Fine-Grain CAM-Tag Cache Resizing Using Miss Tags

Michael Zhang
MIT Laboratory for Computer Science
200 Technology Square
Cambridge, MA 02139
rzhang@cag.lcs.mit.edu

Krste Asanović
MIT Laboratory for Computer Science
200 Technology Square
Cambridge, MA 02139
krste@cag.lcs.mit.edu

ABSTRACT

A new dynamic cache resizing scheme for low-power CAM-tag caches is introduced. A control algorithm that is only activated on cache misses uses a duplicate set of tags, the *miss tags*, to minimize active cache size while sustaining close to the same hit rate as a full size cache. The cache partitioning mechanism saves both switching and leakage energy in unused partitions with little impact on cycle time. Simulation results show that the scheme saves 28–56% of data cache energy and 34–49% of instruction cache energy with minimal performance impact.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Associative Memory, Cache Memory, Primary Memory*

General Terms

Design

Keywords

Content-Addressable-Memory, Low-Power, Cache Resizing, Energy Efficiency, Leakage Current

1. INTRODUCTION

Energy dissipation has emerged as one of the primary constraints for microprocessor designers. In most microprocessor designs, caches dissipate a significant fraction of total power. For example, the Alpha 21264 dissipates 16% [12] and the StrongArm dissipates more than 43% [19] of overall power in caches. As a result, there has been great interest in reducing cache power consumption.

Initial cache energy reduction techniques focused on dynamic switching power [1, 2, 3, 4, 7, 10, 13, 22]. With technology scaling, leakage current is increasing exponentially, and more attention has been paid to leakage power reduction [9, 11, 15, 16, 18, 20].

ISLPED'02, August 12-14, 2002, Monterey, California, USA.

One approach for reducing cache power consumption is cache resizing, where the active size of the cache is reduced to match the current working set. Previously reported cache resizing schemes can be categorized by the mechanism used to activate and deactivate cache entries, and by the control policy used to select the active partition. Some schemes deactivate cache entries line by line [9, 11], while others deactivate the cache by sets, ways, or both [1, 16, 20]. The control policy used to select the active set can be off-line, where the working set is statically determined by profiling the application [1], or on-line, where the working set is dynamically determined as the application executes [9, 11, 16, 20].

Previous cache resizing techniques are designed for RAM-tag caches, where cache tags are held in RAM structures. However, commercial low-power microprocessors use CAM-tag caches, where the cache tags are held in Content Addressable Memory [14, 19]. CAM-tag caches are popular in low-power processors because they provide high associativity, which avoids expensive cache misses, and results in lower overall energy [23].

This paper introduces *miss tag resizing* (MTR), a new cache resizing scheme for CAM-tag caches. MTR uses hierarchical bitlines to divide each cache subbank into small way partitions, such that switching and leakage power is only dissipated in active ways. In addition, individual cache lines within an active partition can be disabled to further reduce leakage power. Because CAM-tag caches have high associativity (32-way for the design simulated), partitioning the cache by way gives much finer grain control over cache size compared to RAM-tag way activation [1]. It also avoids the data remapping problem inherent in set resizing schemes [16]. In addition, the scheme proposed here adapts associativity independently in each sub-bank, thereby allowing total cache size to be varied a single line at a time. Resizing of different subbanks is spaced evenly in time so that at most a single dirty line needs to be written back for a resize event.

The size of an MTR cache is governed using an on-line control policy which aims to reduce the cache size to the smallest value that will give a minimal miss rate increase compared to the full sized cache. The control policy uses an extra set of tags, the *miss tags*, which are only accessed on misses to determine if a full-sized cache would have hit. Because the miss tags are only accessed on misses, they add no additional switching energy to hits and can be implemented using slower, denser, and less leaky transistors, e.g., high V_T or long channel transistors. The main penalty for using miss

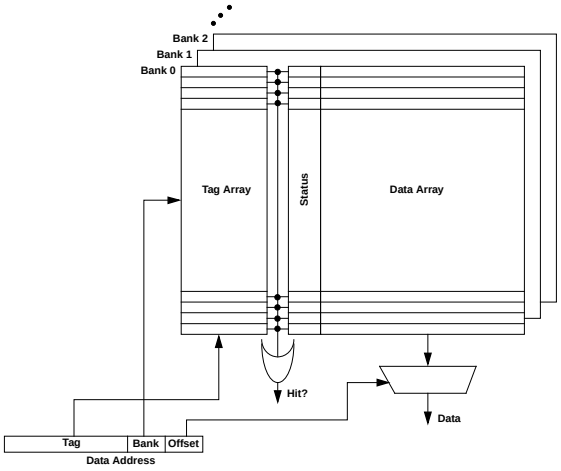

Figure 1: CAM-tag cache organization.

tags is the additional area overhead, which we estimate at around 10% depending on actual layout styles.

