Location Cache: A Low-Power L2 Cache System

Rui Min, Wen-Ben Jone and Yiming Hu
Department of Electrical & Computer Engineering and Computer Science
University of Cincinnati
Cincinnati, OH 45221-0030
e-mail: {minri,wjone,yhu}@ececs.uc.edu

ABSTRACT

While set-associative caches incur fewer misses than direct-mapped caches, they typically have slower hit times and higher power consumption, when multiple tag and data banks are probed in parallel. This paper presents the location cache structure which significantly reduces the power consumption for large set-associative caches. We propose to use a small cache, called *location cache* to store the location of future cache references. If there is a hit in the location cache, the supported cache is accessed as a direct-mapped cache. Otherwise, the supported cache is referenced as a conventional set-associative cache.

The worst case access latency of the location cache system is the same as that of a conventional cache. The location cache is virtually indexed so that operations on it can be performed in parallel with the TLB address translation. These advantages make it ideal for L2 cache systems where traditional way-predication strategies perform poorly.

We used the CACTI cache model to evaluate the power consumption and access latency of proposed cache architecture. *Simplescalar* CPU simulator was used to produce final results. It is shown that the proposed location cache architecture is power-efficient. In the simulated cache configurations, up-to 47% of cache accessing energy and 25% of average cache access latency can be reduced.

Categories and Subject Descriptors: B.3.2 [Design Styles]: Cache memories

General Terms: Design, performance, power

Keywords: L1/L2 caches, TLB, set-associative caches, data location

1. INTRODUCTION

To achieve low miss rates, modern processors employ set-associative caches. In a RAM-tagged *n*-way set-associative cache, *n* tag and data ways are accessed concurrently. This wastes energy because at least $n - 1$ data reads are useless for each cache access. Methods to save energy for set-associative caches have been actively researched.

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.

1.1 Structural Approaches

The structural techniques typically segment the word-lines or the bit-lines. Subbanking (also known as column multiplexing) technique [1, 2] divides the data arrays into subbanks. Only those subbanks that contain the desired data are accessed. The bit-line segmentation scheme [2] partitions the bit-lines. When the memory cells are sampled, only required bit-line segments are discharged. The MDM (multi-divided module) cache [3] consists of small modules with each of them operating as a stand-alone. Only the required small module designated by the reference presented to the cache is accessed. Albonesi proposed the re-sizable selective ways cache [4]. The cache set-associativity can be reset by the software.

An other type of structural method is to a add small piece of cache to capture the most recently referenced data or to contain prefetched data. Line buffer designs [1, 2] were proposed to cache the recently accessed cache lines. Filter cache [5] is a small cache that sits between the CPU and the L1 caches. It reduces L1 cache power consumption by filtering out the references to the L1 caches. Many of the structural approaches have been proved efficient. They can be used with other strategies describe in the following sections.

1.2 Alternative Cache Organizations

Phased caches [1, 6, 7] first access the tag and then the data arrays. Only the hit data way is accessed in the second phase, resulting in less data way access energy at the expense of longer access time. Researchers recently proposed the *way concatenation* technique [8] for reducing dynamic cache power for application-specific systems. The cache can be configured by software to be a direct-mapped, two-way or four-way set-associative cache so as to save power. The MNM mechanism [9] is proposed to discover cache misses early so that power consumption of the cache can be saved. CAM-tagged caches are often used in low-power systems. A CAM based cache puts one set of a cache in a small sub-bank and uses a CAM for the tag lookup of that set. A set may have 32 or even 64 ways. However, the CAM tags must be searched before the data can be retrieved, which increases the cache latency. The area overhead brought by CAM cells is also not negligible.

1.3 Speculative Way Selection

The basic idea of speculative way activation is to make a prediction of the way where the required data may be located. If the prediction is correct, the cache access latency and power consump-

tion is similar to that of a direct-mapped cache of the same size. If the prediction is wrong, the cache is accessed again to retrieve the desired data. The cache is accessed as a direct-mapped cache twice. Because of high prediction accuracy, proposed designs have saved both time and power. Some designs have also been industrialized. Prior work can be categorized by the way the cache is probed.

