Reducing Power Consumption for High-Associativity Data Caches in Embedded Processors

Dan Nicolaescu Alex Veidenbaum Alex Nicolau

Dept. of Information and Computer Science
University of California at Irvine
{dann, alexv, nicolau}@cecs.uci.edu

Abstract

Modern embedded processors use data caches with higher and higher degrees of associativity in order to increase performance. A set–associative data cache consumes a significant fraction of the total power budget in such embedded processors. This paper describes a technique for reducing the D–cache power consumption and shows its impact on power and performance of an embedded processor. The technique utilizes cache line address locality to determine (rather than predict) the cache way prior to the cache access. It thus allows only the desired way to be accessed for both tags and data. The proposed mechanism is shown to reduce the average L1 data cache power consumption when running the MiBench embedded benchmark suite for 8, 16 and 32–way set–associate caches by, respectively, an average of 66%, 72% and 76%. The absolute power savings from this technique increase significantly with associativity. The design has no impact on performance and, given that it does not have mis-prediction penalties, it does not introduce any new non-deterministic behavior in program execution.

1. Introduction

A data caches is an important component of a modern embedded processor, indispensable for achieving high performance. Until recently most embedded processors did not have a cache or had direct–mapped caches, but today there's a growing trend to increase the level of associativity in order to further improve the system performance. For example, Transmeta's Crusoe [7] and Motorola's MPC7450 [8] have 8–way set associative caches and Intel's XScale has 32–way set associative caches.

Unfortunately the power consumption of set–associative caches adds to an already tight power budget of an embedded processor.

In a set–associative cache the data store access is started at the same time as the tag store access. When a cache access is initiated the way containing the requested cache line is not known. Thus all the cache ways in a set are accessed in parallel. The parallel lookup is an inherently inefficient mechanism from the point of view of energy consumption, but very important for not increasing the cache latency. The energy consumption per cache access grows with the increase in associativity.

Several approaches, both hardware and software, have been proposed to reduce the energy consumption of set–associative caches.

A phased cache [4] avoids the associative lookup to the data store by first accessing the tag store and only accessing the desired way after the tag access completes and returns the correct way for the data store. This technique has the undesirable consequence of increasing the cache access latency and has a significant impact on performance.

A way–prediction scheme [4] uses a predictor with an entry for each set in the cache. A predictor entry stores the most recently used way for the cache set and only the predicted way is accessed. In case of an incorrect prediction the access is replayed, accessing all the cache ways in parallel and resulting in additional energy consumption, extra latency and increased complexity of the instruction issue logic. Also, given the size of this predictor, it is likely to increase the cache latency even for correct predictions.

Way prediction for I-caches was described in [6] and [11], but these mechanisms are not as applicable to D-caches.

A mixed hardware–software approach was presented in [12]. Tag checks are avoided by having the compiler output special load/store instructions that use the tags from a previous load. This approach changes the compiler, the ISA and adds extra hardware.

This paper proposes a mechanism that *determines*, rather than predicts, the cache way containing the desired data before starting an access (called *way determination* from now on). Knowing the way allows the cache controller to only

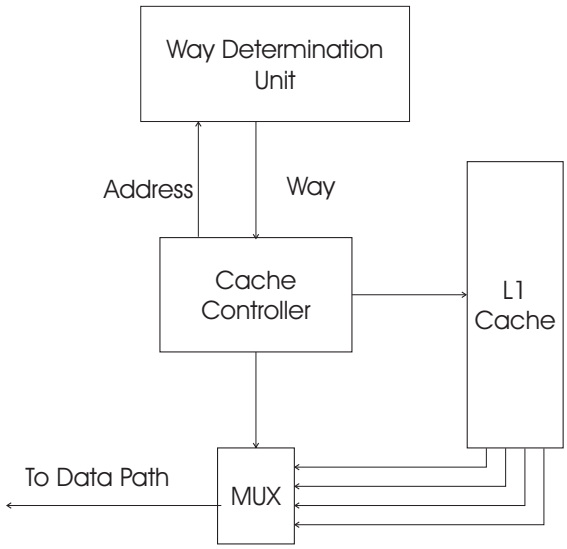

Figure 1. Cache hierarchy with way determination

access one cache way, thus saving energy. The approach is based on the observation that cache line address locality is high. That is, a line address issued to the cache is very likely to have been issued in the recent past. This locality can be exploited by a device that stores cache line address/way pairs and is accessed prior to cache access. This paper shows that such a device can be implemented effectively.