The rest of the paper is organized as follows. Section 2 reviews related work on cache resizing. Section 3 presents the MTR algorithm. Section 4 describes the hardware modifications for energy reduction. Section 5 gives results for active cache size reductions. Section 6 presents the energy savings achieved by MTR. And Section 7 concludes.

2. RELATED WORK

In this section, we discuss existing cache resizing techniques and cache line deactivation techniques. An off-line resizing technique was proposed in [1]. Applications are profiled prior to execution to determine an optimal set-associativity. At run-time, cache ways of the L1 RAM-tag set-associative cache are turned off according to the profile information. This technique reduces both switching and leakage energy by powering down the entire cache way. However, it does not adapt to varying cache usage during different phases of the program execution. As we will show later, many benchmarks have working sets that vary widely during various phases of execution. Furthermore, these static techniques do not work well for multi-programmed machines, where working set size also varies as a function of the active process. The DRI I-cache [16] is an on-line resizing technique that resizes a RAM-tag instruction cache by measuring the miss rate and keeping it under a preset threshold. This performance threshold is set to a typical cache miss rate prior to execution, which does not adapt to program execution phases. Line deactivation techniques are similar to the above resizing techniques. These techniques usually turn off individual cache lines that are not necessarily contiguous. In cache decay [11], a per-line counter tracks the usage of each cache line. Lines with no recent uses are turned off. This technique eliminates the static energy of dead lines but does not reduce switching energy. Adaptive mode control (AMC) [9] resizes a RAM-tag cache using a technique similar to cache decay. AMC keeps all tags turned on. An ideal miss rate is obtained by searching the entire tag array, and an actual miss rate is obtained by only searching the tags of all the active lines. When these two miss rates differ by more than a preset performance factor, the resize

```
cache_access(action, addr_tag, addr_offset, data) {
  if (addr_tag in tag_array) {   /* hit case */
    if (action == Read) {
      return data_array[addr_tag, addr_offset];
    } else {
      data_array[addr_tag, addr_offset] = data;
    }
    return hit;
  } else { /* miss case */
    /* fetch data from L2 and update the cache */
    fetch_from_memory(addr_tag, addr_offset);
    /* check whether tag is in MTR tag array */
    if (addr_tag in MTR_tag_array) {
      /* if tag is found in MTR, */
      /* increment MTR hit counter */
      MTR_hits++;
    } else {
      /* otherwise, write the tag into MTR array */
      update_MTR_tag_content(addr_tag);
    }
    return miss;
  }
}

cache_resize() {
  if (MTR_hits > HI_BOUND) {
    upsize();
  } else if (MTR_hits < LO_BOUND) {
    downsize();
  } else {
    do_nothing();
  }
  /* reset the MTR hit counter for */
  /* next resizing interval */
  MTR_hits = 0;
}
```

Figure 2: Pseudo-code for MTR.

interval is adjusted. This technique eliminates the need for presetting the desired miss rates, but only reduces leakage power in the data arrays. Tag array lookup, however, is a significant portion of the cache access energy, especially for CAM-tag caches. In [20], various design choices are compared to evaluate the usefulness of resizable caches. On average, over 50% cache size reduction is achieved with either selective ways [1] or selective sets [16]. Turning off portions of the cache generally discards the stored data, thus increasing miss rate and the number of L2 accesses. In [8], the effect of L2 energy overhead is examined. Our MTR scheme is similar to AMC in that we resize based on the difference between the full cache hit rate and the reduced cache hit rate. However, we employ a separate set of tags that are only accessed on misses to gather the full cache hit rate. This avoids additional switching and leakage power in the regular CAM tags. Also, we use the miss rate difference to control a fine-grain partitionable cache which can save switching as well as leakage power. Another problem with previous partitioning schemes is that when applied to a data cache, they can generate a large number of dirty line writebacks in a short time interval when a set or way is deactivated, or when a decay interval elapses. These write back bursts add to cache control complexity and can cause additional performance degradation. MTR performs way deactivation within a highly associative cache one line at a time, thus avoids write back bursts.

