Low Power Trace Cache for Embedded Processor

Je-Gil Moon*, Ha-Young Jeong*, Yong-Surk Lee**
* Dept. of Electrical and Electronic Engineering, Yonsei University, Seoul, 120-749, Korea
Tel : +82-31-209-2800 Fax : +82-31-209-4666 E-mail: {jgmoon,sixt06}@dubiki.yonsei.ac.kr
** Dept. of Electrical and Electronic Engineering, Yonsei University, Seoul, 120-749, Korea
Tel : +82-02-2123-2872 Fax : +82-02-312-4584 E-mail:yonglee@yonsei.ac.kr

Abstract:

Embedded business will be expanded market more and more since customers seek more wearable and ubiquitous systems. Cellular telephones, PDAs, notebooks and portable multimedia devices could bring higher microprocessor revenues and more rewarding improvements in performance and functions. Increasing battery capacity is still creeping along the roadmap. Until a small practical fuel cell becomes available, microprocessor developers must come up with power-reduction methods. According to MPR 2003, the instruction and data caches of ARM920T processor consume 44% of total processor power. The rest of it is split into the power consumptions of the integer core, memory management units, bus interface unit and other essential CPU circuitry. And the relationships among CPU, peripherals and caches may change in the future. The processor working on higher operating frequency will exact larger cache RAM and consume more energy.
In this paper, we propose advanced low power trace cache which caches traces of the dynamic instruction stream, and reduces cache access times. And we evaluate the performance of the trace cache and estimate the power of the trace cache, which is compared with conventional cache.

Keywords: Trace cache, Power Consumption, Embedded Cache, Instruction buffer

1. INTRODUCTION

With growing embedded businesses in the market, CPUs are running faster and faster. This makes memory reference a bottleneck in the overall computer performance. Thus more and more caches are used to ease the situations. The tendency of fast core and large cache consumes lots of power, and thus embedded microprocessor developers must come up with power-reduction methods. According to MPR 2003, the instruction and data caches of ARM920T processor consume 44% of total processor power. The memory access consume about half of the embedded microprocessor system power, the estimate of power distribution for the ARM920T processor, in which instruction and data caches consume 44% of total power as showing in Fig.1.

The remaining 56% is split among the integer core, memory management units, bus interface unit and other essential CPU circuitry. Higher operating frequencies will result in larger cache RAM and consume more energy. According to Amdahls's law, it is effective to reduce the memory access power of total processor power.

The instruction cache stores the static sequence of instructions generated by the compiler. Within a cache, there may be no relation between each basic block, because taken branches can randomly change the dynamic sequences of instructions in execution. Theses activities of taken branches introduce power consumption in the instruction cache. For instruction caches that support single basic-block fetching, any branch instructions within a fetched cache line will abort the instructions following that branch whether it is taken or not. As the size of a basic-block is normally around 5-6 instructions for general integer applications, this mechanism wastes not only the fetch bandwidth of the instruction cache but also the power consumption in fetching the whole cache line. The power consumption is due to the increased accesses to the instruction cache [3].

In this paper, we first introduce the low power caches with power-reduction methods in section 2. In section 3, we propose low power trace cache which replaces instruction cache. We have calculated trace cache hit rate, and our experimental result shows that the hit rate of trace cache only is about 70-90%, so we propose another low power trace cache, which includes instruction buffers, and the hit rate of instruction can be increased to about 99% with only 64 entry instruction buffers. The experimental model and result are detailed in section 4 and section 5. In section 6, we conclude our work and discuss some future works.

