ICICS-PCM 2003
15-18 December 2003
Singapore

Smart Cache: An Energy-Efficient D-Cache for a Software MPEG-2 Video Decoder

Chia-Lin Yang, Hung-Wei Tseng, Chia-Chiang Ho
Department of Computer Science and Information Engineering,
National Taiwan University,
Taipei 106, Taiwan

ABSTRACT

Power consumption is an important design issue of current embedded systems. Data caches consume a significant portion of total processor power for data intensive applications. In this paper, we propose to utilize application-specific information for cache resource allocation to achieve energy saving, including cache bypassing, the mini-cache and way-partition. We use a software MPEG-2 video decoder as our first targeted application. Cache bypassing excludes data types that have little reuse from the L1 cache. The mini-cache stores data types with high access frequency and small memory footprints to a small on-chip memory area. The way-partition mechanism maps program data structures to different ways of caches and enables only the matching ways on each access. The results show up to 40% of cache energy reduction without sacrificing performance.

1. Introduction

Power consumption is becoming a critical design issue of embedded system due to the popularity of portable devices such as cellular phones and personal digital assistants. Caches consume a significant portion of the total processor power. For example, 42% of processor power is dissipated in the cache subsystem in StrongARM 110 [1]. Many embedded applications, in both the multimedia and communication domains, are data dominated. Data storage and transfer account for a significant portion of overall power consumption. Whether a reference goes to the main memory or not, it must access the data cache.

Cache partitioning and way-prediction are two commonly used techniques to reduce energy dissipation in data caches. Cache partitioning schemes divide caches into smaller components since a smaller cache has a lower load capacitance. Way-prediction predicts the matching way and probes only the matching way instead of all ways to reduce power consumption for set-associative caches. These techniques often increase average access latency if the referenced data is not located in the predicted region.

The need for prediction is due to the fact that cache management is transparent to software. If we allow software to control cache resource allocation, we can access the region where a memory reference is located directly. In this way, we can achieve energy saving without increasing average cache access time. Allowing software to control caches has been proposed to improve cache performance for embedded systems [2][3]. In this paper, we exploit the potential of using a software-managed cache for energy optimization.

We use a software MPEG-2 video decoder as our targeted application. An MPEG-2 decoder has large data set and requires high data processing rate, which are two important characteristics of real-time signal processing applications. We consider three software-controlled cache management mechanisms, cache-bypassing, mini cache, and way-partition, and demonstrate how to utilize the application-specific information of an MPEG-2 decoder to achieve energy saving.

Our results show that bypassing the MPEG-2 video output can achieve cache energy reduction of 1.4%. Storing the most frequently accessed data type to a 512B mini-cache can reduce 21% of cache energy. Way-partition can even achieve higher energy saving – up to 40%. The cache bypassing and mini-cache schemes offer slight performance advantage. Way-partition incurs performance degradation but it is within 1%.

The rest of the paper is organized as follows. Section 2 provides background information on a MPEG2 software decoder. Section 3 describes how we utilize application-specific information of a software MPEG-2 decoder to manage cache resources for energy optimization. Section 4 describes our experimental methodology and Section 5 presents the results. Section 6 concludes this paper.

2. MPEG-2 Overview

MPEG-2 was released in 1994 to provide coding algorithms for video and associated audio targeting at bit rate between 2 and 10 Mbps. It has been widely used in applications such as digital cable TV service, network database video service and DVD playback. In this section, we briefly describe the decoding process and main data types in a software MPEG-2 decoder.

0-7803-8185-8/03/$17.00 © 2003 IEEE

2.1 MPEG-2 Architecture

A MPEG-2 bitstream consists of three different frame types: I-frame, P-frame and B-frame. Each frame is partitioned into macroblocks and each macroblock is separately encoded. Macroblocks in I-frames are intra-coded only (without temporal prediction); macroblocks in P and B frames are inter-coded which exploits the temporal redundancy of reference frames. For inter-coded macroblocks, motion estimation is performed to find the best motion vector and coding mode. Image data of intra-coded macroblocks and motion compensated error of inter-coded macroblocks go through DCT, quantization, and run-length entropy coding to become a compressed bitstream. The decoding of a compressed MPEG-2 bitstream, the application our work focuses on, is the inverse of the above encoding process, but without motion estimation.