3. MISS-TAG RESIZING TECHNIQUE

Figure 1 shows a typical CAM-tag cache organization. The entire cache is divided into subbanks, each consisting of a tag array and a data array, where a subbank is a cache *set*. Within each set are the cache *ways*. The tags are stored in CAM structures to give high associativity at low power. During each cache access, one subbank (set) of the cache is accessed and the tag is broadcast to the entire tag array. A matched tag results in a hit and triggers the appropriate wordline to enable the access.

To implement MTR, we add an extra set of tags, the *miss-tags*, which act as the tags of a fixed-size cache. These tags keep track of what the cache contents would have been if the cache was always full size. During a regular cache miss, we consult the miss-tag arrays to see whether having a full cache could have avoided the miss. A per-subbank counter is used to record the number of miss-tag hits, which is precisely the difference between the number of misses in the down-sized cache and in a full size cache. A large difference in the miss rates suggests that having a larger cache will reduce the miss rate; a small difference indicates that perhaps a smaller cache would be adequate. Two scenarios could explain a small difference in miss rate between the full size and reduced size caches. First, there are no misses in the regular tags, indicating that the program has a small working set. In the second scenario, there are many misses in the regular tags, most of which also miss in the miss tags. This suggests that the program has little temporal locality, such as a data streaming application.

The resizing decision is based on the difference in miss rates between the active tags and the miss tags. The pseudo-code in Figure 2 illustrates the resizing control loop of MTR. There are three parameters in the MTR scheme: *miss lower bound*, *miss upper bound*, and *resize interval*. In Section 5.2, we will discuss the choices of resizing parameters in detail. Each subbank is independently resized once during each resizing interval. Resizing events are spread out evenly within each interval so that only one subbank resizes at a time to minimize writeback traffic burst to the lower levels of the memory hierarchy.

4. HARDWARE MODIFICATION

Figure 3 details three circuit techniques used by MTR. For the SRAM cells in both data and tag arrays, we use the Gated-Vdd technique [15] to reduce leakage energy by adding an N-type stack transistor. When signal `Line_On` is turned off, it virtually eliminates leakage current in the SRAM cells. We also use the leakage-biased bitline (LBB) technique proposed in [17] to reduce the leakage in SRAM bitlines, CAM bitlines and search lines, and CAM match lines. The leakage power of the circuit depends on the actual voltage of these heavily capacitive lines. The LBB technique turns off the precharge of these lines, allowing them to self-bias their voltage levels to the optimal values, at which leakage power is minimized using leakage currents. The cache subbanks are divided into eight equal partitions using hierarchical bitlines [7]. The `Partition_On` bits are used to control the activation of each partition. An inactive partition consumes no switching energy and minimal leakage energy.

Since the miss-tags are only used during a cache miss, we can use slow, low-leakage components without incurring

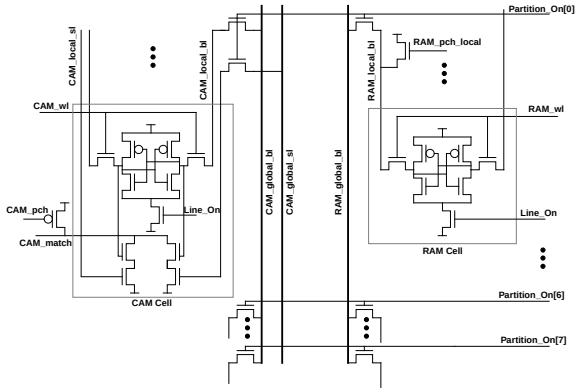

Figure 3: Energy reduction techniques used by MTR: Gated-Vdd for SRAM cell leakage reduction; Leakage-Bias for CAM match line; hierarchical bitlines for subbank partitioning.

delay overhead. The energy overhead of miss-tag accesses is added to L2 access energy and is discussed in Section 6. The area overhead can be reduced by using a denser layout for the tags, for example, adopting a hybrid RAM-CAM structure to reduce the number of match comparators.