1.3.1 Statically Ordered Cache Probes

The Hash-Rehash cache design [10] and the Pseudo-associative cache design [11] were originally proposed to reduce the miss rates of direct-mapped caches. When a memory reference is presented to the cache, +the direct-mapped location is checked. If there is a miss, a hash function is used to index the next cache entry. In both designs, the most-recently-accessed cache line will be moved to the direct-mapped location. However, exchanging large cache lines consumes large amount of power as well as bus bandwidth.

1.3.2 Dynamically Ordered Cache Probes

In contrast to the static schemes, researchers have developed schemes which redirect the first probe to a predicted location. The MRU cache design [12] keeps the MRU information associated to each set. When searching for data, the block indicated by the MRU bit is probed. However the MRU bits must be fetched prior to accessing the cache. The PSA (Predictive Sequential Associative) cache design [13] moves the prediction procedure to previous stages of pipelining so that the MRU information is presented to the cache simultaneously with the memory reference. The Reactive-Associative Cache design [14] moves most active blocks to direct-mapped positions and reactively displaces only conflicting blocks based on the PSA cache design. It reduces cache assess latency at the cost of higher miss rates and larger power consumption. Dropsho [15] discussed an accounting cache architecture. The accounting cache first accesses part of the ways of a set associative cache, known as a primary access. If there is a miss, then the cache accesses the other ways, known as a secondary access. A swap between the primary and secondary accesses is needed when there is a miss in the primary and a hit in the secondary access. Energy is saved on a hit during the primary access. Way-prediction was first proposed to reduce the cache access latency. The power efficiency of way-prediction techniques were discussed later [16, 17].

1.3.3 Limitations of Way-Prediction Schemes

Way-prediction designs have been proposed for fast L1 caches. There are several reasons for which the original way-prediction idea cannot be applied directly to large L2 caches.

First, in way-prediction designs, the predicted way number must be made available before the actual data address is generated [13, 16, 14]. We call this an *out-cache*[1] feature for way-prediction designs. As large L2 caches are typically physically-indexed caches, a virtual to physical address translation must be conducted before the address can be presented to the way-prediction hardware. The way-prediction mechanism sitting between the TLB and the L2 cache will add extra delay to the critical path. Second, L2 caches are unified caches, where most of the references come from L1 data cache misses. MRU based prediction does not always work well with data references [13, 14]. Third, the cache line size of the L2 cache is large. In Intel P4 processors, the L2 cache line size is 128 bytes. This means exchanging the locations of cache lines is prohibitively expensive. Finally, way-prediction introduces non-unified cache access latency. The processor must be redesigned to take the advantage of non-unified L2 cache latency.

[1]Existing in-cache prediction schemes, including the bit-difference cache [18] and the partial comparison cache [19], fetch the prediction information in parallel with data array accesses, which saves no power. They also have performance limitations [12, 20], which makes them less popular.

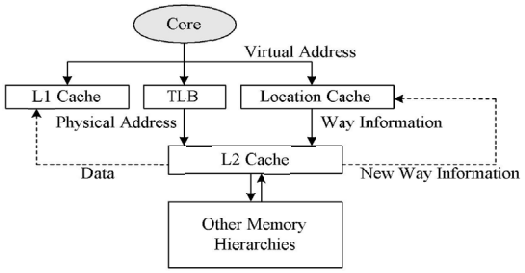

Figure 1: The Location Cache System

This paper examines the popular MRU information used by existing way-prediction mechanisms. We show that it is difficult to directly use existing way-prediction on L2 caches. We propose to use another kind of information, namely *address affinity* to provide accurate location information for L2 cache references. The proposed cache design reduces cache access power while improving the performance, compared with a conventional set-associative L2 cache.

The rest of this paper is organized as follows: Section 2 introduces the architecture of the location cache system. Section 3 presents simulation results on access delay and power consumption of the proposed hardware. Section 4 studies the performance and power efficiency of the proposed system. We conclude the paper in Section 5.

2. OUR SOLUTION

We propose a new cache architecture called the location cache. Figure 1 illustrates its structure. The location cache is a small virtually-indexed direct-mapped cache. It caches the location information (the way number in one set a memory reference falls into). This cache works in parallel with the TLB and the L1 cache. On an L1 cache miss, the physical address translated by the TLB and the way information of the reference are both presented to the L2 cache. The L2 cache is then accessed as a direct-mapped cache. There can be a miss in the location cache, then the L2 cache is accessed as a conventional set-associative cache. As opposed to way-prediction information, the cached location is not a prediction. Thus when there is a hit, both time and power will be saved. Even if there is a miss, we do not see any extra delay penalty as seen in way- prediction caches.

