IEICE TRANS. INF. & SYST., VOL.E96–D, NO.11 NOVEMBER 2013

LETTER

Region-Based Way-Partitioning on L1 Data Cache for Low Power

Zhong ZHENG[†a)], ***Student Member***, **Zhiying WANG**[†], ***and*** **Li SHEN**[†], ***Nonmembers***

SUMMARY Power consumption has become a critical factor for embedded systems, especially for battery powered ones. Caches in these systems consume a large portion of the whole chip power. Embedded systems usually adopt set-associative caches to get better performance. However, parallel accessed cache ways incur more energy dissipation. This paper proposed a region-based way-partitioning scheme to reduce cache way access, and without sacrificing performance, to reduce the cache power consumption. The stack accesses and non-stack accesses are isolated and redirected to different ways of the L1 data cache. Under way-partitioning, cache way accesses are reduced, as well as the memory reference interference. Experimental results show that the proposed approach could save around 27.5% of L1 data cache energy on average, without significant performance degradation.

key words: *cache, low power, region-based, way-partitioning*

1. Introduction

Caches have been the most effective approach in bridging the gap between the fast CPU and the slower main memory. Due to the temporal and spatial locality of programs, caches can provide fast access to the frequently accessed data. Typically, caches are organized as a cache line along with a corresponding tag. As a result, a cache access is comprised of tag checking and data fetching. Set-associative caches are widely adopted with higher hit-rate than direct-mapped caches, at the expense of more data reading and comparison. These extra tag comparisons and data accesses directly result in additional power consumption. For example, caches consume around 40% of whole processor power in StrongARM [1].

Many previous works have been done to address the cache power consumption issue. Way halting [2] approach uses a halt tag array to pre-determine which tags cannot match, to reduce cache activities. Way prediction [3] predicts one way that the current cache access would most likely to hit. By accessing the predicted way instead of all of the cache ways, way prediction could save much energy. However, the penalty of revisiting all the ways will be paid if the prediction turns out to be wrong. Direct addressed cache [4] leverages the compiler to identify the consecutive accesses that will access the same cache line and let the later memory reference directly access the cache line without tag check to save power. Region-based caching [5] provides multiple separated caches optimized for global, stack, and heap references, instead of a unified one. This type of caching can reduce power as small caches consume much lower power and can offer faster access. Cooperative partitioning [6] adopts way-partitioning on shared LLC (Last Level Cache) for energy efficiency on high-performance CMPs (Chip Multi-Processors).

In this paper, we combined the idea of region-based caching and way-partitioning for L1 data cache to reduce power consumption without causing significant performance degradation. Our proposal is based on the fact that small cache capacity could satisfy the requirement of stack accesses and less way accesses could save much cache power. As stack access occupies a large portion of the whole memory references, as much as 60% in Mibench [7], we redirect the stack accesses and non-stack accesses to different groups of L1 data cache ways. Thus, the ways that need to be read and checked are reduced and, consequently, the power consumption is reduced. The advantage of the proposed approach over the region-based caching is that the cache is still a unified one and can be configured with different partitioning scheme. In an extreme case, this cache still can be used as traditional cache if significant performance degradation is caused by way-partitioning.

The experimental results show that the proposed region-based way-partitioning scheme could reduce L1 cache data energy by 27.5% on average, without significant performance degradation.

2. Region-Based Way-Partitioning Scheme

2.1 Region-Based Cache Partitioning

Figure 1 shows the runtime memory subdivision of a program. There are several separated virtual memory address regions, namely the stack, heap, data, and code region. The stack region usually holds the local variables of the current program function, whose accesses are usually bounded within the current stack frame. Thus, the memory reference to the stack is more predictable than other regions, for example, the heap region. We name the data access to stack region and other regions as stack and non-stack access, respectively.

There has been region-based caching which designs separated small stack and global caches for the stack and global data, as shown in Fig. 2. Smaller cache can offer faster access and lower power consumption. By capturing

Manuscript received March 18, 2013.
Manuscript revised July 2, 2013.

[†]The authors are with the State Key Laboratory of High Performance Computing & School of Computer, National University of Defense Technology (NUDT), Changsha 410073, China.
a) E-mail: zheng_zhong@nudt.edu.cn

DOI: 10.1587/transinf.E96.D.2466

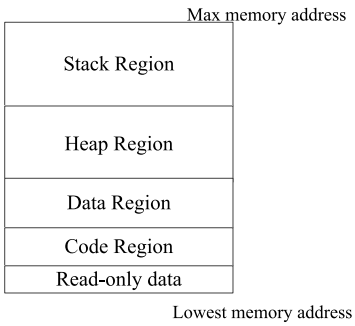

Fig. 1 Typical runtime memory subdivision.

