

TABLE III
SPEEDUP AND GATE COUNTS OF APPROACH A, B, AND C FOR
 J -UNFOLDED ARCHITECTURE

	$J=8$	$J=16$	$J=24$	$J=32$
$deg(p(x))$	7	15	21	30
$deg(p'(x))$ (used v) in Appr. C	17(63)	49(127)	71(255)	93(255)
# of clock cycles/data block in [5] and Appr. A	962	482	322	242
# of clock cycles/data block in Appr. B	961	481	321	241
# of clock cycles/data block in Appr. C	963	484	324	244
T_{∞} of [5]($\#T_{XOR}$)	8	16	24	32
T_{∞} of Appr. A, B ($\#T_{XOR}$)	5.103	8.000	12.793	15.956
T_{∞} of Appr. C ($\#T_{XOR}$)	4.167	7.769	11.111	14.034
Speedup of Appr. A over [5]	33.33%	100.00%	84.62%	100%
Speedup of Appr. B over [5]	33.47%	100.42%	85.19%	100.83%
Speedup of Appr. C over [5]	59.83%	99.17%	98.77%	111.58%
# of XOR in Appr. A	57622	67338	69230	71549
# of XOR in Appr. B	4048	8172	11981	16344
# of XOR in Appr. C	2845	5469	9432	12512

As an example, all of the three proposed schemes are applied to an (8191, 7684) BCH code using generator polynomial $g(x) = x^{507} + x^{506} + x^{502} + \dots + x^6 + x^2 + 1$. Some results for different unfolding factors are summarized in Table III. Since the polynomial multiplications and the matrix multiplications can be pipelined, the associated latency can be excluded from the total number of clock cycles needed to encode each data block. Therefore, the number of clock cycles to process one data block is determined by the latency of the LFSR architecture, which can be computed as $\lceil \text{the length of the input to the LFSR} / J \rceil$. For example, in Approach A, the length of the input to the LFSR, which is $m(x)p(x)$, is $7684 + 15 = 7699$ for $J = 16$. Hence, the encoding of one data block requires $\lceil 7699/16 \rceil = 482$ clock cycles in this case. The critical loop of the architecture in [5] lies in the LFSR implementing the division by $p(x)$, and the iteration bound of this architecture is JT_{XOR} for the J -unfolded (8191, 7684) BCH encoder. Approach A, B and C can reduce the iteration bound by replacing the LFSR implementing the division by $p(x)$ with architectures free of feedback loops. As can be observed from Table III, the iteration bounds of these three approaches are much smaller than that of [5]. In addition, since the same $p(x)$ can be used in Approach A and B, the iteration bounds of these two approaches are the same. However, $p'(x)$ instead of $p(x)$ is used in Approach C. As a result, the iteration bound of Approach C is different. The speedup and gate count for each proposed architecture are also listed in Table III. As can be observed, Approach C has the smallest area requirement and can achieve similar or higher speedup than Approach A and B. Therefore, Approach C is the optimum approach for unfolding factors ranging from 8 to 32.

V. CONCLUSION

Three novel architectures have been proposed in this paper to reduce the achievable minimum clock period of parallel long BCH encoders derived through unfolding after the fanout bottleneck has been eliminated. In addition, the proposed schemes can also be used to speed up long cyclic redundancy checking and systematic long Reed–Solomon encoders. Among the three approaches, Approach C is optimum for moderate unfolding factors. However, the complexity for finding the smallest v , such that some extensions of $p(x)$ divides $x^v - 1$, is extremely high if exhaustive searching is used. Future work will be directed to developing systematic algorithms to compute the smallest v and the corresponding extensions of $p(x)$. In addition, reducing the iter-

