On-line Reconfigurable Cache for Low Power Embedded Systems

Yu-Tien Liao
Department of Computer Science and Information Engineering
National Chi Nan University
Puli, Nantou Hsien, 54561 Taiwan
Email: s95321054@ncnu.edu.tw

Dyi-Rong Duh
Department of Computer Science and Information Engineering
National Chi Nan University
Puli, Nantou Hsien, 54561 Taiwan
Email: drduh@ncnu.edu.tw

***Abstract*—Modern embedded systems execute a small set of applications or even a single one repeatedly. Specializing cache configurations to a particular application is well-known to have great benefits on performance and power. To reduce the searching for optimal cache configuration, a most-case optimal cache configuration searching algorithm was proposed which greatly reduces the time and power in searching. However, the fact that the behavior of an application varies from phase to phase has been shown in recent years. Tuning cache configuration to fit a target application in different phases gives a further improvement in power consumption. This work presents a mechanism which determines the optimal configurations in different phases during an execution process. By dividing an execution process into small time intervals and applying corresponding local optimal cache configuration for each interval on L1 instruction cache, this work shows that over 4.751% energy saving is obtained compared with whole application be divided into 64 intervals. On average 6.626% power reduction is achieved compared to a benchmark with its respective global optimal cache configuration.**

***Index Terms*—Cache, reconfigurable cache; embedded systems; power consumption; benchmark.**

I. Introduction

The research of computer systems is growing and is getting more important in the last three decades. The gap between the processor and the memory has widened when computer systems improve. It has been shown that inserting caches between processors and off-chip memories fills the gap efficiently. Embedded microprocessors have been widely used for the hand held devices due to their high performance, low energy consumption, and low cost. Caches are infixed in embedded microprocessors with the growth of embedded systems. The most important thing for embedded systems like PDA and cellular phone is how long the device is alive. Segars [9] showed that an embedded microprocessor such as the ARM920T processor dissipates more than 50% of the total power in the cache. With the increasing usage of cache, energy savings in caches have become more critical.

Sherwood et al. [10] showed that programs may have different run-time behaviors in different portions of an execution as a series of phases. A phase is defined as an interval of execution during which a measured program metric is relatively stable. The scheme proposed by Sherwood et al. captures phases that account for over 80% of execution using less than 500 bytes of on-chip memory [10].

Some strategies augmented small caches between the processor and lower-level cache to reduce the energy consumption, such as the filter cache introduced by Kin et al. [6] and the hotspot cache introduced by Yang et al. [14]. Others improved the cache itself to reduce the energy consumption. Many researches on caches with tunable parameters to reduce power consumption have been introduced in recently years. Zhang et al. [15] introduced a highly configurable cache architecture which dynamically configures the associativity size and line size with certain restrict combinations. Banerjee et al. 0 combined both program phase detection and cache way reconfigurable to achieve 32% power reduction on memory access with less than 2% performance degradation.

A single-pass multi-cache evaluation has been proposed and implemented in hardware by Gordon-Ross et al. [4] which speeds up 7.7 times in tuning time. Even so, the hardware overhead is so tremendous which occupies up to 12% area of ARM920T. Lin and Duh proposed an on-line reconfigurable cache which greatly reduces the search time and the cache tuning can proceed during the execution [7].

Jheng and Duh proposed a reconfigurable cache scheme which is divided an application into a number of fixed 64 intervals to find every local optimal cache configuration. By dividing an application into a number of fixed intervals, it not only achieves the purpose of energy saving, but also overcomes meeting two low points could find the better one [8]. The work improves Jheng et al.'s scheme, just try to divide the interval into smaller ones which were fixed 1 million instructions tuning once, but the number of intervals were not fixed. The algorithm is almost the same as Jheng et al. and Lin et al.'s algorithms but Lin et al.'s algorithm goes further in statistics. Based on Jheng et al.'s study, this work enhances the scheme to on-line tuning.

The remainder of this paper is organized as follows. Section 2 first introduces the architecture of reconfigurable cache. The energy model is then presented. Section 3 presents how to implement an on-line reconfigurable cache system. Section 4 demonstrates experimental methodology and results. Conclusions are finally drawn in Section 5.