Caching the position, unlike caching the data itself, will not cause coherence problems in multi-processor systems. Although the snooping mechanism may modify the data stored in the L2 cache, the location will not change. Also, even if a cache line is replaced in the L2 cache, the way information stored in the location cache will not generate a fault.

One interesting issue arises here: the locations for which references should be cached? The location cache should catch the references which turn out to be L1 misses. A recency based strategy is

Cache conf.	16KB 4way	16KB 1way	32 entry	64 entry	128 entry	256 entry	512 entry	1024 entry
Access Delay	1.1674	1	0.7340	0.7574	0.7919	0.8168	0.8817	0.9772

Table 1: Normalized access delays for various location cache configurations

not suitable because the recent accesses to the L2 caches are very likely to be cached in the L1 caches. The equation below defines the optimal coverage of the location cache.

```
  Opt. Coverage =  L2  coverage -
L1  coverage
```

As the indexing rules of L1 and L2 caches are different, this optimal coverage is not reachable. Fortunately, the memory locations are usually referenced in sequences or strides. Whenever a reference to the L2 cache is generated, we calculate the location of the next cache line and feed it into the location cache.

The proposed cache system works in the following way. The location cache is accessed in parallel with the L1 caches. If the L1 cache sees a hit, then the results from the location cache is discarded. If there is a miss in the L1 cache, and there is a hit in the location cache, the L2 cache is accessed as a direct-mapped cache. If both the L1 cache and the location cache see a miss, then the L2 cache is accessed as a traditional L2 cache. The tags of the L2 cache is duplicated. We call the duplicated tag arrays of the L2 cache *location tag arrays*. When the L2 cache is accessed, the location tag arrays are accessed to generate the location information for the next memory reference. The generated location information is then sent to and stored in the location cache.

3. LOCATION CACHE SYSTEM HARDWARE

The hardware design of the location cache system consists of two parts. The first part is a small location cache which works in parallel with L1 caches. The second part includes the modification to the L2 cache. We need to either make a duplication of the tags or double the number of ports to L2 tags.

3.1 Location Cache Architecture

The location cache should be small so that its access latency can be covered by the L1 cache access or the TLB translation. Table 1 lists the access delays of the location cache for various cache sizes. The L1 cache is a 16KB 4-way set-associative cache, with a cache line size of 64-bytes, implemented with a 0.13μm technology. The results were produced using the CACTI3.2 simulator. We chose the access delay of a 16KB direct-mapped cache as the baseline, which is the best-case delay when a way-prediction mechanism is implemented in the L1 cache. We normalized the baseline delay to 1. It is observed that a location cache with up-to 1024 entries has shorter access latency than the L1 cache. Though the organization of the location cache is similar to that of a direct-mapped cache, there is a small change in the indexing rule. The block offset is 7 bit as the cache line size for the simulated L2 cache is 128 bytes. Thus the width of the tag is smaller for the location cache, compared with a regular cache.

3.2 New L2 Cache Architecture

Figures 2 and 3 compares the structures of a conventional 2-way set-associative cache and a 2-way set-associative cache supported by a location cache. Compared to a regular cache design, the modification is minor. Note that we need to double the tags (or the number of ports to the tag) because when the original tags are compared to validate the accesses, a spare set of tag is compared to generate the future location information. This idea is similar to the phased cache. The difference is that we overlap the tag comparison for future references with existing cache reference and use the location cache to store such location information. The simulated cache geometry parameters were optimized for the set-associative cache. The simulation results show that the access latency for a direct-mapped hit is 40% faster than a set-associative hit.