A Case for Asymmetric-Cell Cache Memories

Andreas Moshovos, Babak Falsafi, Farid N. Najm, and Navid Azizi

Abstract—In this paper, we make the case for building high-performance asymmetric-cell caches (ACCs) that employ recently-proposed asymmetric SRAMs to reduce leakage proportionally to the number of resident zero bits. Because ACCs target memory value content (independent of cell activity and access patterns), they complement prior proposals for reducing cache leakage that target memory access characteristics. Through detailed simulation and leakage estimation using a commercial 0.13- μm CMOS process model, we show that: 1) on average 75% of resident data cache bits and 64% of resident instruction cache bits are zero; 2) while prior research carefully evaluated the fraction of accessed zero bytes, we show that a high fraction of accessed zero bytes is neither a necessary nor a sufficient condition for a high fraction of resident zero bits; 3) the zero-bit program behavior persists even when we restrict our attention to live data, thereby complementing prior leakage-saving techniques that target inactive cells; and 4) ACCs can reduce leakage on the average by $4.3\times$ compared to a conventional data cache without any performance loss, and by $9\times$ at the cost of a 5% increase in overall cache access latency.

Index Terms—Cache memories, computer architecture, high performance, leakage power reduction.

I. INTRODUCTION

In this work, we study methods that exploit the memory bit-value behavior of programs to drastically reduce leakage or static power dis-

Manuscript received June 14, 2004; revised February 7, 2005. This work was supported in part by the Semiconductor Research Corporation under Grant SRC 2001-HJ-901 and in part by an NSERC Discovery Grant. The work of N. Azizi was supported by an NSERC postgraduate scholarship (PGS A).

A. Moshovos, F. N. Najm, and N. Azizi are with the Electrical and Computer Engineering Department, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: moshovos@eecg.toronto.edu; najm@toronto.edu; nazizi@eecg.toronto.edu).

B. Falsafi is with the Electrical and Computer Engineering Department of the Carnegie-Mellon University, Pittsburgh, PA 15213 USA (e-mail: babak@cmu.edu).

Digital Object Identifier 10.1109/TVLSI.2005.850127

sipation in caches without sacrificing performance. Leakage power, or simply leakage, is already a significant fraction of overall power dissipation and is expected to grow by a factor of five every chip generation [3]. It is estimated that in a 0.1- μm technology, leakage power will account for about 50% of all on-chip power [9]. This increase in leakage is an unwanted side-effect of technological trends: successive processor generations have used more transistors clocked at higher frequencies to improve performance. Keeping the resulting increase in dynamic (or switching) power within limits is possible by scaling down the supply voltage. But, to maintain high operating frequency it is also necessary to scale the transistor threshold voltage (V_t), giving rise to sub-threshold leakage current and, hence, leakage power dissipation when the transistor is off [3]. Because leakage is proportional to the number of on-chip transistors and caches account for the dominant fraction of transistors in high-performance designs, we focus on leakage-aware cache design. There are a number of recent proposals for reducing leakage in caches. Circuit-level only techniques typically trade off performance for reduced leakage power where this is acceptable, e.g., L2 caches [7]. Proposals to reduce leakage in high-performance caches (e.g., L1 caches in high-performance processors) often use combined circuit- and architectural-level methods to reduce leakage. These techniques exploit the spatial (how much data) and temporal (for how long) characteristics of the memory reference stream of typical programs to reduce leakage in those parts of the cache that are left unused for long periods of time [6], [8], [10], [15], [16].

In this paper, we look beyond the time and space characteristics of program memory behavior. We offer a new degree of freedom in reducing leakage by exploiting the memory value content at the bit level. We evaluate asymmetric-cell caches (ACCs) that drastically reduce leakage even when most of the cache is actively used. At the core of ACCs is a set of asymmetric SRAMs that are designed to drastically reduce leakage when they store a zero (bit) while maintaining high performance. A circuit-level analysis of a family of asymmetric SRAMs was presented recently [1], [2]. While asymmetric SRAMs provide the means to reduce leakage in any generalized SRAM-based structure, e.g., cache memories, register files, translation look-aside buffers (TLBs), etc., it is the memory value behavior of programs that determines the actual potential for leakage reduction in cache memories using ACCs. In this paper we study the cache-resident memory value behavior of ordinary programs and show the leakage benefits possible in various applications. This work complements the previous studies that looked only at the circuit-level aspects of ACCs.