II. Reconfigurable Architecture

This section describes the fundamentals of the proposed reconfigurable cache architecture. The search space of configurable cache parameters and how the parameters are tuned are introduced at first. Then, how the hardware is modified to support reconfiguration is illustrated. Finally, an energy model is used to evaluate the proposed scheme.

A. Parameters

Instead of adopting conventional cache parameters such as cache size, cache block size, and way associativity, the proposed reconfigurable cache architecture includes three parameters: the number of blocks in a set, cache block size, and way associativity. The three parameters are nblks, bsize, and assoc for short in this paper. Cache size is composed of nblks, bsize and assoc. The parameter nblks is chosen to replace cache size because of the "simple and dump" principle. Considering hardware implementation, it is under a complicated circumstance for the conventional cache parameters. Changing cache size affects nblks simultaneously and so are the other two parameters. As cache size is not a parameter in this work, tuning one of the three parameters just simply doubles or halves the cache size and does not affect other parameters.

The bounds of configuration search space are decided by extending Lin et al.'s work [7]. The upper bound of nblks is 512 and the lower bound is 16. The upper bound of bsize is 64 bytes and the lower bound is 8 bytes. The minimum value of bsize is also the lowest bound supported by the simulator. The upper bound of assoc is 16 and the lower bound is 1. There are 120 combinations of three parameters which is much larger than the number of combinations in Lin et al.'s work [7]. According to the statistic of more than forty benchmarks, there is no any value of the three parameters abandoned by every benchmark. The extension of searching space is necessary because the instability of a phase is larger than that of entire execution of a program.

B. Architecture of Reconfigurable Cache

The number of blocks in a set is limited to be powers of two to enable simple decoding. The blocks are physically divided into 6 sub arrays: 256, 128, 64, 32, 16, and 16 blocks. The partition makes it possible to tune from 512 to 16 nblks. The unused blocks are shutdown for saving energy consumption.

An approach introduced by Chen et al. [3] can adjust the line size by configuring a counter which indicates the number of words to be read from the off-chip memory when there is a miss. The associativity tuning is implemented by Banerjee et al.0. A simple hardware counter keeps track of the miss rate and the information is fed back to the way controller. Each cache way can be selectively enable or disable by the controller.

In an non-tuned cache, the size of tag adjusts to $32 - \log_2(\text{nblks} \times \text{bsize})$ under 32bits addressing mode. Dynamically tuning cache size during an execution of programs changes tag size as well. Even so, implementing a multi-length tag comparator is not considered efficient in this work. A feasible solution is the overlapped wide-tag [3] which extends the size of tag to maximum length in search space. Although some bits of tag may be redundant, it makes cache be any size without additional control hardware.

C. Energy Calculation

With the increasing of working frequency, the dynamic power consumption dominates most energy consumption in caches, so the static power consumption is not considered. The energy consumption caused by off-chip memory should not be ignored since the accessing of off-chip memory is a part of a normal memory access. When a cache misses, the instruction fetching and data referencing cause huge miss penalty. Besides the cache inner calculates, the off-chip accessing is included.

To calculate the energy consumed in the proposed cache architecture, the adopted energy model should be calculated fast and easily. We modify the energy model used by Shiue and Chakrabarti [11] for calculating the energy consumption to support cache reconfiguration. This model shows in (1) that is separated by four components.

Edec is the energy consumption of the decoding logic and translating in address buses. The capacitance of a decoding logic is usually much less than that of an address bus; hence the energy consumption of the address buses dominates the entire address decoding path. Ecell denotes the energy consumption of the read/write circuitry and cell array due to status bit, tag, and data cell arrays. State bit, tag cell, and data cell consume most energy in cell array, and they can be implemented in dynamic or static logic. In dynamic circuit design, word/bit lines are pre-charged on every access. The energy consumed on bit lines is the same whether the values on the bit lines are 1 or 0. In a static circuit design, there is no pre-charge on word/bit lines. The energy consumption on tag and data memory array directly depends on the bit switch of the word/bit lines. For above reason we only compute the dynamic energy consumption. It will access off-chip memory via address/data pad when cache misses.