2.2. Data Set Composition

We can break down mpeg-2 decoder data types into the following classes:

Input— The MPEG-2 bitstreams. Bitstreams coming from network or storage are temporarily stored in a fixed size buffer which is refreshed when the stored data has been decoded.
Output— The decoded picture data. The output data is write-only and transferred to the frame buffer through system calls.
Tabular— Static and read-only tables used in the decoder, such as lookup tables for variable length decoding.
Reference— Buffers for both current and reference frames.
Block—The buffer for pixel values of a single macroblock. The inverse quantization and inverse DCT are performed in-place in this buffer.
State—Variables needed for setting and operation of the decoder. Note that we do not include those local variables for temporary usage in various decoding modules/functions.

Table 1 lists the data set size and percentage of total memory references for different data types. Note that the access percentage from the major data types only adds up to 82%. A significant portion of the remaining references comes from accessing the stack region (12% of total memory references).

3. Energy-Aware Cache Management

3.1 Energy Dissipation in Caches

Caches are composed of CMOS static RAM. The dominant energy dissipation in CMOS circuit comes from dynamic switching dissipation, i.e., transitions that charge or precharge the load capacitance. Dynamic switching power dissipation P can be described as:

Data type	size	Access%	Data type	size	Access%
Input	2KB	0.2%	reference	1500KB	27.5%
Output	500KB	1.9%	block	1.5KB	34.2%
Tabular	5KB	9.2%	state	1.5KB	9.1%

Table 1: Summary of decoder data types, size and % of memory references

$$P = a_{0\to 1} C V^2 f$$

where C is the capacitance, V is the supply voltage, f is the clock frequency and $a_{0\to 1}$ represents the number of access transitions. Power can be reduced by reducing the supply voltage/clock frequency, load capacitance, or switching frequency generally. Reducing voltage/clock frequency often sacrifices performance. Therefore, for cache energy optimization, we should aim at reducing the cache capacitance and switching frequency.

3.2 Cache Energy Optimization Techniques

In this section, we describe how we utilize application-specific information of a MPEG-2 decoder for cache energy optimization.

Cache Bypassing

Bypassing memory references that have little reuse can improve L1 cache performance because it results in less cache pollution [4]. Accessing the lower level of the memory hierarchy directly for memory references that always miss in the L1 cache can reduce the power consumption of L1 cache (i.e. reducing L1 cache switching frequency) without increasing that of other memory components. The video output data of the MPEG-2 decoder is an ideal data type for bypassing since the output stream is written and never read by the CPU.

Instruction annotation techniques can be used to implement the bypassing mechanism. Several commodity processors have already provided different flavors of load instruction to allow more sophisticated cache management policy. For example, the ULTRA SPARC processor provides a *block_load* instruction that loads several floating-point registers while bypassing the first level cache [5].

The mini-cache

Several studies have proposed to add a cache that is very small relative to the conventional L1 cache on chip for power optimization since a smaller cache has lower load capacitance [6][7][8]. The filter cache proposes to place this mini-cache between CPU and L1 cache as shown in Figure 1(a) [6]. This approach can reduce the power

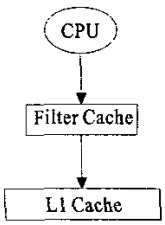


Figure 1(a): Filter cache

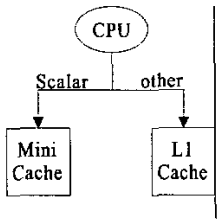


Figure 1(b): Mini-cache

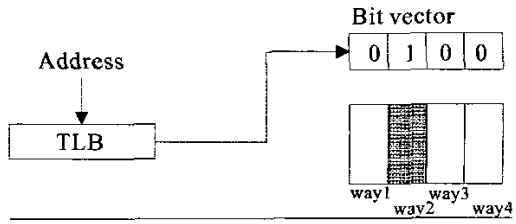


Figure 2: Modified Column caching

consumption by 58 % but degrade performance by 21%. The minimax cache [8] proposes to use special load/store instruction to direct scalar data to the mini-cache as shown in Figure 1(b). In this paper, we look at a similar architecture but map data types with small memory footprint and high access frequency to the mini-cache. As indicated in Table 1, the block data type has highest access frequency and only 1.5KB data set size. Therefore, it is an ideal target for storing in the mini-cache.