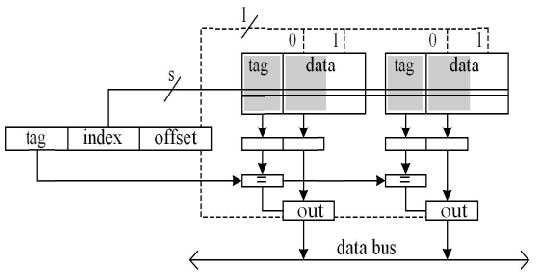

Figure 2: A Conventional 2-Way Set-Associative L2 Cache (with two subbanks)

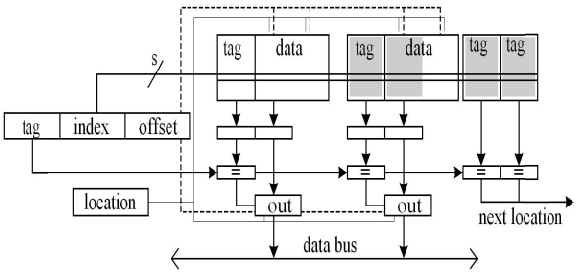

Figure 3: A 2-Way Set-Associative L2 Cache in the Location Cache System

Although the extra hardware employed by the location cache design does not introduce extra delay on the memory reference critical path, it does introduce extra power consumption. The extra power consumption comes from the small location cache and the duplicated tag arrays. We summarize the power consumption of the control circuits as well as the original L2 cache in Table 2. The simulated L2 cache is a 512KB 8-way set-associative cache, with the cache line size of 128 bytes. We normalize the power consumption for the tag access of a direct-mapped hit to one. Comparing to the L2 cache power consumption, the location cache consumes a small amount of power. However, as the location cache is triggered much often than the L2 cache, its power consumption cannot be

ignored. The total chip area of the proposed location cache system (with duplicated tag and a location cache of 1024 entries) is only 1.39% larger than that of the original cache system.

4. SIMULATION RESULTS

To evaluate the performance and power consumption of the location cache system on real-world workloads, we used the simplescalar3.0b simulator to simulate a 4-issue multi-scalar CPU. Separate 16KB L1 instruction and L1 data caches were simulated. They are both 4-way set-associative caches with a cache line size of 64 bytes. The L2 cache is a 512KB 8-way set-associative unified cache with the cache line size of 128 bytes. The L2 cache has 8 banks. The bus between the L1 and L2 caches is 512-bit wide. The L1 cache delay is 1 cycle. The L2 cache delay is 6 cycles. A direct-mapped hit in the L2 cache has a delay of 4 cycles. Memory function units have 4 ports. This system configuration is similar to that of an Intel P4 processor. All 26 spec2000 benchmark applications were simulated with the reference input set, for one billion memory references.

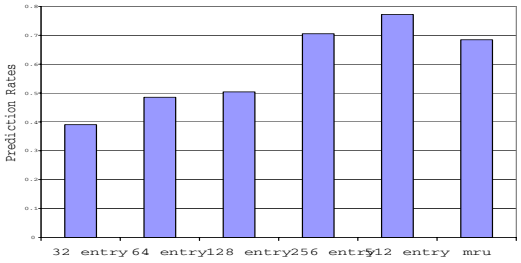

Figure 4: Prediction Rates

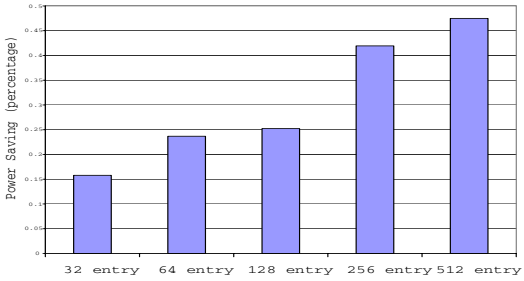

Figure 5: Power Savings

As most of the L2 cache references are generated by L1 data cache misses, we propose to access the location cache only when the L1 data cache is accessed. Due to the page limitation, we only present the average results based on all 26 benchmark applications. Figure 4 presents the prediction accuracy with respect to variation in the numbers of location cache entries. We also provide the accuracy of the MRU based prediction scheme. Its hardware complexity is similar to a location cache with 512 entries. It can be observed that prediction based on location information has better prediction rate. When the number of location cache entries is larger than 256, the prediction rate improved drastically, indicating the coverage of the location cache is very important.

Figure 5 presents the power savings when different number of location cache entries are used. The power saving is closely related to the prediction rate and the L1 data cache miss rate. When the L1 data has very high hit rates, the location cache itself will consume a lot of power, sometimes even more than the power saved in the L2 cache system. In our experiment, the average L1 data cache miss rate is 4.9%. It can be observed that in an average, as much as 47.5% of L2 accessing power can saved by the location cache design.