E_{io} represents the energy consumption in I/O pad. In current microprocessors, the capacitance of I/O pads is usually larger than that on address and data buses. Therefore the energy consumption of I/O pad is dominated by the energy consumed during the bit switch of the I/O path. Emem stands for accessing an off-chip memory. The energy model adopted in this work is stated as follows:

$$E_{total} = E_{dec} + E_{cell} + E_{io} + E_{main} \quad (1)$$

$$E_{dec} = \alpha \times Pr_A \times W_{add} \times (C / L) \times (N_{hit} + N_{miss})$$

$$E_{cell} = \beta \times [W \times (8 \times L + T + st)] \times [C / (W \times L) + 4.8] \times (N_{hit} + N_{miss} - N_{WPhits}) + \beta \times (8 \times L + T + St) \times [C / (W \times L) + 4.8] \times N_{WPhits}$$

$$E_{io} = \gamma \times (Pr_D \times 8 \times L + Pr_A \times W_{add}) \times N_{miss}$$

$$E_{mem} = \gamma \times (Pr_D \times 8 \times L + Pr_A \times W_{add}) \times N_{miss} + E_m \times 8 \times L \times N_{miss}$$

where

Pr_A =	the probability of one bit switch in address bus/pad
Pr_D =	the probability of one bit switch in data pad
W_{add} =	the width of address bus
C =	cache size in bytes
L =	cache line size in bytes
W =	way associativity
N_{hit} =	number of hits
N_{miss} =	number of miss
N_{WPhits} =	number of way prediction hits
T =	tag size in bits
St =	number of status bits per block frame
E_m =	energy consumption of a main memory access

$\alpha = 7.89\text{e-}17$; $\beta = 1.44\text{e-}14$; $\gamma = 5.45\text{e-}11$

α, β, γ are used for 0.8 μm CMOS technology

Su and Despain [12] revealed the relationship between bit switch rate and miss rate. An average bit switch rate is chosen as 4K bytes for cache size, 16 Bytes for cache line size, and 2-way for way associativity. By Su et al. [13], the value of PrA is 0.0045, and PrD is 0.0044. Clearly, it takes at least 14 bits address bus to access 16 Kbytes cache size. The SRAM CY7C1326-133 from Cypreess is used as our cache memory. The size of the SRAM is 2Mbits, access time 4ns, voltage of 3.3V, energy consumption of 4.95nJ per access. St is the symbol of a valid bit which is the only one status bit in the proposed scheme.

III. On-line Reconfigurable Cache

This section shows how to achieve reconfiguration in real-time. First, searching algorithms are showed and the proposed searching algorithm is given. Second, the initial configuration of searching algorithm is decided by statistic. Third, the priority of the effect of csize, bsize, and assoc is revealed. Finally, a real-time scheme is proposed by combining ideas mentioned in this chapter.

A. Algorithm

The algorithm used in this work is based on Lin et al.'s work [7] and the algorithm is modified to fit the on-line scheme. The algorithm tunes one of three parameters at a time. When the low point is reached, the algorithm goes to the next parameter. The proposed algorithm has three input sets: nblks, bsize, and assoc. In each interval, number of cache misses can be generated by miss counter and sent to the energy calculator. The following is the general model of the algorithm.

```
run the application in the time interval
while(time is up )
{
current energy = energy calculator( nblks, bsize, assoc, misses, length of interval);
if(optimal energy > current energy)
   optimal value = current value;

if(optimal value is minimum or maximum in search space)
   if(value next to the optimal value is checked)
      optimal value of the parameter is found;
   else
      next value = the value next to the optimal value;
else
   if(both sides of optimal value is checked)
      optimal value of the parameter is found;
   else
      if(value = 1/2 × optimal value is checked)
         next value = 2 × optimal value;
      else
         next value = 1/2 × optimal value;

if(optimal value of the parameter is found)
   goto next parameter;
else
   current value = next value;
}
```

The proposed algorithm searches for optimal value of each parameter sequentially. As the optimal energy and current energy are the only information available, the direction of tuning is unknown when the algorithm begins. For the consistency of implementation, a smaller value is always chosen to be the next configuration at first. If a value is selected as the optimal parameter, both larger and smaller ones around the optimal parameter must have been tested. If the value is the minimum or maximum in the search space, only one side has to be tested.