2. Way Determination

Way determination can be performed by a Way Determination Unit (*WDU*) that exploits the high line address locality. The WDU is accessed prior to cache access and supplies the way number to use as shown in Figure 1.

The WDU records previously seen cache line addresses and their way number. An address issued to the cache is first sent to WDU. If the WDU contains an entry with this address the determination is made and only the supplied cache way is accessed for both tags and data. Since the address was previously seen it is not a prediction and is always correct. If the WDU does not have the information for the requested address, a cache lookup is performed with all the ways accessed in parallel. The missing address is added to the WDU and the cache controller supplied way number is recorded for it.

Because of the high data cache address locality the number of entries in the WDU can be small, thus allowing fast access and low energy overhead. WDU lookup can be done in parallel with load/store queues access as it has about the same latency. This should add no extra latency to the cache access.

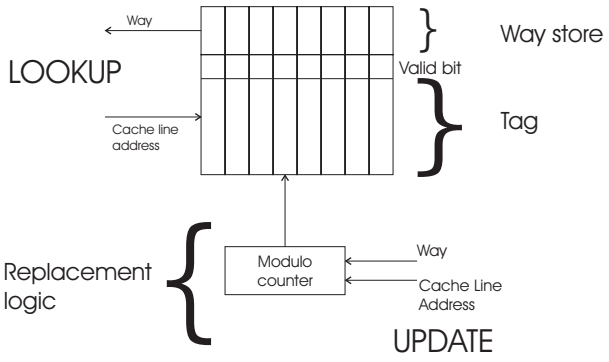

Figure 2. Way determination unit

The way determination system proposed here is based on access history. It has an energy penalty on misses similar to the mis-prediction penalty in a predictor. But it doesn't have the performance penalty of a mis-prediction.

3. The Way Determination Unit design

The WDU, as shown in Figure 2, is a small, cache–like structure. Each entry is an address/way pair plus a valid bit. The tag part is fully associative and is accessed by a cache line address. The address is compared to a tag on lookup to guarantee the correct way number.

There are two types of WDU access: lookup and update. The lookup is cache-like: given a cache line address as input, the WDU returns a way number for a matching tag if the entry is valid. Updates happen on D–cache misses or WDU miss and cache hit. On a miss the WDU entry is immediately allocated and its way number is recorded when the cache controller determines it. If there are no free entries in the WDU the new entry replaces the oldest entry in the WDU. Our initial implementation uses a FIFO entry pointed to by the modulo counter.

One other issue the WDU design needs to address is coherence: when a line is replaced or invalidated in the L1 cache the WDU needs to be checked for the matching entry. The WDU entry can be made invalid. Another possible approach is to allow the access to proceed using the WDU–supplied way and a cache miss to occur when the cache tag access is performed. The way accessed was the only place the line could have been found. The WDU can be updated when a line is allocated again. This is the approach used in the design presented here.

4. Experimental Setup

To evaluate the WDU design, the Wattch version 1.02 simulator [1] was augmented with a model for the WDU.