Way-Partition

Modern microprocessors often employ set-associative caches to achieve low miss rate. The tag and data arrays of all the ways are probed in parallel, and then the data is selected from the matching way. A large part of energy is actually wasted since only the data of the matching way is used. Powell et al. [9] proposes to predict the matching way for energy saving. However, this approach often increases average cache access time due to mis-prediction and still needs to access all the ways of the tag array. In this paper, we propose to statically map particular data types to specific ways of an n-way set-associative cache according to their reference patterns (i.e., working set size and access frequency). Therefore, on each cache access, we can pinpoint the matching way.

To implement this idea we can adopt column-caching proposed by Chiou et al.[2]. Column-caching allows software to map program data structures to different ways of caches. The mapping information is represented by a bit vector, which guides the replacement when a cache miss occurs. The bit vector is stored in the TLB (translation-look-aside-buffer) since the TLB is accessed at every memory reference for address translation. The similar framework can be adopted for energy saving with small modification as shown in Figure 2. On each access, the bit vector is checked to determine the mapping ways and only the mapping ways are accessed.

4 Experimental Methodology

We use Wattch toolset [15] developed at Princeton University to conduct our experiments. Wattch generates both the performance and energy data through execution-driven simulation. We modified Wattch to simulate the energy-saving techniques proposed in this paper. Our baseline machine model is an ARM-like

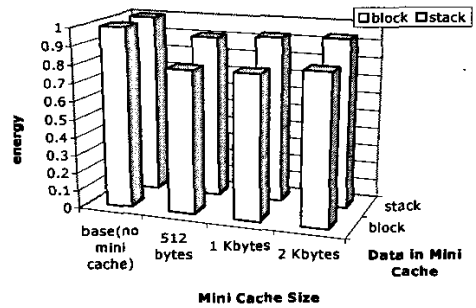


Figure 3: The mini-cache energy consumption

single-issue in-order processor which contains an 8K, 4-way associative cache. We select the cache size based on the SA-1110 design [14]. Since DRAM memory power is not modeled in the Wattch toolset, a four-way 512KB L2 cache is used as a backing storage[1]. All the caches are single-ported. We evaluate energy consumption assuming 0.35um process technology and activity sensitive conditional clocking.

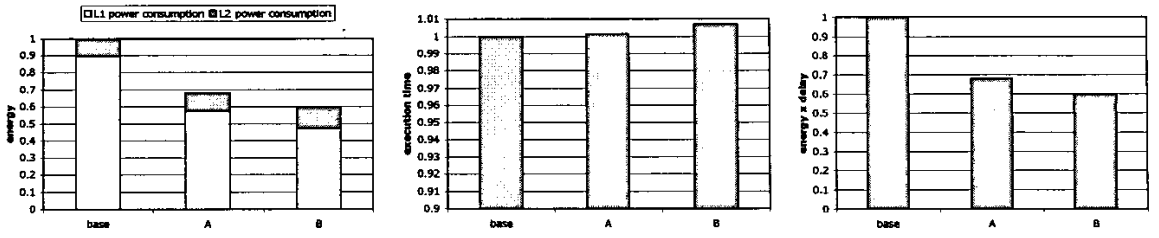
We obtain the MPEG-2 software decoder from the MPEG Software Simulation Group [16]. For input sensitivity study, we test three video sequences with a resolution of 704x480 pixels at 30 frames/s and a compressed bitrate of 6 Mbits/s. We found little variation among different video sequences regarding their memory behavior (i.e., access frequency of different data types and cache miss ratios). Therefore, we obtained the results in the next section using one test stream and restricted each run to 15 frames to limit the simulation time. [2]

5 Results

Since energy-saving methods may reduce energy dissipation at the expense of performance degradation, we evaluate the performance, energy consumption and energy-delay product of the proposed energy-saving techniques. Energy-delay product is often used as the metric for an energy efficient design since it considers both

[1] The L2 cache miss ratio is close to zero for an MPEG-2 decoder.