B. Priority of Three Parameters

The algorithm introduced in above section tunes one of three parameters at a time. The priority of parameter tuning is decided by the order of impact on energy. Figure 1 is the analysis of the energy model stated in Section 2.5. The x–axis is the number of misses during 1 million instructions and the y-axis is the average percentage of energy increasing when one parameter doubles its value.

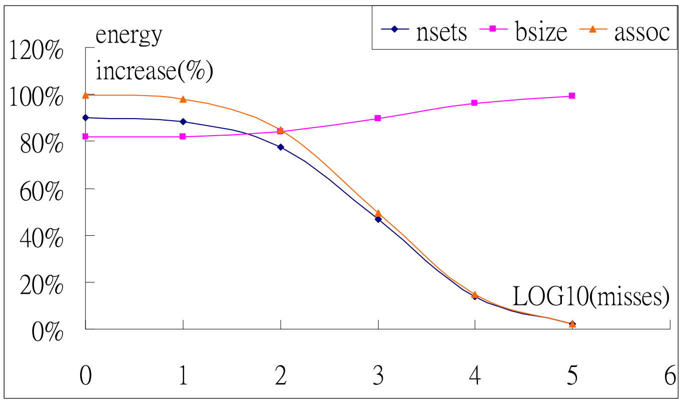

Figure 1. Analysis of the energy model.

As shown in Figure 1, when the number of misses is less than 100, the assoc is the most influential parameter. When the number of misses is larger than 100, the impact of assoc and nblks decays in a fast speed while the impact of bsize increases on the contrary. Based on statistics of the simulation, the number of misses under most cache configuration is greater than 100 in 1 million instructions. The priority of parameters is decided as bsize, assoc and nblks.

C. Statistic Starting Point

An important observation has been introduced in Lin et al.'s work [7]. With the increasing of nblks, the arc of energy consumption versus nblks should have only one low point for any benchmark and so are the other two parameters. The red line in Figure 2 is not available in the simulation of this work. The phenomenon is apparent when one parameter is tuned. The situation gets much complicated if there are three parameters tuning simultaneously.

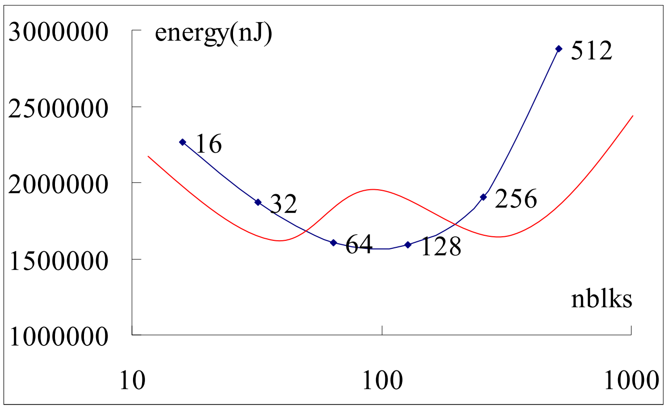

Figure 2. The energy consumption of nblks from 16 to 512. The benchmark is bitcount; bsize is 8 bytes, and assoc is 4. The red line is a synthetic example which is not available in this work.

Starting point is the initial value of cache configuration when the proposed algorithm begins. If the starting point is too far from the low point, it is possible for the cache configuration tuning to be trapped in sub-low points. To avoid this case, the starting point should be near the low point. After gathering statistics of all benchmarks in this work, the values of each parameter appear most in optimal configurations are chosen as starting point. The nblks is set as 64, bsize as 16 bytes and assoc as 8 to the configuration of starting point for L1 instruction cache.

D. Analysis of average Tuning Time

According to the algorithm in section 3.1, the best case of tuning time is 1 + 1 + 1 and the worst case of tuning time is 6 + 5 + 4. The best case and the worst case are the same as Lin et al.'s work [7]. The average tuning time of nblks is given in Table 1.