A number of studies have shown that the data processed and the instructions fetched by processors contain many zeros, e.g., [4], [5], [14]. These studies have focused on the dynamic memory stream (i.e., the values read or written by the processor) and at larger than a bit granularities (e.g., bytes). But all cache cells contribute to leakage equally independently of how often they are accessed by the processor. In this paper, we show that a high dynamic frequency of zero bytes is neither a necessary nor a sufficient condition for reducing leakage using ACCs, and the potential for ACCs is greater than that indicated by the fraction of zero bytes in the dynamic memory stream. The rest of this paper is organized as follows. In Section II, we comment on related work. In Section III, we discuss the rationale behind ACCs, explain how they relate to existing leakage reduction methods and present our analysis of memory value behavior. In Section IV, we review the asymmetric SRAM cell family that forms the core of the ACCs and present leakage reduction results. Finally, in Section V we summarize our findings.

II. RELATED WORK

We restrict our attention to those proposals that target leakage power reduction in high-performance caches. Many recent proposals to

reduce leakage in caches employ a supply-gating circuit-level mechanism called gated-V_{dd} [15] to effectively “turn off” cache cells that remain inactive for long execution periods. Yang *et al.*, proposed the dynamically resizable instruction (DRI) cache [12] which dynamically resizes and turns off portions of a cache. Kaxiras *et al.*, proposed cache decay [10] which turns off individual blocks using expiration counters. Zhou *et al.*, extend cache decay by dynamically extracting the expiration count value during runtime [16]. With all aforementioned methods, care must be taken to avoid disabling blocks that are live since doing so will increase miss rate and potentially dynamic power dissipation. Flautner *et al.*, propose drowsy caches [6] a variation on cache decay that obviates the need to turn off and evict cache blocks by putting potential dead blocks into “sleep” using supply-voltage-scaling. Heo *et al.*, propose floating the bitlines in unused subarrays of a sequentially-precharged cache to reduce bitline leakage [8]. As we show in Section III-D, our technique is orthogonal to all aforementioned methods. Even if it was possible to perfectly predict which blocks are not live, ACCs can reduce leakage significantly. Moreover, even when most cache blocks hold live data ACCs can reduce leakage power significantly. Villa *et al.*, present dynamic zero compression [13] where zero values are exploited to reduce dynamic power dissipation. ACCs attack leakage dissipation and rely on bit values. As we show in Section III-C, even if zero bytes were completely disabled, our method could still reduce leakage significantly.

III. EXPLOITING DATA VALUES TO REDUCE LEAKAGE

Ideally, cache cells would be as fast as possible consuming as little leakage power as possible. Unfortunately, this requirement is increasingly at odds with a fundamental technology trade off: as the supply voltage gets smaller, fast transistors tend to dissipate high leakage power. One of the key circuit-level mechanisms for reducing leakage is to use “weaker” transistors (e.g., having higher V_t or higher length-to-width ratio). Unfortunately, “weaker” transistors are also slower and the resulting performance loss is unacceptable for high-performance caches. ACCs exploit memory value content to reduce leakage in high-performance caches. ACCs are built using a family of novel SRAM cells (proposed in [1], [2]) built on the following premise: weaken (asymmetrically) only those transistors necessary to drastically reduce leakage when the cell stores a zero while maintaining high performance. Compared to conventional, high-performance SRAM cells, the asymmetric cells dissipate lower leakage in both states. However, the reduction in leakage is much higher when they store a “zero” as opposed to when they store a “one”. A detailed, circuit-level analysis of asymmetric SRAM appears in [1], [2] and is beyond the scope of this paper. What makes ACCs successful is the fraction of resident zero bits in typical program behavior. We analyze the bit-value program behavior and show that most (but not all) of the programs we studied exhibit a strong bias toward zero bits.