[2] We observed that decoding a 15 frames has the similar memory behavior as decoding a complete sequence.



(a) Normalized energy consumption (b) Normalized execution time (c) Normalized energy-delay

Figure 4: The effect of way-partition.
A: block (1 way);others (3 ways) B: block+state (1 way); tabular+stack (1 way);others (2 ways)

performance and energy at the same time. Note that the proposed energy-saving techniques may increase L1 cache miss rate, thereby increasing the L2 cache energy consumption. Therefore, for a fair comparison, we consider both the L1 and L2 caches for energy evaluation.

5.1 Cache Bypassing

As mentioned previously, the video output data of the MPEG-2 decoder is an ideal data type for bypassing since the output stream is written and never read by the CPU. The experimental results show that excluding video output data from the L1 cache can reduce the energy consumption by 1.4%. It also offers slight performance improvement. We do not see significant energy saving because the output data accounts for only 2% of total memory references. Since bypassing the output data does not cause performance degradation, for the results presented below, we bypass the output data in all configurations.

5.2 The mini-cache

Here we evaluate the energy efficiency of storing the block data type of an MPEG-2 decoder to the mini-cache. Several studies propose to have a separate partition for the stack data references [7][12]. The stack data references in an MPEG-2 decoder accounts for 12% of total memory references and has a relative small memory footprint (0.5K). Therefore, we compare the energy efficiency of mapping the block data type to the mini-cache to that of stack memory references. Figure 3 shows the normalized energy consumption. The mini-cache is direct-mapped. We test three mini- cache sizes: 512B, 1K and 2K. The results show that a 512B mini-cache is sufficient to keep most of the working set of the block data type and stack references (close to zero miss rate). Mapping the block data type to the mini-cache achieves higher energy saving compared to the stack memory references (21% vs. 10%). On the performance side, the addition of a mini-cache only offers slight performance improvement because the baseline model has already very low L1 cache miss rate (1.02%).

5.3 Way-Partition

The cache resource allocation strategy used in the way-partition mechanism is to give frequently accessed data types priority and allocate resources close to their working set sizes[3]. Note that the working set size of a data type is typically smaller than its data set size. We consider two partitioning schemes. The first scheme reserves one way of the L1 cache for the block data type and maps other data types to the remaining three ways. The second scheme aggressively partitions data into three groups: block+state (1 way), tabular+stack (1 way) and others (2 ways). A finer partitioning saves more energy of the L1 cache since each access consumes less power but it could cause more capacity and conflict misses. Therefore, the tradeoff between performance and energy saving needs to be carefully evaluated.

Figure 4(a) shows the normalized energy consumption of these two partitioning schemes. We divide the energy consumption into the L1 and L2 components. The results show significant energy saving. The first scheme reduces the energy consumption by 31.8% and the second by 40.4%. Both partitioning schemes have only little effect on the L2 power consumption. That implies insignificant performance impact. Figure 4(b) shows the normalized execution time. Even though both schemes these cause performance degradation but it is within 1%. The normalized energy-delay product is shown in Figure 4(c). The second partitioning scheme reduces the energy-delay product by 40%.

6. Related work

Several studies have proposed to partition cache for power optimization. The filter cache [6] extends this idea further by using a larger buffer (smaller than the L1 cache)

[3] We define working set size as the smallest cache size required to obtain a specific miss ratio.

to store recently accessed blocks. Region-based caching [7] reduces cache energy dissipation by partitioning the L1 data cache into three components: stack, global and heap. Data accesses are directed to different component based on their memory region types. The minimax cache [8] looks at the effect of storing scalar data types in a mini-cache for a set of multimedia applications. To reduce the energy dissipation of set-associative caches specifically, Powell et al. [9] propose to predict the way that contains the accessed data and probe only the predicted way to save energy. Albonesi et al. [10] statically turn off unneeded ways in a set-associative cache (selective-ways) while Yang et al. suggest a dynamic approach to disable unused sets (selective-sets) [11]. Several studies have proposed to use software-managed cache for performance optimization. Panda et al. [13] map scalar data to the scratch-pad memory for embedded applications. Chiou et al. [2] provide a flexible method to partition cache regions dynamically. Soderquist et al. [3] suggest to exploit application-specific information for better cache resource management to improve cache efficiency for an MPEG-2 decoder. This paper is the first attempt to use software-managed cache for energy optimization.