Each element in Table 1 records the steps needed when tuning from any nblks in column X to next nblks in row Y. For example, when current nblks is 512 and the next optimal nblks is 32, it takes 4 steps to go from 512 to 32. There is no information whether the value 16 is better or not, so it is necessary to change nblks to 16. After applying try and error strategy, the value 16 is found worse than 32 and the nblks is changed back to 32. It totally costs 6 tune times as shown in Table 1. The assumption that the probability of next nblks is average to every nblks value is adopted in this analysis.

Table 1. Tuning Time of All Cases for Current nblks to Next nblks

Y / X	512	256	128	64	32	16	sum
512	2	3	4	5	6	5	25
256	2	3	3	4	5	4	21
128	3	4	3	3	4	3	20
64	4	5	4	3	3	2	21
32	5	6	5	4	3	1	24
16	5	6	5	4	3	2	25
sum							136

The possibility of next nblks is 1/6 to each value so the average tuning time from nblks 512 to next nblks is 25/6. Because the possibility of current nblks is also 1/6 for each nblks, the average tuning time for nblks is 136/36. By the same estimation, the average tuning time of bsize is 45/16 and the average tuning time of assoc is 83/25. The overall average tuning time of the proposed algorithm is the sum of average cases of three parameters which equals 9.91. When the proposed algorithm is applied, it tunes 9.91 times on average to find an optimal configuration.

E. On-line Reconfigurable Scheme

The idea of on-line reconfiguration is dividing the execution of an application into constant time intervals. The interval between tuning is set to be 1 million instructions in this work. During each interval, the proposed algorithm is applied to find the optimal cache configuration and the found configuration is recorded for the next round. The optimal configuration found in an interval is called a local optimal configuration while the optimal cache configuration found during the whole execution is called global optimal one. A local optimal configuration may not be optimal for the next interval because the behavior of the application may differ. A reasonable scheme is shifting the recorded local optimal configuration backward to the start of the interval when the second round is coming. This scheme ensures that the applied optimal configuration is suitable for the current time interval and saves energy in a significant degree.

IV. Experimental Results

This section demonstrates the simulation results. The simulation environment in this work is first introduced and simulation results are then shown.

A. Environment

SimpleScalar introduced by Buger et al. [2], a MIPS-like microprocessor simulator, is the tool set used in this work. This tool is developed by University of Wisconsin, Madison. We carefully modify and verify the SimpleScalar tool set to support on-line cache reconfiguration. Mibench benchmark suit is taken for embedded systems simulation that is developed by University of Michigan, Ann Arbor [5]. The simulation is under the cache which is modeled as in-order execution, write-back and the replacement policy is LRU.

B. On-line Reconfiguration Results

The proposed reconfigurable scheme saves energy by finding local optimal configuration which is considered better than global optimal one. The dijkstra benchmark calculates the shortest path between every pair of nodes in a graph. The global optimal cache configuration of dijkstra is 64 for nblks, 8 bytes for bsize and 16 for assoc. As shown in Figure 3, the global optimal configuration is not always the best configuration in the execution. There is another configuration better than the global one periodically. The purpose of the proposed scheme is always to select the cache configuration with lowest energy consumption.

However, not every benchmark in Mibench is sensible to cache reconfiguration. The sha benchmark in Figure 4 is a secure hash algorithm that produces a 160-bit message digests for a given input. The global optimal configuration of sha dominates the whole execution and there is no configuration better in any time interval. To these insensible benchmarks, the proposed scheme still makes sense. Although it is in vain for the proposed scheme to tune cache configuration, the global optimal cache configuration of insensible applications will be found by the scheme and the insensibility will be revealed.

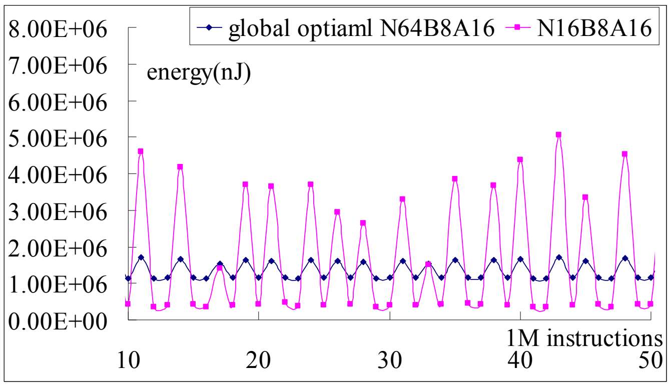

Figure 3. The energy consumption of dijkstra benchmark with different cache configurations.