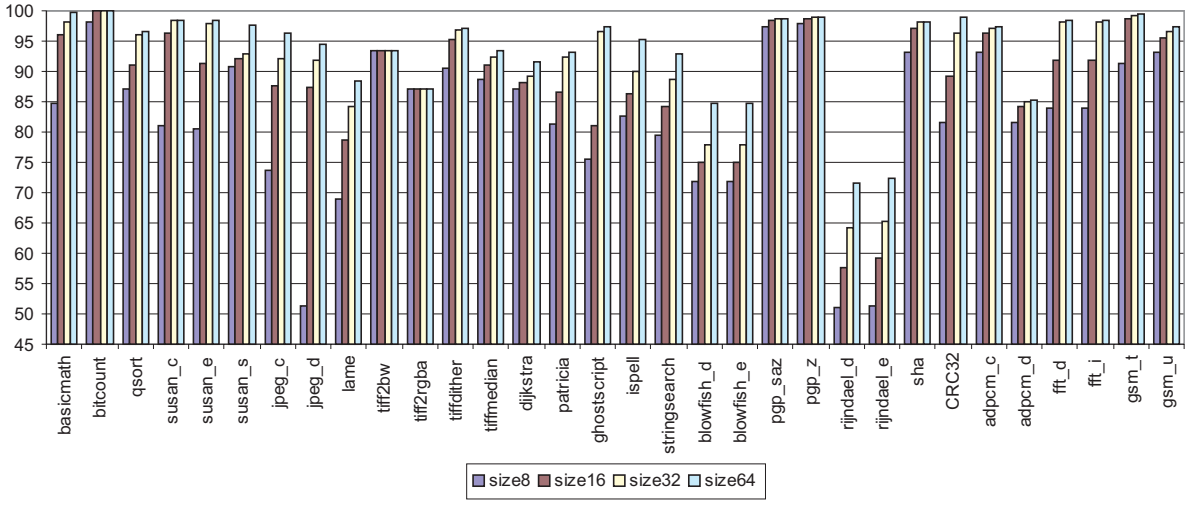

Figure 3. Percentage load/store instructions "covered" by way determination for a 32–way set associative cache

Based on SimpleScalar [2], Wattch is a simulator for a superscalar processor that can do detailed power simulations. The CMOS process parameters for the simulated architecture are 400MHz clock and $.18\mu$m feature size.

The processor modeled uses a memory and cache organization based on XScale [5]: 32KB data and instruction L1 caches with 32 byte lines and 1 cycles latency, no L2 cache, 50 cycle main memory access latency. The machine is in-order, it has a load/store queue with 32 entries. The machine is 2–issue, it has one of each of the following units: integer unit, floating point unit and multiplication/division unit, all with 1 cycle latency. The branch predictor is bimodal and has 128 entries. The instruction and data TLBs are fully associative and have 32 entries.

4.1. The WDU power model

The WDU tags and way storage are modeled using a Wattch model for a fully associative cache. The processor modeled is 32bit and has a virtually indexed L1 data cache with 32 byte lines, so the WDU tags are $32 - 5 = 27$ bits wide, and the data store is 1, 2, 3, 4 or 5 bits wide for a 2, 4, 8 or 32–way set associative L1, respectively. The power consumption of the modulo counter is insignificant compared to the rest of the WDU. The power model takes into account the power consumed by the different units when idle.

For a processor with a physically tagged cache the size of the WDU is substantially smaller and so would be the power consumption of a WDU for such an architecture.

Cacti3 [10] has been used to model and check the timing parameters of the WDU in the desired technology.

4.2. Benchmarks

MiBench[3] is a publicly available benchmark suite designed to be representative for several embedded system domains. The benchmarks are divided in six categories targeting different parts of the embedded systems market. The suites are: Automotive and Industrial Control (basicmath, bitcount, susan (edges, corners and smoothing)), Consumer Devices (jpeg encode and decode, lame, tiff2bw, tiff2rgba, tiffdither, tiffmedian, typeset), Office Automation (ghostscript, ispell, stringsearch), Networking (dijkstra, patricia), Security (blowfish encode and decode, pgp sign and verify, rijndael encode and decode, sha) and Telecommunications (CRC32, FFT direct and inverse, adpcm encode and decode, gsm encode and decode).

All the benchmarks were compiled with the -O3 compiler flag and were simulated to completion using the "large" input set. Various cache associativities and WDU sizes have been simulated, all the other processor parameters where kept constant during this exploration.