Although our primary concern is to save power, the location cache can also improve the performance. If the processor is able to support non-unified L2 cache access latency, the location cache design can also improve the performance of the L2 cache. We use the average cache access latency, which is the time the cache is busy for each reference, to evaluate the performance. In general, a smaller cache latency is preferred. The cache latency for a conventional set-associative cache is 6 cycles. It is 4 cycles for that of a direct-mapped cache. We summarize the simulation results in Figure 6. It can be observed that the location cache design can achieve an average access latency of 4.5 cycles, which is 25% smaller than the original set-associative cache. We also provide the performance of the MRU based prediction scheme[2]. Though the MRU scheme has good prediction rates, its performance is not as good as that of the location cache, as prediction performs poorly with applications like ammp, art and galgel. The long worst-case latency for MRU way-prediction introduces significant performance degradation. The location cache design, in any situation, will not hurt the performance.

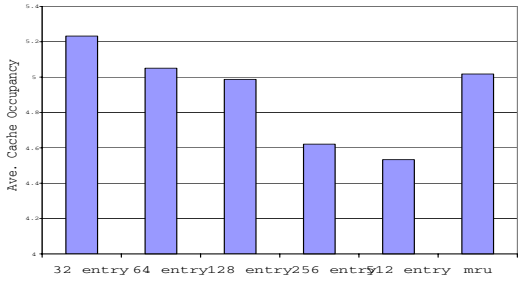

Figure 6: Average Cache Access Latency

5. CONCLUSION

Caches have accounted for a significant fraction of the overall chip dynamic energy. For instance, the Alpha 21264 CPU consumes about 16% energy in caches, even without on-chip L2 caches. The Pentium Pro consumes about 33% of the chip power in instruction fetch and d-cache together. Large L2 caches typically consume even more power. For example, the Intel 200 MHz P2 processor with 256 KB L2 Cache has a typical thermal design power consumption of 27.3*W*. When the L2-Cache size is increased to 1MB, the typical power consumption for the same processor jumps

[2]In real situation, the MRU information is not available. Even larger average latency will be introduced in order to wait for the prediction information to be generated

L2 Data 8-Way	L2 Tag 8-Way	L2 Data 1-Way	L2 Tag 1-Way	32 entry	64 entry	128 entry	256 entry	512 entry
97.2606	1.4284	16.7114	1	0.7362	0.7200	0.7356	0.7206	0.7668

Table 2: Normalized access power consumption for location cache components

to 47W, a 72% increase. In today's computer systems, even larger caches are packed into the processor. Thus lowering the power-consumption of the cache system is an important research topic.

There are various strategies for reducing the cache energy with varying performance trade-offs, including turning off parts of the cache [21, 22, 23, 4, 24], feeding the cache with different voltages [25], instruction scheduling techniques [1], low power designs based on value frequencies [26] and strategies introduced in Section 1.

The location cache attacks the problem from a different angle and is able to reduce up-to 47% of L2 cache access power with a 25% reduction in average cache occupancy. To further bring down the system power consumption, the location cache can be placed below the TLB level and be accessed only when L1 misses are observed. Although the wrost-case L2 cache access delay will be longer, the overall average L2 cache access lantency will still be faster than the traditional L2 cache design.

As the location design explores information which is not used by other designs, the location cache mechanism can be used together with other strategies to further reduce power consumption and improve the performance.

6. ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under Career Award CCR-9984852 and ACI-0232647, and the Ohio Board of Regents. We would also like to thank the anonymous reviewers for their helpful comments on drafts of this paper.

7. REFERENCES

[1] C. Su and A. Despain, "Cache design tradeoffs for power and performance optimization: A case study," in *International Symposium on Low Power Electronics and Design*, pp. 63–68, 1997.

[2] K. Ghose and M. B. Kamble, "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation," in *International Symposium on Low Power Electronics and Design*, pp. 70 – 75, 1999.

[3] U. Ko, P. T. Balsara, and A. K. Nanda, "Energy optimization of multi-level process cache architectures," in *Prod. of the 1995 Internation Symposium on Low Power Design*, pp. 45 – 49, 1995.

[4] D. H. Albonesi, "Selective cache ways: on-demand cache resource allocation," in *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 248–259, 1999.

[5] J. Kin, M. Gupta, and W. Mangione-Smith, "The filter cache: an energy efficient memory structure," in *30th Annual International Symposium on Microarchitecture (Micro '97)*, pp. 184–193, December 1997.