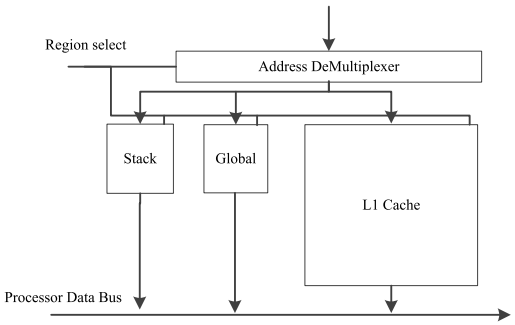

Fig. 2 Cache partitioning: separated caches for different regions.

most of the data access on smaller caches, this separated design can reduce 60% power consumption, on average, compared with traditional design [5].

2.2 Region-Based Way-Partitioning

2.2.1 Design Overview

We find that a physically separated cache design is not feasible for all programs, as different program have different stack and non-stack accesses, and the design cannot be changed once the processor has been produced. In contrast, a unified data cache could make better use of the on-die resources, as it can be configured to adapt to different programs. As a result, we propose a region-based way-partitioning scheme for L1 data cache, which only requires minor modification to the current cache design. Our proposal is orthogonal to other schemes on shared L2 cache, and can be combined with other L2 cache based schemes for CMPs.

It has been shown, by region-based caching, that a smaller cache could satisfy the stack access. For a set-associative cache, we could configure some ways dedicated for stack access, and the rest ways for non-stack access. As illustrated in Fig. 3, we configure one way for stack accesses in a four-way set-associative cache. Thus, the stack access only reads one cache tag and one data, and other memory references will access the rest three ways. By reducing the tag and data read, the corresponding dynamic power dissipation is reduced. In addition, the separated cache ways could potentially isolate the memory reference interference between the stack and non-stack accesses.

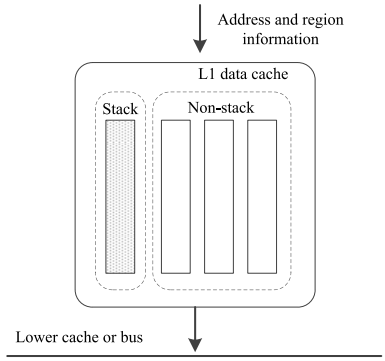

Fig. 3 Region-based way-partitioning overview.

2.2.2 Energy Modeling

In this section, we build a energy model to estimate the energy consumption of a n-way cache partitioning where n ways are configured for stack access and rest ways for non-stack access. Using E to denote the energy consumption, the energy of the cache, E_{cache}, can be obtained by:

$$E_{cache} = E_{static} + E_{dynamic} \tag{1}$$

where E_{static} and $E_{dynamic}$ are static and dynamic energy consumption, respectively.

Our approach tries to reduce the way access that directly results in dynamic energy consumption. We adopted the dynamic energy calculation from [2], where E_{dec}, E_{mux}, E_{tag}, E_{data}, E_{pre}, E_{com}, E_{SA}, E_{way} denote the energy dissipation of the address decoder, the mux and output driver, one tag array access, one data array access, one way's precharging, one way's comparator, one way's sense amplifier and one way in total, respectively. Then, the energy consumption of a N-way set-associative cache can be computed using Eq. (2).

$$\begin{aligned} E_{dynamic} &= E_{dec} + E_{mux} + N * E_{way} \\ &= E_{dec} + E_{mux} + N * (E_{tag} \\ &\quad + E_{data} + E_{pre} + E_{com} + E_{SA}) \end{aligned} \tag{2}$$

In our way-partitioning cache design, the E_{dec} and E_{mux} can not be avoided. Assuming that, in a program, P portion of the memory references locate in the stack region and n ($0 < n < N$) ways are configured for stack region. Thus, the dynamic energy dissipation under the region-based way-partitioning can be computed as follows:

$$\begin{aligned} E'_{dynamic} &= E_{dec} + E_{mux} + P * n * E_{way} \\ &\quad +(1 - P) * (N - n) * E_{way} \end{aligned} \tag{3}$$

The saved cache dynamic energy is:

$$\begin{aligned} E_{dyn\_save} &= E_{dynamic} - E'_{dynamic} \\ &= (n + (N - 2n) * P) * E_{way} \end{aligned} \tag{4}$$

where $(n + (N - 2n) * P)$ is the number of way access that

could be reduced per cache access under *n*-way partitioning.