A. Methodology

We use SimpleScalar v3.0 to simulate the state-of-the-art superscalar processor with a two-level cache hierarchy detailed in Table I. We assume split level-one data (L1D), level-one instruction (L1I) caches, and a unified level-two (L2) cache. We present results for systems with an L1I and an L1D of 32 kbytes each (32-byte blocks, 2-way set-associative) and an L2 of 1 Mbytes (64-byte blocks, 4-way set-associative). We only present results for the data arrays of level-one caches. We have also experimented with characterizing bit values in L2 and found the L2 results quite similar to L1D results due to: 1) SPEC2000’s tiny instruction footprints allowing the data streams to dominate occupancy in L2 and 2) SimpleScalar v3.0 only simulating a uniprogrammed environment with a single application’s footprint in L2. We do not present a bit-value

TABLE I
PROCESSOR CONFIGURATION

Branch Predictor	Fetch Unit
16k GShare+16K bi-modal with 16K selector	Up to 8 instr. per cycle. 64-entry Fetch Buffer 10 cycles misprediction penalty.
Scheduler	Issue/Decode/Commit
128 insts and 64 loads/stores Perfect Disambiguation	Any 8 instructions / cycle 4 loads/stores per cycle
L1/UL2 Access Latencies	Main Memory
3/16 cycles	Infinite, 100 cycles

characterization for the cache tag arrays despite finding the values to be highly skewed. This is because SimpleScalar does not simulate address translation and hence tag values are artificially skewed toward zero. We used the following SPEC CPU2000 benchmarks: gzip, swim, applu, vpr, gcc, mesa, art, mcf, earthquake, ammp, parser, gap, vortex, bzip, and twolf. We also included mpeg2encode from mediabench. The binaries were compiled for the Alpha 21264 architecture using Compaq's compilers and for the Digital Unix V4.0F. We simulated two billion committed instructions or to completion (whichever happened first) after skipping the initialization. To account for variations in bit-value distributions across compilations and instruction sets, we also present results for binaries compiled for PISA. We used the gcc and g77 version 2.7.2 compilers for PISA. We consider only cache lines that have been touched at least once by the program.

B. Bit-Value Distribution

Fig. 1(a) illustrates the bit-value distribution in L1 caches including a breakdown in terms of the address space these bits belong to. On average, almost 75% and 64% of bit values are zero in the data and instruction caches, respectively. The instruction cache does not exhibit much variation in the distribution of zeros across applications. Furthermore, while zero is the common bit value, there is not a strong bias. Data caches exhibit a large variation in the skew toward zero. The breakdown of bits across the various segments indicate that except for three applications, gzip, applu, and swim, the majority of zero-bit distributions are from the heap.

The data cache results can be divided into three groups: 1) the applications with predominantly dynamic data structures and a significant skew toward a high fraction of zero bits; 2) the compression applications; and 3) the dense numerical (floating-point) applications. Vpr, gcc, mesa, art, mcf, earthquake, parser, gap, vortex, and twolf all heavily use dynamic memory allocation. Many of the heap objects are heavily biased toward zero and are actively used. Heap objects often contain small positive integer values and pointers. Also compilers often align aggregates of structures to prevent misaligned accesses, padding aggregates with fields that remain zero. The second group of applications, bzip, gzip, and mpeg2encode, compress the input streams and as such reduce the longevity of large sequences of zero bits in the data cache during execution. The third group of applications, amp, applu, and swim are dense numerical computations, in which bit values are more uniformly distributed across zeros and ones. In the extreme case, swim bit values are almost equally distributed across zeros and ones. While we do not show these results we note that using 64K L1 caches resulted in very similar results.