5. CACHE SIZE REDUCTION RESULTS

In order to evaluate MTR, we modified the SimpleScalar [5] simulator. We modeled an in-order single issue core in our experiments. The benchmark set we used is a subset of `SpecINT2000` and `SpecFP2000`, each running for 1.5 billion cycles with the reference inputs. We chose a typical low-power cache configuration [14] as a baseline. It is a 32KB cache implemented in 32 1KB subbanks. Each subbank consists of 32 cache lines of 32 bytes. The cache is 32-way set-associative with a FIFO replacement policy in each subbank.

One unary encoded resizing pointer per subbank is used to control which cache lines to activate/deactivate, similar to the XScale FIFO pointer [14]. When a cache is downsized, only the last active line is turned off. When it is upsized, however, the entire partition where the last active line resides is turned on. If all the lines in the entire partition are already active, the next partition is turned on. When all the lines in a partition are inactive, the partition is turned off. To avoid thrashing with small cache sizes, we set the minimum cache size to be one partition.

5.1 Baseline Case

We implemented a baseline resizing technique to compare against the miss tags scheme. This baseline technique works exactly like MTR *except* it compares the *actual cache miss rate* with the miss bounds to make resizing decisions, similar to DRI I-cache [16]. We will refer to this baseline technique as *Miss-Rate-Based-Resizing* (MRBR).

5.2 Impact of Resizing Parameters

From simulation results, we found that no individual parameter has a large impact on resizing performance. The most important parameter, rather, is the ratio of the miss upper/lower bounds to the resize interval. For example, setting the miss bound of 5 to 10 misses for a 32k resizing interval yields similar results for a range of 10 to 20 misses for

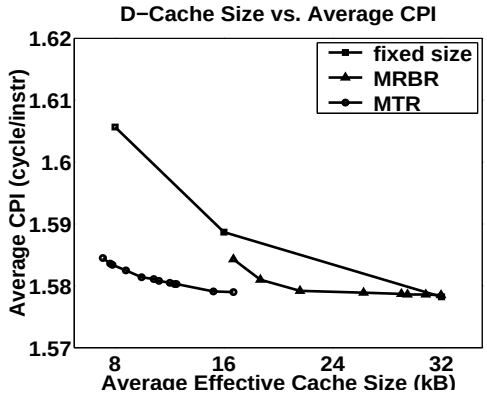

Figure 4: CPI versus effective cache size for L1 data cache. MTR gives the smallest effective cache size for a given CPI.

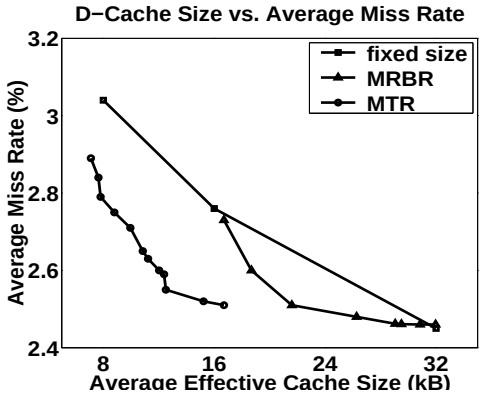

Figure 5: Miss Rate versus effective cache size for L1 data cache. MTR gives the smallest effective cache size for a given miss rate.

a 64k resizing interval. Simulations show that for larger resize intervals, the number of writebacks decrease. However, when the resize interval is too large, MTR starts to yield sub-optimal results. We have found that resize intervals of 128K references worked well for the benchmarks studied, i.e., resize every 128k memory references.

5.3 Data Cache Resizing Results

Figure 4 shows the resizing results for the L1 data cache. Each data point (effective cache size and CPI pair) is obtained by varying the miss bounds and resizing interval length to obtain the optimal CPI for a given effective cache size. Average cache size is calculated by averaging the percentage of active partitions in each resizing period. In order to verify that both resizing techniques work better than a fixed-size cache, we simulated the CPI of fixed-size caches of sizes 32KB, 16KB, and 8KB. This figure shows that for the same CPI, MTR yields much smaller effective cache sizes. We limited ourselves to considering configurations that yield less than a 2% CPI increase to ensure MTR does not incur a large performance penalty. Parameters were varied to show the trade off between effective cache size and performance. For the same effective cache size, MTR performs much better than the baseline technique. Figure 5 further supports the above result. MTR introduces less than a 16% increase in the largest fixed cache miss rate. Again, for the same effective cache size, MTR has the lowest miss rate. On average, MTR uses less than an 8KB effective cache size while increasing the CPI by less than 1.5%.