3. Experiment and Results

3.1 Simulation Environment

To evaluate the performance and energy reduction of our proposal, we employed the widely used Gem5 [8] simulator. We use the timing simulation for CPU with SE (System-call Emulation) mode to prevent the memory access from Operating System.

In addition, we adopt CACTI 6.5 [9], under cache model, with the technology of 32nm, to estimate the energy consumption of region-based way-partitioning. The Gem5 simulator will offer the run time information of different benchmarks, while CACTI 6.5 provides the leakage power and access energy for each cache configuration. Combining these two, we can obtain the overall cache energy dissipation of different benchmarks.

The system configuration is detailed in Table 1. We use the traditional 4-way cache as the baseline, and we evaluate the 1-way partitioning and 2-way partitioning, where one and two ways out of four are configured for stack access. The benchmarks with different input sizes are denoted using {*bench name*}_{*input size*}. For instance, *basicmath* program with *small* and *large* input are denoted by *basic_small* and *basic_large*.

3.2 Performance

We have analyzed that this way-partitioning design on cache could save dynamic energy as a result of reducing cache way accesses. However, a power saving design should not cause much performance degradation. By running the benchmark on timing model, we get the execution time of the each program. The results are shown in Fig. 4. Generally, there is no significant performance difference between the proposed approach and the baseline platform. Interestingly, some programs even get some performance improvement. For example, the program *qsort_small* gets around 1.2% performance improvement under 1-way partitioning. The performance improvements are reasonable, as the separated ways could isolate the memory reference interference between the stack and non-stack accesses.

On the other hand, the *qsort_small* benchmark suffers from around 0.7% performance degradation under 2-way partitioning, which is insignificant. This performance degradation is mainly caused by the limited capacity of the separated cache ways.

Table 1 Experimental configuration.

System Parameter	Value
CPU	ARM ISA, one core, in-order
L1 I-Cache	32KB, 4-ways, 64B cahce line, 1 cycle
L1 D-Cache	32KB, 4-ways, 64B cache line, 1 cycle
L2 Cache	4 cycles
Memory Latency	32 cycles
Benchmark	Selected from Mibench with large and small input

3.3 Energy Dissipation

As descripted in Sect. 2.2.2, the saved dynamic energy could be calculated from Eq. (4). The dynamic energy of a cache is mainly caused by way access. The dynamic access energy of $E_{dynamic}$ and E_{way} obtained from CACTI are 0.0446 (nJ) and 0.0111 (nJ) in our 4-way cache, respectively. Thus, the energy of way access ($N * E_{way}$) in traditional cache acounts for about 99.5% of the total cache dynamic energy, according to Eq. (2). Then the percentage of dynamic energy saving will be almost the same as the percentage of reduced way access. We will first present the dynamic energy saving, and then the overall cache energy saving.

3.3.1 Cache Dynamic Energy Saving

The cache dynamic energy saving is shown in Fig. 5. The percentages of cache dynamic energy that can be saved under our way-partitioning approach vary among different benchmarks. These programs with higher portion of the stack access will enjoy more dynamic energy saving under 1-way partitioning, for example the *sha_small* and *sha_large*, as only one way is dedicated for stack access. In the 2-way partitioning, accesses to the stack and non-stack will always involve two-way data accesses and tag checks, just like a two-way cache. As a result, 50% accesses and checks are reduced among all of the benchmarks under 2-way partitioning, which results in around 49.8% of cache

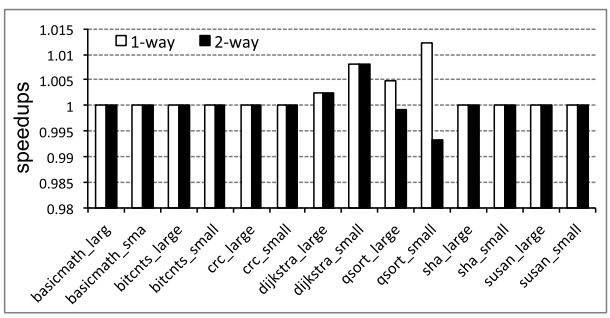

Fig. 4 Speedups under 1-way and 2-way partitioning.

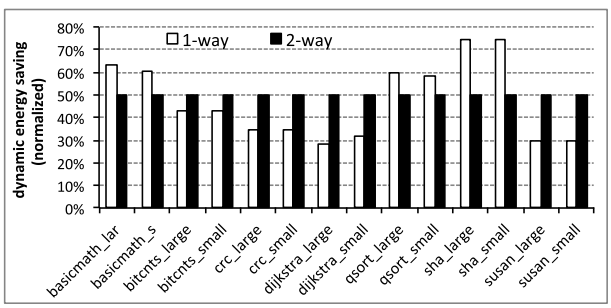

Fig. 5 Cache dynamic energy saving under 1-way and 2-way-partitioning, normalized to baseline.