5. Performance Evaluation

Figure 3 shows the percentage of load/store instructions for which a 8, 16, 32 or 64–entry WDU can determine the correct cache way. An 8–entry WDU can determine the way for between 51 and 98% of the load/store instructions, with an average of 82%. With few exceptions (susan_s, tiff2bw,

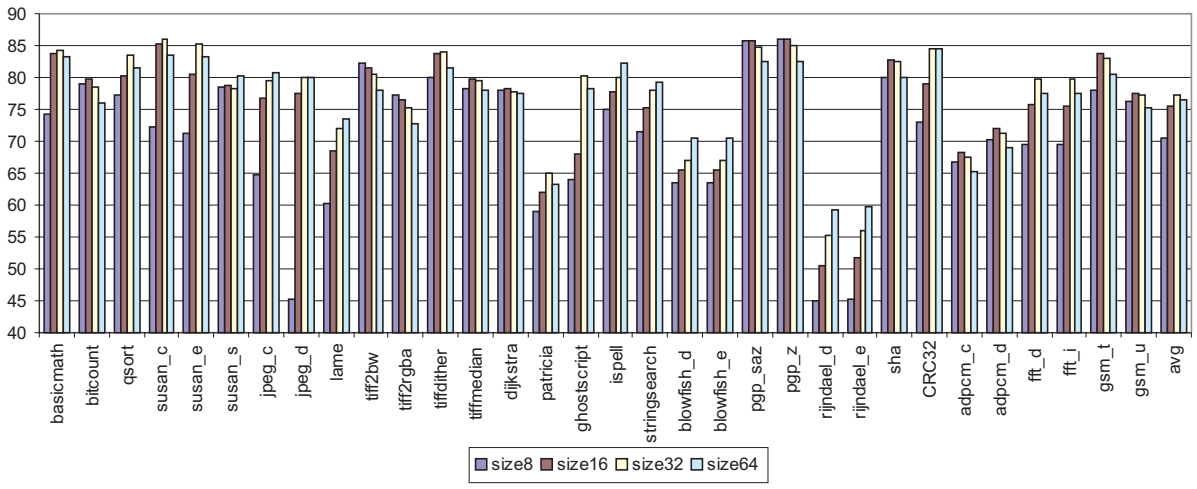

Figure 4. Percentage data cache power reduction for a 32–way set associative cache using different WDU sizes

tiff2rgba, pgp, adpcm, gsm_u) for the majority of benchmarks increasing the WDU size to 16 results in a significant improvement in the number of instructions with way determination. The increase from 16 to 32 entries only improves the performance for a few benchmarks, and the increase from 32 to 64 for even fewer benchmarks.

Figure 4 shows the percentage data cache power savings for a 32–way cache when using an 8, 16, 32 or 64–entry WDUs. For space and legibility reasons all other results will only show averages, the complete set of results can be found in [9].

A summary of the average number of instructions for which way determination worked for 2, 4, 8, 16 and 32–way set–associative L1 data cache and 8, 16, 32 and 64–entry WDU is presented in Figure 5. It is remarkable that the WDU detects a similar number of instructions independent of the L1 cache associativity. Increasing the WDU size from 8 to 16 produces the highest increase in the percentage of instructions with way determination, from 82% to 88%. The corresponding values for a 32 and 64-entry WDU are 91% and 93%.

Figure 6 shows the average data cache power savings for the MiBench benchmark suite due to using the WDU, compared to a system that does not have a WDU. When computing the power savings the WDU power consumption is added to the D–cache power. For all the associativities studied the 16–entry WDU has the best implementation cost / power savings ratio. It's average D–cache power savings of 36%, 56%, 66%, 72% and 76% for, respectively, a 2, 4, 8, 16 and 32–way set associative cache are within 1% of the power savings of a 32–entry WDU for a given associativ-

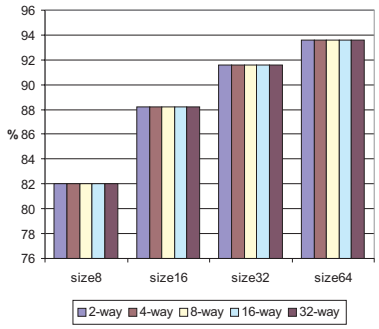

Figure 5. Average percentage load/store instructions "covered" by way determination