Figure 6 shows how the effective cache size and the actual miss rates change over time with MTR. The figures on the left-hand side show the effective cache size over time. We observe two different behaviors. Benchmarks `164.gzip`, `177.mesa`, `183.equake`, `197.parser`, and `256.bzip2` demonstrate MTR's ability to adapt to different phases of the execution with varying cache usage. For the rest of the benchmarks, cache usage is constant throughout the execution. MTR is able to find the optimal size for each benchmark without prior profiling information. The figures on the right-hand-side show how the miss rates change throughout the execution. We observe that an increase in the miss rate is countered by an increase in cache size, which in return, reduces miss rate.

5.4 Instruction Cache Resizing Results

For our benchmark set, the instruction cache has extremely low miss rates. Therefore, it is easier to find a common reference miss rate for a large set of benchmarks. For all the benchmarks we used in this paper, the baseline resizing technique and MTR have similar performance. Both of them outperform the fixed size instruction cache. Figures 9 and 8 show that MTR uses an effective cache size of less than 12KB while introducing, on average, less than 12% increase in miss rate and 1.4% increase in CPI.

6. ENERGY REDUCTION RESULTS

In this section, we present the energy savings obtained by MTR. The energy consumption figures are obtained through HSpice simulation of extracted layout from Cadence [6] using TSMC 0.25μm technology [21]. The cache design has been significantly optimized for low power, including divided word lines and low-swing bitlines. Table 1 shows the different energy components of this CAM-tag cache. MTR reduces the data array and CAM-tag array access energy but not decoding energy. Since the actual percentage of cache leakage power in the total cache power can vary significantly due to process technology, operating temperatures and voltages, among other factors, we quantify cache leakage as a percentage of total cache power, and demonstrate the savings across a range of possible values. We perform a similar sensitivity analysis for L2 cache energy by quantifying L2 access energy as a multiple of L1 access energy and give results for a range of values. We include the search energy for the miss-tags as part of L2 energy. The energy reduction is calculated as

	L1 switching energy reduction × % of switching energy
+	L1 leakage energy reduction × % of leakage energy
−	Miss Rate Increase × L2 access energy

Figures 10 and 11 show the energy reduction of data and instruction cache. The x-axis represents the percentage of leakage energy in the total energy consumption. The y-axis represents the energy savings. From previous experiments, we use resizing parameters such that the effective data cache size is 8KB and effective instruction cache is 12KB. These parameters are chosen to minimize the performance impact

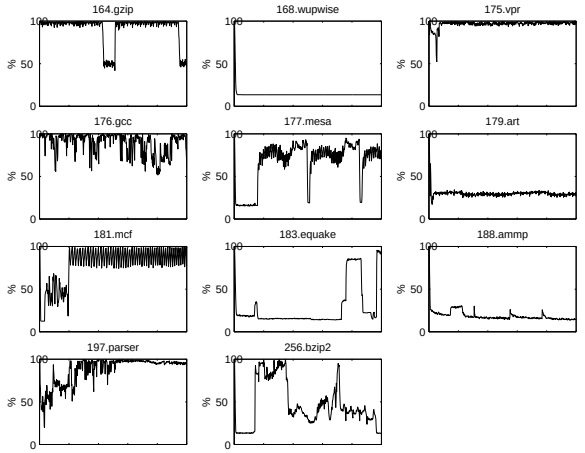

Figure 6: Different effective cache sizes during different phases of a 32KB data cache determined by MTR. The x-axis represents 0 to 1.5 billion cycles.

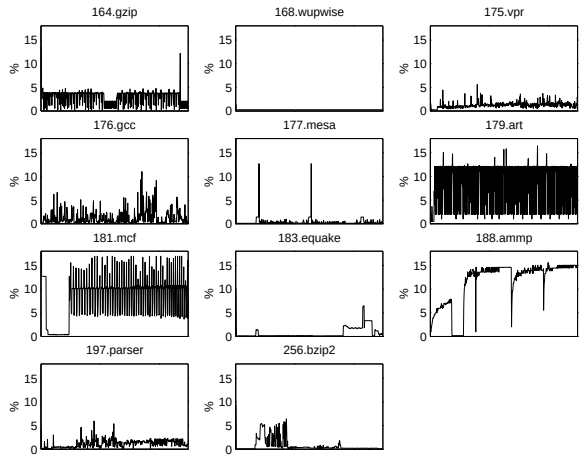

Figure 7: Cache miss rates during different phases of a 32KB data cache determined by MTR. The x-axis represents 0 to 1.5 billion cycles.