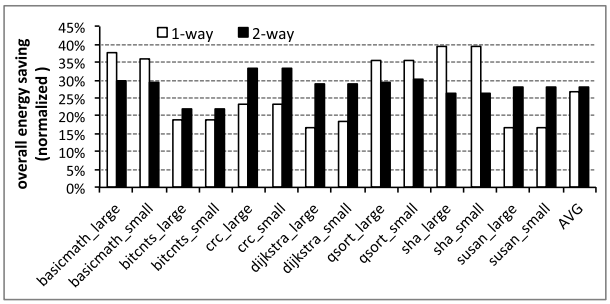

Fig. 6 Cache overall energy saving (including static and dynamic energy) under 1-way and 2-way partitioning, normalized to baseline.

dynamic energy saving.

3.3.2 Cache Overall Energy Saving

When considering the data cache overall energy saving, including static and dynamic energy, there are some differences against dynamic energy saving. As shown in Fig. 6, around 26.8% and 28.2% (27.5% for this two kinds of partitioning) energy are saved, on average, under 1-way and 2-way partitioning for different programs, respectively. The highest overall L1 data cache energy saving is around 39%, while the lowest is more than 16% across different programs. It's noticeable that, in the 2-way partitioning, the saved overall energy is different even the saved dynamic energy is the same, around 50%. The reason is that the dynamic energies account for different portion of the total cache energy for different programs, as the execution time (related to static energy) and number of data access (related to dynamic energy) differ.

According to Fig. 4 and Fig. 6, 1-way and 2-way cache partitioning could get different performance and energy saving on different programs. Maybe dynamically selecting the best partitioning could get the optimal results. However, we do not intend to make the CPU core and the cache system much more complicated.

4. Conclusion

Cache power accounts for a large portion of the whole chip power dissipation. In the paper, we proposed a region-based way-partitioning scheme, which redirects stack and non-stack access to different groups of ways in the set-associative L1 data cache. Power consumption is reduced since fewer cache ways are accessed to fetch data. Our experimental results based on gem5 simulator show that this proposed scheme could reduce the cache power around 27.5%, and without sacrificing performance. Our approach is orthogonal to the schemes on the shared L2 caches on CMPs and can be incorporated with other approaches for L2 caches.

Acknowledgements

This work is partially supported by China National 863 Program (No. 2012AA010905), the National Natural Science Foundation of China (No. 61070037, 61272144, 61103016, and 61202121), New Teachers' Fund for Doctor Stations (Ministry of Education, No.20114307120013), the Innovation Foundation for Excellent Postgraduate (No. B120607) from NUDT and Hunan Province.

References

[1] J. Montanaro, R.T. Witek, K. Anne, A.J. Black, E.M. Cooper, D.W. Dobberpuhl, P.M. Donahue, J. Eno, W. Hoeppner, D. Kruckemyer, et al., "A 160-mhz, 32-b, 0.5-w CMOS risc microprocessor," IEEE J. Solid-State Circuits, vol.31, no.11, pp.1703–1714, 1996.

[2] C. Zhang, F. Vahid, J. Yang, and W. Najjar, "A way-halting cache for low-energy high-performance systems," ACM Trans. Architecture and Code Optimization (TACO), vol.2, no.1, pp.34–54, 2005.

[3] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," Proc. 1999 International Symposium on Low Power Electronics and Design (ISLPED), pp.273–275, 1999.

[4] E. Witchel, S. Larsen, C.S. Ananian, and K. Asanović, "Direct addressed caches for reduced power consumption," Proc. 34th annual ACM/IEEE International Symposium on Microarchitecture (MICRO), pp.124–133, 2001.

[5] S.L. Hsien-hsin and G.S. Tyson, "Region-based caching: An energy-delay efficient memory architecture for embedded processors," Proc. 2000 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), 2000.

[6] K.T. Sundararajan, V. Porpodas, T.M. Jones, N.P. Topham, and B. Franke, "Cooperative partitioning: Energy-efficient cache partitioning for high-performance CMPS," IEEE 18th International Symposium on High Performance Computer Architecture (HPCA), 2012, pp.1–12, 2012.

[7] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," 2001 IEEE International Workshop on Workload Characterization, pp.3–14, 2001.

[8] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, et al., "The GEM5 simulator," ACM SIGARCH Computer Architecture News, vol.39, no.2, pp.1–7, 2011.

[9] N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi, "Cacti 6.0: A tool to model large caches," HP Laboratories, 2009.