an optimal solution (Sect.IV). The experimental results will be explained in Section V, and finally Section VI concludes this paper.

II. RELATED WORKS

Cache, which improves the system performance by the locality of program, is transparent to the programmers. However, due to its high energy consumption, large area occupation, Cache has been restricted in embedded systems. By contrast, SPM requires explicit support from the programmer while requiring less energy and on-chip areas. In order to fully take advantage of their characteristics, most of the high-end embedded microprocessors employ both Cache and SPM to optimize the performance and energy consumption.

Hyungmin introduced a dynamic SPM management based on MMU to deal with data [2]. The MMU provided a logically sequential address space which scattered in different physical address spaces. However, they used LDR/STR instructions to swap the contents of SPM, which may contaminate instruction Cache and eventually compromise the total profit.

Stack and heap data, allocated and used at run-time, could not be analyzed through traditional SPM optimization algorithm. Soyoung [9] proposed a dynamic address mapping with a MMU for stack management without architecture modification and compiler assistance. Since our method is based on VMS, we can dynamically remap any portion of data segments (including global data, stack and heap data) into SPM without compiler’s assistance.

Compared with earlier researches, Poletti Francesco [10] placed a DMA to enable memory transfers between the SPM
and the main memory. However, their method had to insert some high level APIs into the source code to configure the DMA. This required direct access to the source code, which might not be available when the optimization was implemented.

III. PLATFORM

In this section we will detail the hardware implementation and software strategy. The hardware consists of an enhanced VMS, an SPM controller with a tightly coupled DMA and a traditional timer. The software strategy sub-section will provide the fundamental basis of this paper: the Time-Slotted Cache Conflict Graph. And then, the complete optimization process will be illustrated step by step.

A. Hardware Implementation

As mentioned above, with traditional VMS we can remap those memory accessing, which will lead to severe Data Cache conflict, into the SPM. However, due to the relatively large conventional MMU page (typically 4KB / page), the contents of SPM cannot be managed in fine-grained, and eventually compromise the SPM utilization. More over, the relatively large page will significantly increase burden on swapping the SPM. We therefore adopt the micro-paged MMU (512 Bytes / page) proposed by Hyungmin [2].

![Figure 1. SPM Controller with A Tightly Coupled DMA](image)

In order to minimize the swapping overhead, this paper proposes a novel SPM controller with a tightly coupled DMA, shown in Fig. 1. Any DMA operation will be configured and issued by the SPM controller, consisted of the SPM control logic, a tightly-coupled DMA and a group of region registers. The SPM control logic is primarily used to load configuration of current time-slot from configuration pool and configure the DMA to swap SPM during the timer interrupt. The SPM Region Registers, shown in Fig. 2, preserve information about the page size, two flags (valid bit and dirty bit) and the original physical page number, which will be loaded into DMA destination register when swapping SPM. The complete process will be presented later.

In addition, this paper introduces a conventional timer to slice the execution progress into several phases, and help us to explicitly model the run-time characteristics of Data Cache conflict behaviors.

![Figure 2. SPM Region Registers](image)

B. Software Strategy

The TSCCG introduces a new dimension: TIME into the CCG. Therefore we can slice the whole program execution into several phases, and establish the conflict graph within each of these phases independently, as shown in Figure 3.

![Figure 3. the Time-Slotted Cache Conflicts Graph](image)

The TSCCG \( G(T)=\langle Page, E \rangle \) is a directed weighted graph with node set \( Page=\{Page_1,...,Page_n\} \). \( T(Time\ slot) \) indicates the time slot of the execution time. Each vertex \( Page_i \) in \( G \) corresponds to a memory page, which is also the granularity of our optimization method. The edge set \( E \) contains an edge \( e_{ij} \) from node \( Page_i \) to \( Page_j \), if a Cache line belonging to \( Page_j \) is replaced by Cache line belonging to \( Page_i \). The weight \( m_{ij} \) of the edge \( e_{ij} \) is the number of Cache misses of \( Page_i \) that occur due to \( Page_j \). The weight \( dw(n) \) of vertex \( Page_i \) stands for the total accessing times within \( Page_i \).

C. Co-Design and Simulation Flow

We implement a Timer Counter Register (TCR) to preserve the index of current Time Slot. During the timer interrupt, the swapping of SPM is accomplished through the SPM controller:

Step 1: Load the index of current Time Slot from the TCR to determine the offset of current configuration within the configuration pool.

Step 2: When a new page has to be swapped into SPM, check the dirty bit of the correspondent SPM Region Register. If it is set, a DMA swapping out operation will be started.

Step 3: Modify the page table entries which should be updated within this time slot.
Step 4: The SPM controller configures DMA to start a swapping in operation according to the configuration, and clear the correspondent dirty bit.

IV. ENERGY MODEL AND ALGORITHM

We establish TSCCG to model the Cache conflict behaviors of every page according to the profile information. To select the pages which cause more cache conflict (the most profitable pages), we abstract TSCCG to a mathematical model and then use some mathematical tools to obtain the optimal solution.