[6] A. Hasegawa, I. Kawasaki, K. Yamada, S. Yoshioka, S. Kawasaki, and P. Biswas, "Sh3: High code density, low power," *IEEE Micro*, vol. 15, pp. 11–19, December 1995.

[7] T. Lyon, E. Delano, C. McNairy, and D. Mulla, "Data cache design considerations for the itanium2 processor," in *Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02)*, pp. 356 –362, 2002.

[8] C. Zhang, F. Vahlid, and W. Najjar, "A highly configurable cache architecture for embedded systems," in *The Prod. of the 30th Annual International Symposium on Computer Architecture (ISCA03)*, pp. 125–136, 2003.

[9] G. Memik, G. Reinman, and W. Mangio-Smith, "Just say no: Benefits of early cache miss determination," in *Prod. of the Ninth International Symposium on High-Performance Computer Architecture*, pp. 307–316, 2003.

[10] A. Agarwal, J. Hennesy, and M. Horowits, "Cache performance of operating systems and multiprogramming," in *ACM Transactions on Computer Systems*, pp. 393–431, November 1988.

[11] A. Agarwal and S. D. Pudar, "Column-associative caches: a technique for reducing the miss rate of direct-mapped caches," in *Proc. of the 35th annual International Syposium on Computer Architecture (ISCA)*, pp. 179–190, 1993.

[12] J. H. Chang, H. Chao, and K. So., "Cache design of a sub-micron cmos system/370," in *14th Annual International Symposium on Computer Architecture, SIGARCH Newsletter*, pp. 208–213, June 1987.

[13] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in *Proc. of the 2nd IEEE Symposium on High-Performance Computer Architecture (HPCA '96)*, pp. 244–254, 1996.

[14] T. N. Vijaykumar, "Reactive-associative caches," in *International Conference on Parallel Architectures and Compiler Techinques (PACT'01)*, pp. 49–61, 2001.

[15] S.Dropsho, A. Buyuktonsunoglu, D. H. A. R. Balasubramonian, G. S. S. Dwarkadas, G. Magklis, and M. Scott, "Integrating adaptive on-chip storage structures for reduced dynamic power," in *International Conference on Parallel Architectures and Compilation Techniques (PACT02)*, pp. 190–202, 2002.

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

[17] M. Powell, A. Agrawal, T. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *34th Annual International Symposium on Microarchitecture (MICRO'01)*, pp. 54–65, December 2001.

[18] T. Juan, T. Lang, and J. J. Navarro, "The difference-bit cache," in *Proc. of the 23rd annual international symposium on computer architecture*, pp. 114–120, 1996.

[19] L. Liu, "Cache designs with partial address matching," in *Proc. of the 27 Internaltional symposium on microarchitecture*, pp. 128–136, 1994.

[20] K. A., N. Chander, P. S., and J. L., "Modeling and analysis of the difference-bit cache," in *Proc. of the 8th Great Lakes Symposium on VLSI*, pp. 140–145, 1998.

[21] Z. Hu, S. Kaxiras, and M. Martonosi, "Let caches decay: reducing leakage energy via exploitation of cache generational behavior," *ACM Transactions on Computer Systems*, vol. 20, no. 11, pp. 161–190, 2002.

[22] M. Zhang and K. Asanovic, "Fine-grain cam-tag cache resizing using miss tags," in *Proceedings of the 2002 international symposium on Low power electronics and design(ISLPED'02)*, pp. 130–135, 2002.

[23] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: Simple techniques for reducing leakage power," in *International Symposium on Computer Architecture*, pp. 148–158, June 2002.

[24] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte, "Adaptive mode control: A static-power-efficient cache design," in *International Conference on Parallel Architectures and Compilation Techniques (PACT'01)*, (Barcelona, Spain), pp. 61–73, September 2001.

[25] V. Moshnyaga and H. Tsuji, "Cache energy reduction by dual voltage supply," in *The 2001 IEEE International Symposium on Circuits and Systems (ISCAS 2001)*, pp. 922–925, May 2001.

[26] J. Yang and R. Gupta, "Energy efficient frequent value data cache design," in *IEEE/ACM 35th International Symposium on Microarchitecture (MICRO)*, pp. 197–207, nov. 2002.