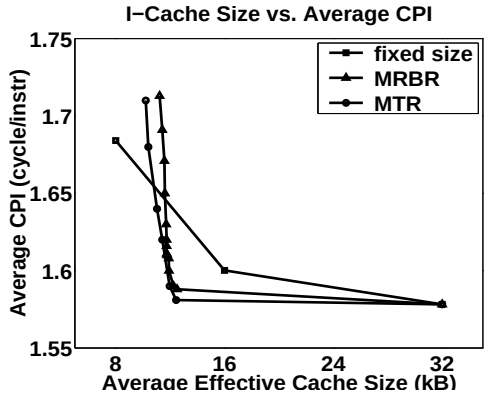

Figure 8: CPI versus effective cache size for L1 instruction cache. MTR and MRBR have similar performance.

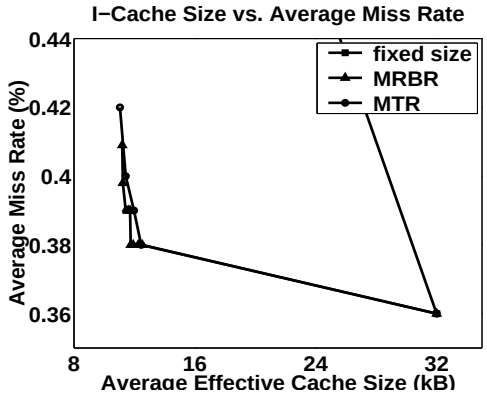

Figure 9: Miss rate versus effective cache size for L1 instruction cache. MTR and MRBR have similar performance.

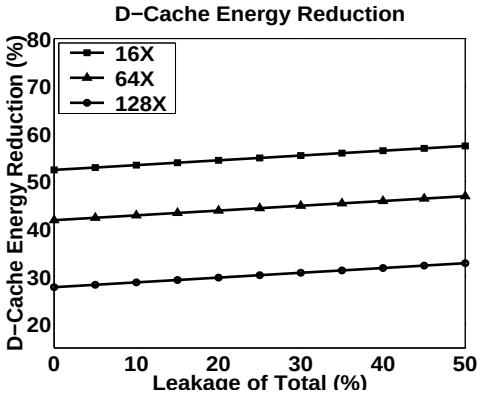

Figure 10: Data cache energy savings. X-axis represent the percentage of leakage energy of total energy. Y-axis represents savings. Each curve represents a different L2 access energy quantified as a factor of L1 write access energy.

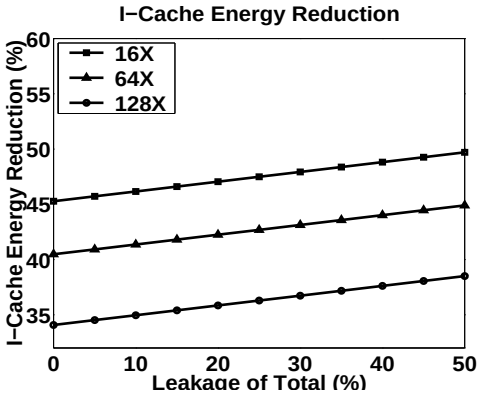

Figure 11: Instruction cache energy savings. X-axis represent the percentage of leakage energy of total energy. Y-axis represents savings. Each curve represents a different L2 access energy quantified as a factor of L1 write access energy.

Table 1: Energy components of CAM-tag cache in TSMC 0.25 μm technology.. A $\surd$ means the read or write access performs that operation, thus uses that energy component.

Operation	Energy (pJ)	Read	Write
CAM-Array Search	57.1	$\surd$	$\surd$
Data-Array Read	26.2	$\surd$	
Data-Array Write	53.5		$\surd$
Decoding & I/O	12.2	$\surd$	$\surd$
Total		95.5 pJ	122.8 pJ

while turning off the maximum number of partitions in the cache.

