Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

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

A survey of architectural techniques for improving cache power efficiency

Sparsh Mittal*

Future Technologies Group, Oak Ridge National Laboratory (ORNL), Oak Ridge, TN, United States

A R T I C L E   I N F O

Article history:
Received 13 November 2012
Received in revised form 17 July 2013
Accepted 6 November 2013

Keywords:
Cache energy saving techniques
Dynamic energy
Leakage energy
Power management
Energy efficiency
Green computing

A B S T R A C T

Modern processors are using increasingly larger sized on-chip caches. Also, with each CMOS technology generation, there has been a significant increase in their leakage energy consumption. For this reason, cache power management has become a crucial research issue in modern processor design. To address this challenge and also meet the goals of sustainable computing, researchers have proposed several techniques for improving energy efficiency of cache architectures. This paper surveys recent architectural techniques for improving cache power efficiency and also presents a classification of these techniques based on their characteristics. For providing an application perspective, this paper also reviews several real-world processor chips that employ cache energy saving techniques. The aim of this survey is to enable engineers and researchers to get insights into the techniques for improving cache power efficiency and motivate them to invent novel solutions for enabling low-power operation of caches.

1. Introduction

As we are entering into an era of green computing, the design of energy efficient IT solutions has become a topic of paramount importance [1]. Recently, the primary objective in chip design has been shifting from achieving highest peak performance to achieving highest performance-energy efficiency. Achieving energy efficiency is important in the design of all range of processors, such as battery-driven portable devices, desktop or server processors to supercomputers.

To meet the dual and often conflicting goals of achieving best possible performance and best energy efficiency, several researchers have proposed architectural techniques for different components of the processor, such as processor core, caches and DRAM (dynamic random access memory). For several reasons, managing energy consumption of caches is a crucial issue in modern processor design. With each CMOS (complementary metal oxide semiconductor) technology generation, there is a significant increase in the leakage energy consumption [2,3]. According to the estimates of International Technology Roadmap for Semiconductors (ITRS); with technology scaling, leakage power consumption will become a major industry crisis, threatening the survival of CMOS technology itself [4]. Further, the number of processor cores on a single chip has greatly increased over years and future chips are expected to have much larger number of cores [5]. Finally, to bridge the gap between the speed of processor and main memory, modern processors are using caches of increasingly larger sizes. For example, modern desktop processors generally have 8 MB last level caches [6] while server systems are designed with 24–32 MB last level caches [7,8]. On chip caches consume 16% of the total power in Alpha 21264 and 30% of the total power in StrongARM processor core [9]. In both Niagara and Niagara-2 processors, L2 cache consumes nearly 24% of the total power consumption [10]. Thus, the power consumption of caches is becoming a large fraction of processor power consumption. To address the challenges posed by these design trends, a substantial amount of research has been directed towards achieving energy efficiency in caches. The focus of this paper is to review some of these techniques.

The rest of the paper is organized as follows. Section 2 presents a brief background on modeling of CMOS energy consumption and discusses the essential design principles which guide cache energy saving approaches. Sections 3 and 4 discuss the approaches which have been proposed for saving dynamic and leakage energy, respectively. In these sections, we first discuss the landscape of the techniques proposed and then discuss a few of them in more detail. We highlight the basis of similarities and differences between the techniques. Since leakage energy is becoming an increasing fraction of cache power consumption [2,3,11], we focus more on leakage energy saving techniques than dynamic energy saving techniques. Section 5 discusses the approaches which are used for saving both dynamic and leakage energy. To show real-life implementation of the research ideas, Section 6 discusses some of the commercial chip designs which employ cache energy saving techniques. Finally, Section 7 concludes the paper.

* Tel.: +1 865 574 8531.
E-mail address: sparsh0mittal@gmail.com

2210-5379/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.suscom.2013.11.001

As it is not possible to cover all the techniques in full detail in a review of this length, we take the following approach to restrict the scope of the paper. Although the techniques designed to improve performance are also likely to save energy, in this survey we only consider those techniques which aim to optimize energy efficiency and have been shown to improve energy efficiency. Further, cache energy saving can also be achieved using circuit-level techniques (e.g., low-leakage devices), however, in this paper we mainly focus on architecture-level techniques which allow runtime cache power management. Lastly, since different techniques have been evaluated using different simulation infrastructure and workloads, we do not include their quantitative improvement results. Rather, we focus on their key design principles, which can provide valuable insights.

Table 1
Classification of techniques proposed for different caches.