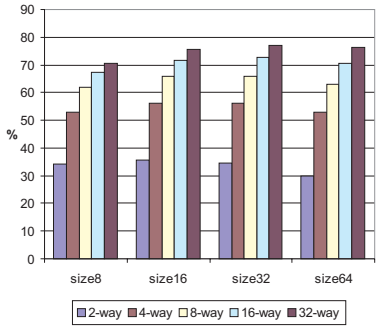

Figure 6. Average percentage D–cache power reduction

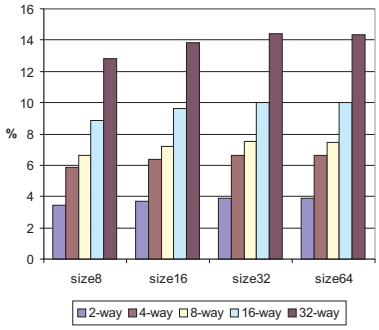

Figure 7. Total processor power reduction

ity. The even smaller 8–entry WDU is within at most 3% of the best case. For the 64–entry WDU the WDU power consumption overhead becomes higher than the additional power savings due to the increased number of WDU entries, so the 64–entry WDU performs worse than the 32–entry one for a given associativity.

Figure 7 shows the percentage of total processor power reduction when using a WDU. For a 16–entry WDU the power savings are 3.73%, 6.37%, 7.21%, 9.59% and 13.86% for, respectively a 2, 4, 8, 16 and 32-way set associative L1 data cache. The total processor power savings are greater for higher levels of associativity due to the increased D–cache power savings and to the increased share of the D–cache power in the total power budget.

6. Conclusions

This paper addresses the problem of the increased power consumption of associative data caches in modern embedded processors. A design for a Way Determination Unit (WDU) that reduces the D-cache power consumption by allowing the cache controller to only access one cache way for a load/store operation was presented. Reducing the number of way accesses greatly reduces the power consumption of the data cache.

Unlike previous work, our design is not a predictor. It does not incur mis-prediction penalties and it does not require changes in the ISA or in the compiler. Not having mis-predictions is an important feature for an embedded system designer, as the WDU does not introduce any new non–deterministic behavior in program execution. The power consumption reduction is achieved with no performance penalty and it grows with the increase in the associativity of the cache.

The WDU components, a small fully associative cache and a modulo counter, are well understood, simple devices that can be easily synthesized. It was shown that very a small (8-16 entries) WDU adds very little to the design gate count, but can still provide significant power savings.

The WDU evaluation was done on a 32–bit processor with virtually indexed L1 cache. For a machine with a physically indexed cache the WDU overhead would be even smaller resulting in higher power savings.

References

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*, pages 83–94, 2000.

[2] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, 1997.

[3] 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. In *IEEE 4th Annual Workshop on Workload Characterization*, pages 83–94, 2001.

[4] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 273–275, 1999.

[5] Intel. *Intel XScale Microarchitecture*, 2001.

[6] R. E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, Mar./Apr. 1999.

[7] A. Klaiber. The technology behind Crusoe processors. Technical report, Transmeta Corporation, january 2000.

[8] Motorola. *MPC7450 RISC Microprocessor Family User's Manual*, 2001.

[9] D. Nicolaescu, A. Veidenbaum, and A. Nicolau. Reducing power consumption for high-associativity data caches in embedded processors. Technical Report TR-2002, University of California, Irvine, 2002.

[10] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model.

[11] W. Tang, A. Veidenbaum, A. Nicolau, and R. Gupta. Simultaneous way-footprint prediction and branch prediction for ener gy savings in set-associative instruction caches.

[12] E. Witchel, S. Larsen, C. S. Ananian, and K. A. ic. Direct addressed caches for reduced power consumption. In *Proceedings of the 34th Annual International Symposium on Microa rchitecture (MICRO-34)*, December 2001.