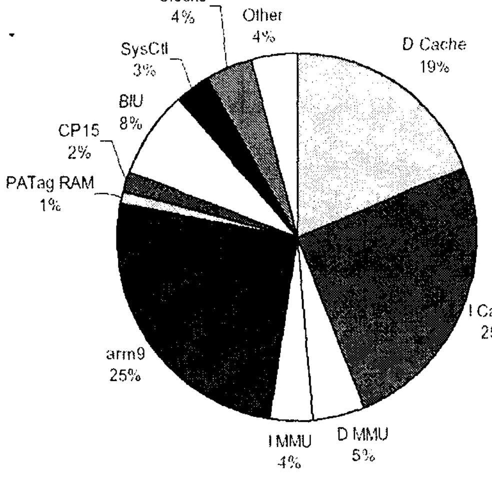

Fig.1. ARM920T power distribution shows dominant power consumption attributed to cache RAM and ALU (MPR 2003)

2. PREVIOUS LOW POWER CACHE

The conventional set-associative cache is commonly used in modern computer systems to reduce the conflict misses. However, the implementation of it is not power-efficient. A conventional n-way set-associative cache selects all n-way tags and data memories in a set, but, at most, it will only use one data block. The percentage of wasted energy will increase as cache associativity increases.

The phased lookup cache [7] first compares all the tags with the accessing address, then selects only the desired data way. The way prediction mechanism [8] is another effective approach that speculatively selects a way to access before making a normal cache access. Compared with the conventional implementation, the phased cache only selects one data sub-array instead of n data sub-arrays, and the way prediction cache first accesses the tag and data sub-arrays of the predicted way. If the prediction is not correct, it then probes the rest of tag and data sub-arrays simultaneously. If the prediction accuracy is high, the way-prediction cache is more energy-efficient than the phased cache [8].

The filter cache [9] is an extremely small cache of 32 to 64 words, which has very small access power consumption. It is tightly integrated with a processor and has very small access power compared to accessing a standard first level cache or a standard on-chip or off-chip program memory. But the frequent filter cache misses reduce overall fetch performance of the processor. Bellas et al. [10] used a profile-guided compiler to map frequently executed loops to a special address range, and discussed architecture extensions that would only load items in that range into the filter cache, thus reducing misses.

Unlike filter caches, a dynamically loaded loop cache does not impose performance overhead because of no suffering any misses from them. It involves no tag comparisons, resulting in even less power per access. However the dynamically loaded loop cache cannot cache loops with control of flow changes (COFs.). In some cases, it may increase more power dissipation due to extensive trashing, caused by particular loop with branches.

The hybrid loop cache designed by Ann Gordon-Ross is consisted of L1 cache, a main loop cache and a second level of preloaded storage. The main loop cache is loaded either from L1 cache dynamically or from a second level of preloaded storage on a COFs., if the next instruction falls within a preloaded region of code [10].

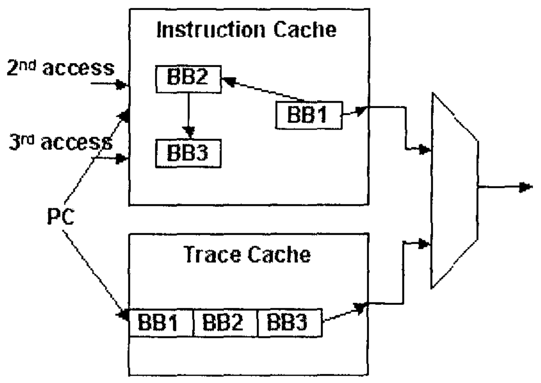

Fig.2. Conventional Trace Cache

The trace cache takes advantage of the fact that programs execute the same instruction paths repeatedly. The trace cache stores frequently executed non-contiguous instruction blocks as straightened out ones called traces [1]. For traditional instruction caches, any branch instruction within a fetched cache line will abort the remaining instructions in the line, whether it is taken or not. And the instruction will be fetched from the cache line following branches. Because, on the other hand, trace cache has contiguous multiple blocks in a trace, the portion of the cache line read on a taken branch is removed as showing Fig.2. But some configurations of conventional trace cache increase the power consumption in fetch unit. The main reasons for this power increment are the concurrent access to both trace cache and instruction cache. In conventional trace cache mechanism, the trace cache and L1 instruction cache are accessed simultaneously to reduce the miss penalty from trace cache.