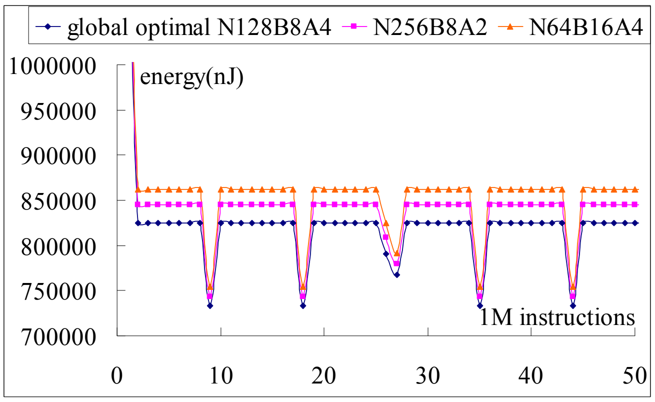

Figure 4. The energy consumption of sha benchmark with different cache configurations.

Figure 5 shows the overall energy saving on Mibench. The proposed scheme achieves on average 6.626% energy reduction compared to respective global optimal cache configurations of all benchmarks. Figure 6 also displays that cache configuration for every 1 million instructions tuning once is more saving energy than for 64 intervals one on average 4.751% energy reduction. Besides some cache tuning insensible applications, applications with tiny number of instructions benefit nothing from the proposed scheme because the time interval is still too larger.

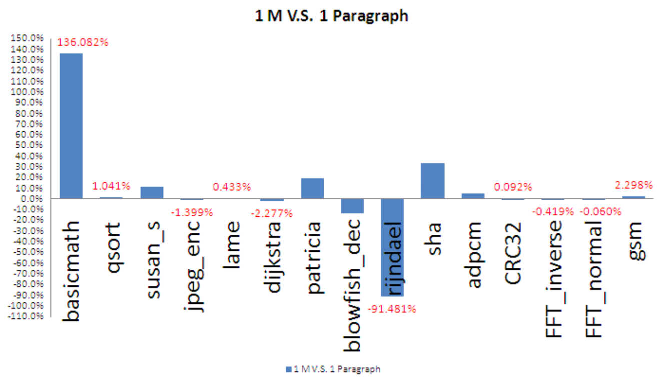

Figure 5. The average energy reduction for each application of Mibench. Local V.S. Global optimal cache configuration.

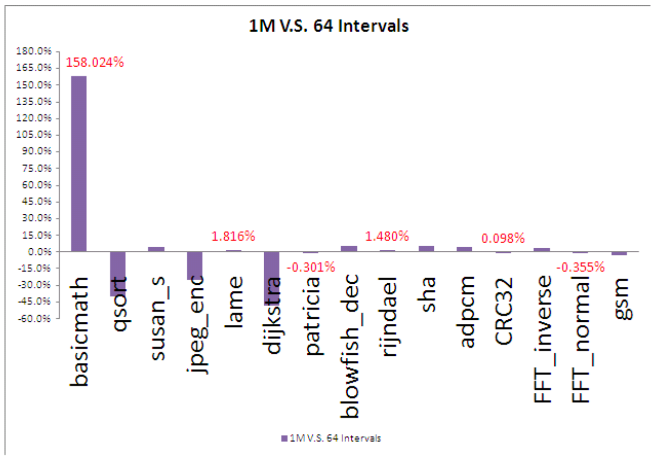

Figure 6. The average energy reduction for each application of Mibench. (Local_1M V.S. Local_64_intervals optimal cache configuration.