Caches/criterion	Energy-saving techniques (ESTs)
For first-level cache (L1)	[23,24,27,28,34,39,71,148]
For last-level caches (L2 or L3)	[14–16,33,48,70,73,75,81,83,84,96]
For instruction caches	[21,35,42,47,59,62,66,68,69,74,85,119,149,150]
For data caches	[9,24,28,28,38,40,58,63,148]
ESTs utilizing hardware support	[26,31,39,58,60,64,65,71]
ESTs utilizing software support	[19,21,34,41,75,83,84,90,96]
ESTs utilizing compiler support	[35,36,52,63,85]

2. Background and related work

The power consumption of CMOS circuits is mainly classified in two parts, namely dynamic power (also called active power) and leakage power (also called static power). In what follows, we present the modeling equations for both dynamic and leakage power in their simplified forms, which helps to gain insights into how energy saving techniques work. For a more detailed modeling, we refer the reader to [12,13].

Dynamic power ($P_{dynamic}$) is dissipated whenever transistors switch to change the voltage in a particular node, while leakage power ($P_{leakage}$) is dissipated due to leakage currents that flow even when the device is inactive. Mathematically, they are given by,

$$P_{dynamic} = \alpha \times C_{eff} \times V_{DD}^2 \times F \tag{1}$$

$$P_{leakage} = V_{DD} \times I_{leak} \times N \times k_{design} \tag{2}$$

Here α shows the activity factor, V_{DD} shows the supply voltage, C_{eff} shows the effective capacitance and F shows the operating frequency. Further, N is the number of transistors, k_{design} is the design dependent parameter and I_{leak} is the leakage current, which is a technology dependent parameter.

From Eq. (1) we infer that, for a given CMOS technology generation, dynamic power consumption can be reduced by adjusting voltage and frequency of operation or by reducing the activity factor (e.g., by reducing the number of cache accesses or the number of bits accessed in each cache access, etc.). Similarly, from Eq. (2), it is clear that, for a given CMOS technology generation, the opportunity of saving leakage energy lies in redesigning the circuit to use low-power cells, reducing the total number of transistors or putting some parts of caches into low (or zero) leakage mode. Based on these essential principles, several architectural techniques have been proposed, which we discuss in the next sections.

Modern processor cores have multi-level cache hierarchy with L1, L2, L3 caches etc. Also, typically at level one, data and instruction caches are designed as separate caches, while at lower levels (i.e., at level 2 and 3), instruction and data caches are unified. These caches have different properties and different techniques utilize these properties to save cache energy. Table 1 summarizes the techniques which are proposed for L1 or L2/L3, and for instruction or data caches. More detailed discussion of some of these techniques is presented in the following sections.

First-level caches (FLCs) are designed to minimize access latency, while last level caches (LLCs) are designed to minimize cache miss-rate and the number of off-chip accesses. Accordingly, FLCs are smaller (e.g., 32 KB or 16 KB), have smaller associativity and employ parallel lookup of data and tag arrays [14]. In contrast, LLCs are much larger (e.g., 2 MB, 4 MB, etc.), have higher associativity and employ serial (phased) lookup of data and tag arrays. Thus, as an example, an energy saving technique which increases the cache access latency is more suitable for LLCs, than for FLCs. Also, due to their relatively smaller sizes and large number of accesses, FLCs spend a larger fraction of their energy in the form of dynamic energy, while LLCs spend a larger fraction of their energy in the form of leakage energy [15,16]. As for instruction/data caches, instruction access stream exhibits strong spatial and temporal locality and hence, the instruction cache is very sensitive to increase in access latency. The working set and reuse characteristic of instruction cache is different from that of data cache. Also, since instruction cache does not hold dirty data, reconfiguring it does not lead to write-back of dirty data and thus, reconfiguration of instruction cache can be more easily implemented than that of the data cache.

Table 1 also classifies the techniques based on whether they need hardware, software and/or compiler support. While compiler-based approaches incur no or minimal hardware overhead, compiler analysis may not be possible in all situations and also compiler only has limited information. Software-based approaches can leverage the software to make more complex decisions and consider the impact of energy saving techniques on components of the processor other than the cache, however, software-only approaches generally cannot exercise the opportunity of frequent reconfigurations and also incur larger implementation overhead than hardware-only approaches. Hardware-based approaches can utilize simple yet low-overhead algorithms, and can also exercise the opportunity of fine-grained and frequent reconfigurations. However, these approaches cannot easily take other components of processor into account.

Kaxiras and Martonosi [17] survey some of the architectural techniques proposed for saving energy in processors and memory hierarchies. This paper differs from their work, in that we review several recent developments which have been made in the fast evolving field of design of energy efficient architectural techniques. Also, we exclusively focus on the techniques aimed to save cache energy to provide more in-depth insights. Finally, to show the practical application of the research ideas, we also discuss the examples of many commercial chips which use cache energy saving designs.

3. Dynamic energy saving approaches

3.1. Overview

Recently several techniques have been proposed for saving dynamic energy. Before discussing them in detail, it is helpful to see their underlying similarities and differences by classifying them across several dimensions. Some techniques save dynamic energy by reducing the number of accesses to a particular level of cache hierarchy by using additional memory structures. These structures are used either for data storage [18–22], or *prediction* of cache access result [23–27] or for *pre-determination* of cache access result [28–33].