Each different curve represents the energy savings of a specific L2 access energy. We chose an range of L2 access energy, from 16× to 128× of the L1 write access energy. For data cache, MTR reduces energy by 28%, when there is no leakage energy and L2 penalty is 128× of L1 write access energy, to 56%, when 50% of the cache energy is leakage and L2 penalty is 16× of L1 access energy. Similarly, MTR reduction ranges from 34% to 49% for instruction cache depending on leakage percentage and L2 penalty.

7. CONCLUSION

In this paper, we presented MTR, a dynamic cache resizing technique for CAM-tag caches. The dynamic control mechanism of MTR uses a set of duplicate miss tags to keep track of the miss rate as if the entire cache was used. Resizing decisions are made according to the difference in the actual miss rate and the miss rate of the miss-tags. The control mechanism is only activated on misses, thereby saving energy and allowing the duplicate tags to be implemented in slower and denser logic using low leakage transistors. The cache partitioning mechanism saves both switching and leakage energy in unused partitions, and allows resizing at a single line granularity. The subbanks are resized independently in non-overlapping phases to avoid write back bursts. With around 10% area overhead, MTR reduces 28–56% of data cache energy and 34–49% of instruction cache energy, where the baseline caches were highly optimized for low-power but fixed-size operation.

8. ACKNOWLEDGMENTS

We would like to thank members of the MIT SCALE group for feedback and comments on earlier drafts of this paper. We also appreciate the comments from the anonymous reviewers. This work was partly funded by DARPA award F30602-00-2-0562, NSF CAREER award CCR-0093354, and a donation from Infineon Technologies.

9. REFERENCES

[1] D. Albonesi. Selective cache ways: On-demand cache resource allocation. In *MICRO-32*, November 1999.

[2] B. Amrutur and M. Horowitz. Techniques to reduce power in fast wide memories. In *ISLPED*, pages 92–93, October 1994.

[3] B. Amrutur and M. Horowitz. A replica technique for wordline and sense control in low-power SRAMs. *IEEE JSSC*, 33(8):1208–1219, August 1998.

[4] N. Bellas, I. Hajj, and C. Polychronopoulos. Using dynamic cache management techniques to reduce energy in a high-performance processor. In *ISLPED*, pages 64–69, August 1999.

[5] D. Burger and T. Austin. The SimpleScalar tool set, version 2.0. Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997.

[6] Cadence Corporation. `http://www.cadence.com/`

[7] K. Ghose and M. B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *ISLPED*, pages 70–75, August 1999.

[8] H. Hanson *et. al.* Static energy reduction techniques for microprocessor caches. In *ICCD*, May 2001.

[9] H. Zhou *et. al.* Adaptive mode control: A static-power-efficient cache design. In *PACT*, September 2001.

[10] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *ISLPED*, pages 273–275, August 1999.

[11] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *ISCA-28*, June 2001.

[12] R. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[13] J. Kin, M. Gupta, and W. Mangione-Smith. The Filter Cache: An energy efficient memory structure. In *Micro-30*, December 1997.

[14] L. Clark *et. al.* An embedded 32-b microprocessor core for low-power and high-performance applications. *JSSC*, 36(11):1599–1608, November 2001.

[15] M. Powell *et. al.* Gated-Vdd: a circuit technique to reduce leakage in cache memories. In *ISLPED*, July 2000.

[16] M. Powell *et. al.* Reducing leakage in a high-performance deep-submicron instruction cache. *TVSLI*, 9(1):77–89, February 2001.

[17] S. Heo *et. al.* Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *ISCA-29*, Anchorage, Alaska, May 2002.

[18] S. Narendra *et. al.* Scaling of stack effect and its application for leakage reduction. In *ISLPED*, pages 195–200, 2001.

[19] S. Santhanam *et al.* A low-cost, 300-MHz, RISC CPU with attached media processor. *IEEE JSSC*, 33(11):1829–1838, November 1998.

[20] S. Yang *et. al.* Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay. In *HPCA-8*, Feburary 2002.

[21] Taiwan Semiconductor Manufacturing Company. `http://www.tsmc.com/`

[22] L. Villa, M. Zhang, and K. Asanović. Dynamic zero compression for cache energy reduction. In *MICRO-33*, 2000.

[23] M. Zhang and K. Asanović. Highly-associative caches for low-power processors. In *Koolchips Workshop, MICRO-33*, December 2000.