Improving Cache Energy Efficiency for Green Computing

Sparsh Mittal
Iowa State University, USA

INTRODUCTION

Green computing refers to the study and practice of designing, operating and disposing computing systems effectively in a manner which creates minimal or no impact on the environment. With increasing use of computing systems, their total energy consumption has also increased and it has been estimated that the carbon emission of ICT (information and communication technology) will triple from 2002 to 2020 (Smarr, 2010). Hence, techniques for improving the energy-efficiency of computing systems are extremely important to achieve the goals of green-computing and sustainability.

In nearly all the computer systems, power consumption presents itself as a primary design constraint. For example, in mobile and embedded computing systems, the amount of power consumed directly affects the battery lifetime. In desktop systems, excessive power dissipation has been one of the important reasons for the halt of clock frequency increases. To alleviate the problem of power dissipation, chip multiprocessors (CMPs) have been adopted since they allow high-performance computing within cost-effective power and thermal envelopes. In Internet datacenters and supercomputers, power consumption has been on rise. For example, each of the 10 most powerful supercomputers on the TOP500 List require up to 10 megawatts of peak power (Feng et al., 2007). This amount of power is sufficient to sustain a city of 40,000. The issue of power consumption drives major design decisions in modern companies and the increased power levels puts stress on the power transmission systems. Thus, high power consumption has significant socio-economic impacts.

Among different on-chip components, caches contribute a major fraction of chip-power consumption (Lahiri et al., 2004). In several processor chips, caches occupy more than 50% of the total area. Further, their size is increasing to fulfill the demands of performance-critical applications (Pande et al., 2009, Mittal et al., 2008, Khaitan et al., 2012). Also, the number of cores on a single chip is increasing; for example, IBM's POWER7, Intel's E7-8800 Series and AMD's Opteron 6000 Series processors use 8 to 16 cores on a single chip. To feed the large number of cores and to bridge the widening gap between the speed of processor core and DRAM memory, large sized caches are being used; as an example, Intel's E7-8800 processor uses 24MB L3 cache. Further, with each CMOS technology generation, leakage power has been dramatically increasing (Rodriguez et al. 2006). Hence, managing power consumption of caches is becoming a crucial issue in modern processor design (Mittal, 2014). In this chapter, we discuss the principles and techniques used for saving cache leakage energy. We also discuss the example of commercial chips which provide hardware functionality for saving cache energy.

BACKGROUND

Cache power can be divided into two categories, namely the dynamic power, which is dissipated due to transistor switching and the leakage power,

DOI: 10.4018/978-1-4666-5202-6.ch106

which is dissipated due to flow of leakage currents (Wang et al., 2013). In last level caches, leakage power forms a major source of power consumption and hence, we focus on the techniques used for saving cache leakage power.

In literature, several cache reconfiguration techniques have been used for saving cache energy. These techniques work on the principle that the cache requirements of different programs vary and hence, by allocating just the right amount of cache to an application, the rest of the cache can be turned off to save leakage energy. Thus, the cache reconfiguration based techniques change the active size of the cache to save energy.

The circuit-level leakage control schemes can be broadly divided into two categories, namely state-preserving and state-destroying schemes. These schemes enable switching the cache block to low-power mode. The difference between them lies in the fact that the state-preserving techniques retain the state of the cache block in the low-power mode (Flautner et al., 2001), while the state-destroying techniques do not maintain the contents of the block in the low-power mode (Powell et al., 2000).

Using these circuit-level schemes, several architectural techniques save cache energy. These techniques turn-off the cache at different granularity, such as cache-sets (called selective-sets Yang et al., 2001), cache-ways (called selective-ways, Sundararajan et al., 2012), hybrid (combination of selective-sets and selective-ways Mittal et al., 2012), cache blocks (Kaxiras et al., 2001, Flautner et al., 2001) and cache colors (Mittal et al., 2013).

To see the maximum number of options provided by different techniques, we take the example of an 8-way set-associative cache of 4MB size with 64B block size and assume page size of 4KB. Then, the number of cache blocks is 65,536; number of ways is 8; number of cache colors is 128. The number of sets is 8,192; however, note that the selective-sets approach only allocates cache at granularity of power of two sets, e.g. 8192, 4096, 2048 or 1024 sets etc. The hybrid approach (i.e. selective-sets and selective-ways) can provide number of combinations as (number of possible set allocations)* (number of possible way allocations). Assuming number of possible set allocations is 4 (i.e. 8192, 4096, 2048 or 1024 sets as in Mittal et al., 2012), the number of combinations provided by hybrid approach is 32 (=4*8).

MAIN FOCUS

Cache energy saving techniques are extremely important since they improve the energy efficiency of the chip. This, in turn, results in reduced requirements of cooling and simpler chip-design. Moreover, reduced power consumption enables increasing the clock frequency to further improve the performance of the applications.

Cache energy saving techniques work by reconfiguring the cache, while trying to keep the increase in miss-rate and performance loss small. In several schemes (e.g. selective-sets approach), cache reconfiguration changes the addressing of the cache blocks and hence, some of the existing cache blocks may be required to be written-back to memory. Thus, these techniques incur cache reconfiguration overhead. To minimize this overhead, cache reconfiguration is done at coarse-granularity of time interval, for example, after every 5M instructions or 5M cycles.

Many of the existing techniques (e.g. Yang et al., 2001) require offline (static) profiling of individual programs. For this reason, they cannot be easily used with modern servers, which run trillions of instructions of arbitrary combinations of multicore workloads. Moreover, the differences between the profiled runs and actual programs make the approach of per-application tuning highly ineffective and difficult-to-scale. To address this limitation, dynamic profiling based techniques are being used (e.g. Mittal et al., 2013), which can be easily used in product-systems.

Cache reconfiguration techniques can be either static or dynamic. The static reconfiguration techniques select a suitable configuration using offline profiling and use that configuration for the entire