Different from conventional trace cache, the sequential trace cache [4] has been investigated for performance and power, which works with instruction cache in a sequential manner. The instruction cache is accessed only after missing a previous trace cache lookup. Thus the sequential trace cache eliminates unnecessary accesses to the instruction cache. Dynamic prediction based trace cache maintains high performance, but its ability to reduce power consumption is limited. If the trace cache is accessed only when the fetch unit is pretty much sure about its hit, then the unnecessary miss penalty can be largely removed.

Jie S.Hu [3] proposed selective trace cache, which has the static prediction fetch unit thus resulting in high performance as well as low power consumption. But all of above trace caches are applicable to high-performance superscalar processors, not to the embedded processors.

In this paper, we first propose low power trace cache for embedded processor, which accesses only trace cache.

3. LOW POWER TRACE CACHE FOR EMBEDDED PROCESSOR

In conventional trace cache mechanism, the trace cache and L1 instruction cache are accessed simultaneously to reduce the miss penalty from trace cache. Because of the concurrent access to both trace cache and instruction cache, some configurations of conventional trace cache increase the power consumption in fetch unit. The sequential trace cache eliminates unnecessary accesses to the instruction cache, but has a long latency for misses in the trace cache. All of them are not applicable to embedded processor because they need so large memory blocks and power for instruction cache and trace cache.

We have designed the fetch unit for embedded processor using only low power trace cache, which can replace one using trace cache and instruction cache together. We evaluate it with the trace cache hit rate, and the target minimum hit rate is 99%. The hit rate of the conventional trace cache without instruction cache is below 70%. For the target minimum hit rate we implement partial tag matching, and branch folding, and optimal trace length, and get about 90% hit rate with 256 entry trace cache. As showing Fig.3, the hit rate of the trace cache is saturated at 256 entry trace lines. The partial tag matching can improve about 20% of the hit rate, and the optimal trace length is

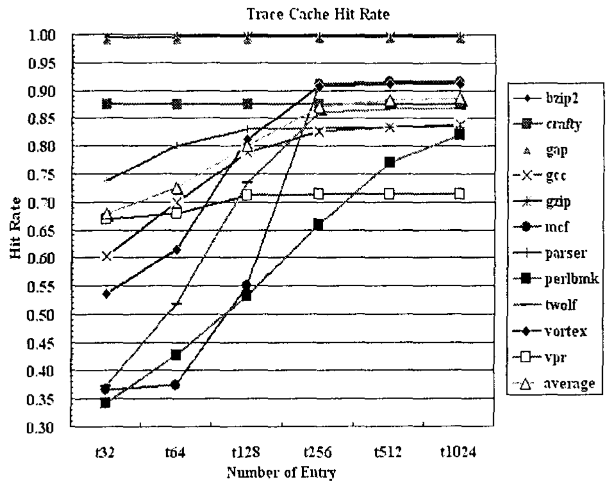

Fig. 3. Trace Cache Hit Rate

about 17~20 words. Partial tag matching is that when an address is located between start address and end address, the address is matching in that tag, and the hit signal is generated. The hit rate of low power trace cache can be increased through long basic blocks. The long basic blocks are made through branch folding about unconditional branches, which allow execution of branches with one-cycle gain compared to a sequential execution.

The miss penalty from the trace cache causes performance degradation on the fetch unit. So we propose low power trace cache, which includes 8-128 entry instruction buffers to reduce the miss penalty from it. As showing in Fig.4, the proposed low power trace cache fetch unit is consisted of instruction buffer, embedded trace cache, line fill buffer, filling logic and fast hit buffer.

The fast hit buffer is implemented because of large ppwer dissipation in trace cache caused by the long word line, which stores an entry trace least recently used. As showing in Fig.5, the hit rate of the fast hit buffer is about 86%, and we can save a lot of power dissipation of trace cache.

For low power, the proposed trace cache is designed to capture the locality of trace accessing and eliminate unnecessary accesses to the instruction cache and reduce the miss penalties.