7. Conclusions

In this paper, we propose to use a software-managed cache for energy optimization for a software MPEG-2 video decoder. We evaluate three energy-reduction techniques. First, we exclude the video output data from the L1 cache since it is write-only and obtain 1.4% energy saving. The second mechanism stores the block data type, which has high access frequency and a small memory footprint, into the mini-cache. The results show that using a 512B mini-cache can reduce energy consumption by 21%. The third mechanism maps specific data types to different ways of the data cache and on each access, only the mapping ways are accessed. This mechanism can achieve up to 40% energy saving. None of the techniques causes significant performance degradation (less than 1%).

This study has shown the potential of using a software-management cache for energy reduction. In future work, we plan to investigate compiler techniques for automatic cache resource allocation.

Acknowledgments

This work is supported in part by the National Science Council under Grant NSC 92-2213-E-002-014- and NSC 91-2215-E-002-043- and research fundings from Microsoft. We would like to thank the anonymous reviewers for their valuable comments.

References

[1] J. Montanaro, et al. A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor. *IEEE Journal of Solid State Circuits*, 31(11):1703-1714, November 1996

[2] D. Chiou, P. Jain, L. Rudolph and S. Devadas. Application-Specific Memory Management for Embedded Systems Using Software-Controlled Cache. In *Proceedings of DAC*, 2000. Los Angeles, California

[3] P. Soderquist and M. Leeser. Memory Traffic and Data Cache Behavior of an MPEG-2 Software Decoder. In *Proceedings of International Conference on Computer Design*, 1997

[4] T. L. Johnson, D. A. Connors, M. C. Merten, and W. W. Hwu. Run-Time Cache Bypassing. *IEEE Transactions on Computers*, Vol. 48, No. 12, December 1999, pp. 1338-1354

[5] B. Case. SPARC V9 Adds Wealth of New Features. *Microprocessor Report*, 7 (9), February 1993

[6] J. Kin, M. Gupta, W. H. Mangione-Simith. The Filter Cache: An Energy Efficient Memory Structure. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, December, 1997

[7] H.-H. Lee and G. S. Tyson. Region-Based Caching: An Energy-Delay Efficient Memory Architecture for Embedded Processors. In *Proceedings of International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES 2000)*, Nov. 2000.

[8] O. S. Unsal, I. Koren, C. M. Krishna and C. A. Mortiz. The Minimax Cache: An Energy-Efficent Framework for Media Processors. In *Proceedings of 8th International Conference on High Performance Computer*, Febuary 2002

[9] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi and K. Roy. Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping. In *Proceedings of 34th Intel Symposium on Microarchitecture*, 2001

[10] David H. Albonesi. Selective Cache Ways: On-Demand Cache Resource Allocation. *Journal of Instruction-Level Parallelism*, 2000

[11] S.-H Yang, M. D. Powell, B. Falsafi, and T. N. Vijaykumar. Exploiting Choices in Resizable Cache Design to Optimize Deep-Submicron Processor Energy-Delay. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, November 2001

[12] M. Huang, R. Reanu and J. Torellas. L1 Cache Decomposition for Energy Efficient Processors. In Proceedings of Internatinal Symposium on Low-Power Electronics and Design (ISPLED'01), Huntington Beach, CA, August 2001.

[13] P. R. Panda, N. D. Dutt and A. Nicolau. Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications. In *Proceeding of European Design & Test Conference*, 1997

[14] Intel StrongARM SA-1110 Microprocessor Brief Datasheet, April 2000

[15] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*, Vancouver, British Columbia, June 2000.

[16] S. Echart and C. Fogg. ISO/IEC MPEG-2 Software Video Codec. In *Proceeding of the SPIE conference on Digital Video Compression: Algorithms and Technologies*, Vol. 2419, 7-10 February 1995, San Jose, California, pp. 100-109.