A. Energy Model

We will firstly establish the system’s energy model to quantify the fluctuation of system energy when some of the nodes have been remapped to the SPM. Note that, the energy consumed by those cacheable data pages, in the given time slot $T_{S_i}$ can be described by the following equation:

$$E(page, DCache, TS_i) = RHit(page, TS_i)*E_{DCache, Rhit} + RMiss(page, TS_i)\times E_{DCache, Rmiss} + W\times RHit(page, TS_i)*E_{DCache, Rhit} + Wmiss(page, TS_i)\times E_{DCache, Wmiss}$$  \hspace{1cm} (1)

The $RHit()$ and $RMiss()$ return the number of read/write hit and read/write miss within $T_{S_i}$. In our scheme, the write-through and read allocation Cache is adopted. On this premise conditions, (1) can be simplified as follows:

$$E(page, DCache, TS_i) = RHit(page, TS_i)*E_{DCache, Rhit} + RMiss(page, TS_i)\times E_{DCache, Rmiss}$$  \hspace{1cm} (2)

For a page remained in the main memory, if it is remapped to SPM, the energy consumed within $T_{S_i}$ can be depicted by the following equation:

$$E(page, SPM, TS_i) = w(n_i)*E_{spm, read} + l(page, TS_i)\times \left[ dpagesize\times (E_{spm, read} + E_{spm, read}) + (1+W)\times E_{spm, write} \right]$$  \hspace{1cm} (3)

$$l(page, TS_i) = \begin{cases} 1 & \text{if page is not in SPM yet} \\ 0 & \text{otherwise} \end{cases}$$

In (3), $l(page, TS_i)$ stands for the overhead to swap a page from main memory to SPM. The factor $W(0 \leq W \leq 1)$, obtained by experiments, is the probability of writing back a single page.

B. Mathematical Model

In order to select the most profitable pages, we combine the two energy model (Cache and SPM) into a unified formation. The $p(page, TS_i)$ denotes the location of the page in the memory hierarchy.

$$p(page, TS_i) = \begin{cases} 1 & \text{if page is in Cache} \\ 0 & \text{if page is in SPM} \end{cases}$$  \hspace{1cm} (4)

In $T_{S_i}$, the energy consumption is described as (5) when page is accessed.

$$E(page, TS_i) = (page, TS_i)\times E(page, DCache, TS_i) + \left[1 - p(page, TS_i)\right]\times E(page, DCache, TS_i)$$  \hspace{1cm} (5)

The $E(TS_i)$ denotes the total energy consumed within $T_{S_i}$:

$$E(TS_i) = \sum_{page \in N(page, TS_i)} E(page, TS_i)$$  \hspace{1cm} (6)

The SPM size constraint can be modeled as (7):

$$\sum_{i \in N} pagesize*[1 - p(page, TS_i)] \leq spmsize$$  \hspace{1cm} (7)

Now we can describe the Mathematical model as (8). The goal of dynamically allocating the data pages is to minimize the (6), and the constraint condition is (7).

$$\min \{E(TS_i) = \sum_{page \in N(page, TS_i)} E(page, TS_i)\}$$

Subject to

$$\sum_{page \in N(page, TS_i)} pagesize*[1 - p(page, TS_i)] \leq spmsize$$  \hspace{1cm} (8)

C. Integer Nonlinear Programming

Since there is a quadratic term (9) in (6), the math problem (8) is a typical integer nonlinear problem.

$$p(page, TS_i)\sum_{page \in N(page, TS_i)} Miss(page, page_j)$$

$$= (page, TS_i)\sum_{page \in N(page, TS_i)} p(page, TS_i)*dw(n_i)$$  \hspace{1cm} (9)

$$= \sum_{page \in N(page, TS_i)} p(page, TS_i)*p(page, TS_i)*dw(n_i)$$

In this paper, we use MATLAB Toolbox $bnb20$ [12] to solve the nonlinear optimization problems (8) and obtain an optimal pages which minimizes the objective function.

V. EXPERIMENTAL RESULTS

Since our optimization algorithm focuses on reducing the system energy consumption by remapping those pages which would cause severe Cache conflicts to SPM, it can be clearly figure out the Cache miss reduction by comparing the distribution of Cache miss.

(a) Before Optimization  (b) After Optimization

Figure 4. The Cache misses’ distribution of Benchmark Dijkstra
We take 14 benchmarks from MediaBench [3] and MiBench [4] to quantitatively analysis the results of our optimization method. The processor’s energy is based on [5]. The energy consumed by Cache, TLB and SPM are obtained from Cacti 3.2 [11]. The SDRAM and bus energy consumption can be found in [6] and [7]. Note that, we ignore the energy consumed by SPM controller with a tightly coupled DMA. According to [8], only memory accessing energy, issued by DMA, will be included. After optimization, the SDRAM and bus energy consumption has reduced significantly while additional SPM accessing energy is neglected.

In Fig. 6 we compare the optimization results with 8K direct-mapped and 8K 4-way set-associative Cache. Compared with 8K Byte directed-mapped Data Cache, our approach, employing 4K Byte Data Cache and 4K Byte SPM, reduces 24.83% energy consumption on average without any performance degradation.

VI. CONCLUSIONS

In this paper, we introduce a VMS to remap those data pages, which can cause serious Cache conflicts within a time slot, to the SPM according to our innovated Time-Slotted Cache Conflict Graph. Then, a novel SPM controller with a tightly coupled DMA is implemented to minimize the swapping overhead of dynamic SPM allocation. Finally, we introduce the INP to select the most profitable pages. The experimental results show that our approach can reduce a great deal of energy while guarantee the performance.

REFERENCES