The low power trace cache achieves almost the same performance as a conventional cache, while the power consumption of it is about 20% less. This improvement comes from reducing conflict in trace cache and eliminating so many accesses to large cache.

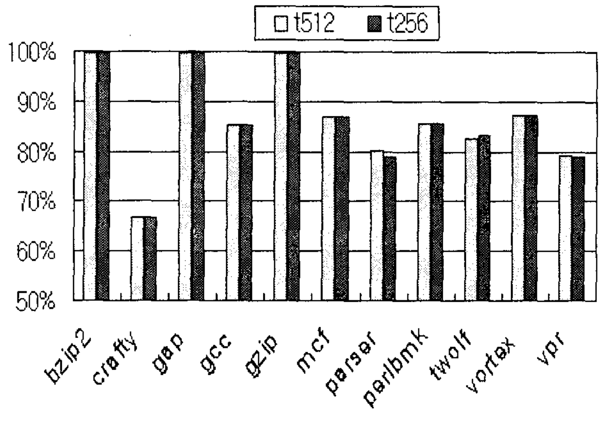

Fig.5. Fast Hit Buffer Hit Rate

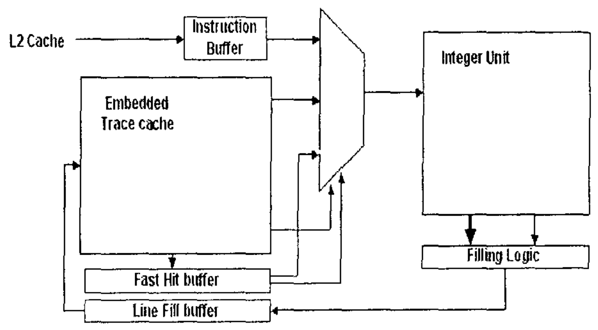

Fig.4. Low Power Trace Cache for Embedded processor

4. EXPERIMENTAL MODEL

The Wattch [12], a power evaluation tool to model the trace cache is augmented for our simulator. The power model of trace cache and fast hit buffer implemented is similar to the one in Wattch for the instruction cache. The power model of trace cache consists of two parts. One is for the power model consumed for trace cache lookup, and the other is for trace cache update, which does in complete stage.

The fetch mechanism is optimized for low power. The instruction lookup is matched in fast hit buffer at first, and the miss from fast hit buffer enables the trace cache and the instruction buffer.

The line-fill buffer is capable of holding up to 20 instructions. Instruction fetch queue size is 8. The decoding bandwidth is 1, and issue unit have a bandwidth of 2 instructions per cycle. The load/store queue has a capacity of holding 8 load/store instructions. The execution engine has 1 integer/FP ALU and 1 integer/FP multipliers/dividers. The timing parameters of memory hierarchy are shown in Table 1.

The integer benchmarks from SPEC2000 CINT are used. All benchmarks except bzip2 and mcf are first fast- forwarded 300 million instructions, then simulated

200 million instructions. No instruction is fast-forwarded for bzip2 and mcf due to their specific characteristics.

Table 1 Timing parameters in memory hierarchy.

Memory Hierarchy	Timing Parameters
L1 Instruction buffer	1 cycle hit latency
L1 Dcache	1 cycle hit latency
Memory	32 cycles for first chunk, 1cycles rest
TLB	30 cycles to service TLB miss

5. EXPERIMENTAL RESULT

As showing in Fig.6, the hit rate of the trace cache with instruction buffer is saturated at 512 entries of trace cache, and 128 entries of instruction buffer. The hit rate of t256 and i128 configuration is about 99% except perlbmk, and gcc, as showing in Fig.7. The hit rate of our low power trace cache about perlbmk, and gcc is small relatively, because they have so many subroutine calls, which cause the low power trace cache miss. For embedded processor, the low power trace cache can be effectively archives 99% hit rates with 256 entries of trace cache and 32 entries of instruction buffer. And the performance degradation is below 4%, as showing in Fig.8, except perlbmk, and gcc.