V. Conclusions

Saving energy is undoubtedly an important issue for embedded systems. As embedded systems work in a quite simple environment, it makes sense to optimize cache configuration to a certain application. Lin et al. proposed an on-line reconfigurable cache architecture which reduces energy over a non-tuned reference cache [7] and Jheng et al. further presented a real-time reconfigurable cache architecture for finding local optimal cache configuration but the interval will be divided into 64 ones which reduces energy over the whole application tuning once reference cache. [8] This work provides an on-line reconfigurable cache scheme which improves on average 6.626% energy reduction over Lin and Duh's work and also on average 4.751% energy reduction over Jheng and Duh's work.

Acknowledgment

The authors would like to thank the National Science Council of the Republic of China, Taiwan for financially supporting this research under Contract No. NSC-97-2221-E-260-001-.

Reference

[1] S. Banerjee, Surendra G, and S. K. Nandy, "Program phase directed dynamic cache way reconfiguration for power efficiency," in: Proc. the 12th Asia and South Pacific Design Automation Conference, Yokohama, Japan, 2007, pp. 884–889.

[2] D. Burger and T.M. Austin, "The Simple Scalar tool set, version 2.0," ACM SIGARCH Computer Architecture News, Vol. 25, No. 3, pp. 13–25, 1997.

[3] L.-M. Chen, X.-C. Zou, J.-M. Lei, and Z.-L. Liu, "Dynamically reconfigurable cache for low-power embedded system," in: Proc. the 3rd International Conference on Natural Computation, Vol. 5, 2007, pp. 180–184.

[4] A. Gordon-Ross, P. Viana, F. Vahid, W. Najjar, and E. Barros, "A one-shot configurable-cache tuner for improved energy and performance," in: Proc. the 2007 IEEE/ACM Design, Automation and Test in Europe (DATE), 2007, pp. 1–6.

[5] 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: Proc. the 4th IEEE Annual Workshop on Workload Characterization, 2001, pp.3–14.

[6] J. Kin, M. Gupta, and W. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," in: Proc. the 30th ACM/IEEE Annual International Symposium on Microarchitecture, Research Triangle Park, North CA, USA, 1997, pp. 184–193.

[7] C.-H. Lin and D.-R. Duh, "On-line reconfigurable cache for embedded systems, " in: Proc. the 2006 Conf. on Information

Technology and Applications in Outlying Islands, Jinning, Kinmen, Taiwan, June 2–3, Taiwan, 2006.

[8] G.-C. Jheng, D.-R. Duh, and C.-N. Lai, "Real-time reconfigurable cache for low power embedded systems," in: Proc. the 25th Workshop on Combinatorial Mathematics and Computation Theory, Chung Hua University, HsinChu, Taiwan, April 25–26, 2008, pp. 181–187.

[9] S. Segars, "Low power design techniques for microprocessors," in: Proc. the 4th IEEE International Solid-State Circuits Conference Tutorial, San Francisco, CA, USA, 2001.

[10] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," in: Proc. the 30th Annual International Symposium on Computer Architecture, 2003, pp. 336–347.

[11] W.-T. Shiue and C. Chakrabarti, "Memory design and exploration for low power, embedded systems," Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology, Vol. 29, No. 3, pp. 167–178, 2001.

[12] C.-L. Su and A. Despain, "Cache design trade-offs for power and performance optimization: A case study, " in: Proc. the 1995 International Symposium on Low Power Design, Dana Point, CA, USA, 1995, pp. 63–68.

[13] C.-L. Su and A. M. Despain, "Cache designs for energy efficiency, " in: Proc. the 28th Hawaii International Conference on System Sciences, Kihei, Maui, Hawaii, 1995, pp. 306–315.

[14] C.-L. Yang and C.-H. Lee, "HotSpot cache: Joint temporal and spatial locality exploitation for I-cache energy reduction," in: Proc. the International Symposium on Low Power Electronics and Design, 2004, pp.114–119.

[15] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache for low energy embedded systems," ACM Transactions Embedded Computing Systems, Vol. 4, No. 2, pp. 363–387, 2005.

[16] C. Zhang, F. Vahid, and W. Najjar, "Energy benefits of a configurable line size cache for embedded systems," in: Proc. IEEE Computer Society Annual Symposium on Very Large Scale Integration, Darmstadt, Germany, 2003, pp. 87–91.