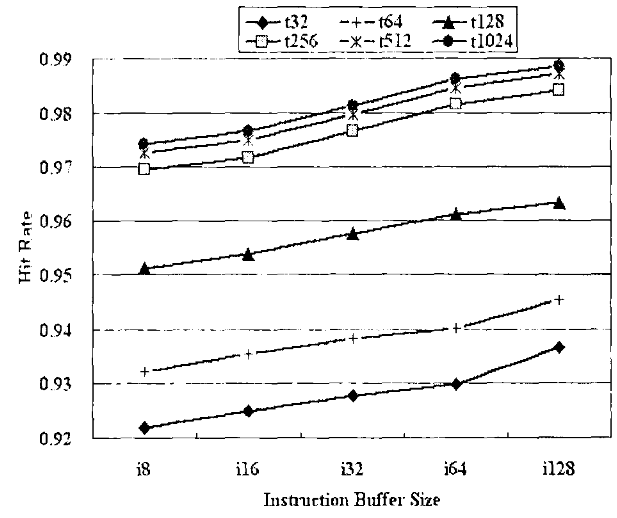

Fig.6. The hit rate of low power trace cache

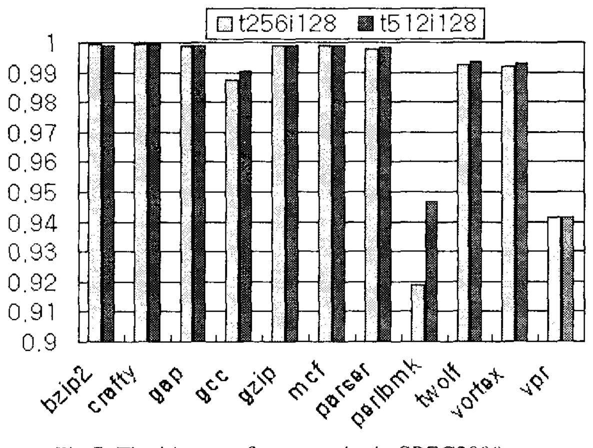

Fig.7. The hit rate of trace cache in SPEC2000

The design principle of low power trace cache is that only traces that belong to the dominant set are allowed to be stored in the trace cache, and others are stored in small size instruction buffers. The instruction lookup is matched in fast hit buffer at first, and the miss from fast hit buffer enables the trace cache and the instruction buffer. The hit rate of one entry fast hit buffer is about 86%. From working in this way, the low power trace cache eliminates not only the unnecessary accesses to the instruction cache, but also reduces the majority of lookups that result in misses to the trace cache.

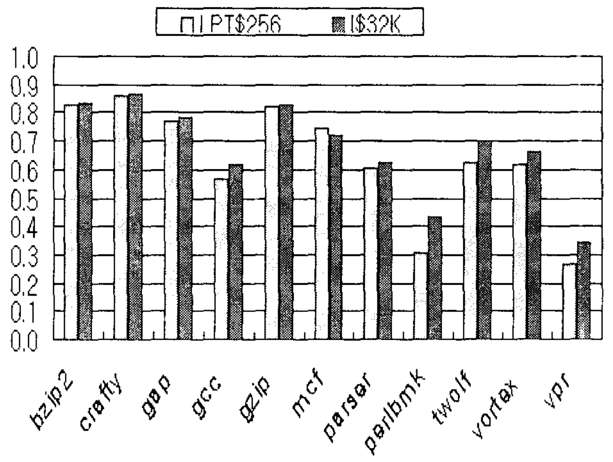

Fig.7. Performance comparison

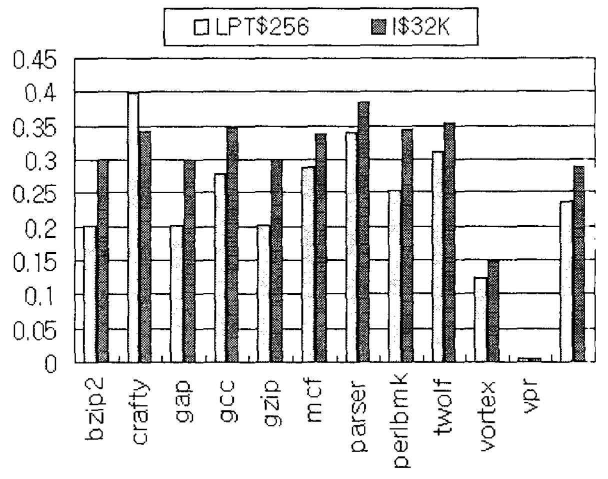

Fig.8. Power comparison

Thus the low power trace cache achieves almost the same performance as a conventional cache, while the power consumption of low power trace cache is about 18% less

6. CONCLUSIONS

Power consumption in embedded processors s becoming an important for battery-life and performance. In this paper, we first explore the power saving techniques in traditional instruction caches and conventional trace caches. With analyzing the distribution of individual trace and trace hits, we propose low power embedded trace cache which replace the instruction cache. But the hit rate of trace cache only is about 70-90%, so we propose another low power embedded trace cache including instruction buffers. Our result shows that the hit rate of instruction can be increased to about 99% and saved power about 18% with those buffers.

This paper shows that optimizing trace cache and instruction buffer cache replaces the conventional cache, and then reduces power consumption.

In view of the encouraging results of the low power trace cache, we must compare other low power caches with our work. And we believe that other high performance trace cache techniques should be investigated. We must consider selection techniques in [3], study some more effective hit mechanism. For applying to embedded processor, we must consider the utilization of trace cache memory.

References

[1] E. Rotenberg, S. Bennett, and J. E. Smith, "Trace cache: A low latency approach to high bandwidth instruction fetching", MICRO-29, pp.24-34, 1996

[2] E. Rotenberg, S. Bennett, and J. Smith, "A trace cache microarchitecture and evaluation," *IEEE Transactions on Computers (Special issue on cache memory)*, vol. 48, pp. 111–120, 1999.

[3] Jei S. Hu and N. Vijaykrishnan. "Selective Trace Cache: A Low Power and High performance Fetch Mechanism", *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2002

[4] J. S. Hu, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Power-efficient trace caches" in *Proc. Of the 5th Design Automation and Test in Europe Conference (DATE'02)*, 2002.

[5] J. Faistl and T. Jaracz, "Trace cache: Effect on instruction cache miss frequency." http://www.ece.cmu.edu/_ee742/proj s98/faistl/index.html, 1998.

[6] N. P. Jouppi, ``Improving Direct Mapped Cache Performance by the Addition of a Small Associative Cache and Prefetch Buffers", Proceedings of the 17th Annual International Symposium. on Computer Architecture, pp. 363-373, 1990

[7] A. Hasegawa et al., "SH3: High Code Density, Low-Power", IEEE Micro, vol.15, pp. 11-19, 1995.

[8] Zhichun Zhu, Xiaodong Zhang "Access-mode predictions for low-power cache design" Micro, IEEE vol. 22, pp. 58-71, 2002.

[9] J. Kin, M. Gupta, W. Mangione-Smith. "The Filter Cache: An Energy Efficient Memory Structure." Int. Symposium on Microarchitecture, pp. 184-193, 1997.

[10] N. Bellas, I. Hajj, C. Polychronopoulos, G. Stamoulis. "Energy and Performance Improvements in Microprocessor Design Using a Loop Cache." Int. Conference on Computer Design, pp. 378-383, 1999.

[11] Ann Gordon-Ross, Frank Vahid, "Dynamic Loop Caching Meets Preloaded Loop Caching-A Hybrid Approach", Int.Conference on Computer Design, pp. 446-449, 2001

[12] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in Proc. of the 27th Annual Int. Symposium on Computer Architecture, pp. 83-94, 2000.

[13] "SimpleScaler Introduction and Tutorial", www.simplescalar.com/docs/users_guide_v2.pdf