The next experiment was set up to identify how sensitive the cache bit-value behavior of programs is to the specific instruction set architecture. Fig. 1(b) shows the bit-value distributions for L1D caches for PISA binaries. We do not compare the bit-value distributions in the instruction cache because PISA uses a 64-bit instruction format that is highly skewed toward zero. The results indicate a slight increase in the

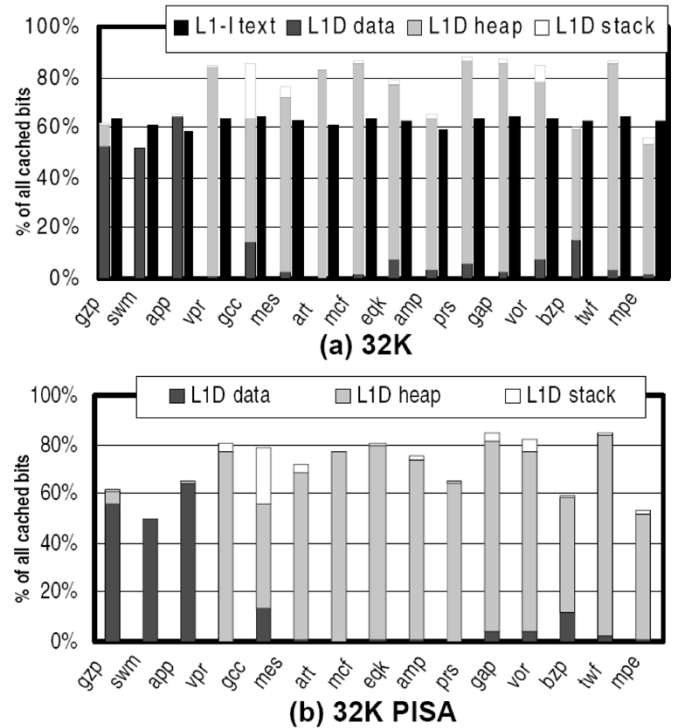


Fig. 1. (a) Fraction of cache bits that are zeroes for L1D and L1I for the ALPHA binaries for 32-Kbyte caches. (b) Same for PISA binaries and for 32-kbyte L1D caches. These PISA experiments do not include the benchmark gap, because the PISA capable gcc compiler cannot produce code for this benchmark.

skew toward zero when using the Alpha binaries. Unlike PISA, Alpha uses 64-bit integer and floating-point values.

C. Dynamic Versus Cache Resident Bit-Value Behavior

Prior work carefully evaluates the dynamic occurrence of zeros in the memory access stream [11], [14] and the datapath [4], [5]. Our key observation is that unlike prior work, ACCs target: 1) the behavior of cache resident (rather than accessed) values and 2) a high fraction of zero bits even if an application's cache-resident data exhibit a small fraction of zero bytes. We make the case that a high dynamic occurrence of zeros is neither necessary nor sufficient for ensuring a large fraction of zero bits in the cache. Fig. 2(a) and (b) show a breakdown of the cache data byte values in terms of the number of their bits that are "one" (range 0 to 8). Fig. 2(a) shows this breakdown for all cache resident data values while Fig. 2(b) does the same for the dynamic data reference stream. To understand the cache resident metric consider that in the trivial example of a cache with one line with the value all ones for one cycle and all zeroes for nine cycles, the cache-resident percentage of zero-bits is 90%. The dynamic frequency of zero bytes is not correlated to the fraction of zero bytes in the data cache. In some programs the fraction of zero bytes seen by the processor is much larger than that of the data stored in the cache. In other programs, the reverse is true. For example in twolf, 41% of the bytes accessed is zero, while 61% of the bytes in the cache are zero. The relatively frequency of static versus dynamic zero bytes is reversed for gzip where 40% of the bytes accessed versus only 20% of the cache bytes are zeros. There are applications such as swim, bzip, and mpeg2encode that have very few zero bytes in the data cache and that access very few zero bytes also. Still, even in these applications a large fraction of cache bits are zero suggesting that there is great potential for ACCs. For example, in swim more than half of the cache bits are zero while only 4% of the bytes are zero.

Fig. 2(c) and (d) illustrates the same byte-level distribution for instruction caches. The graphs indicate that zero distributions across

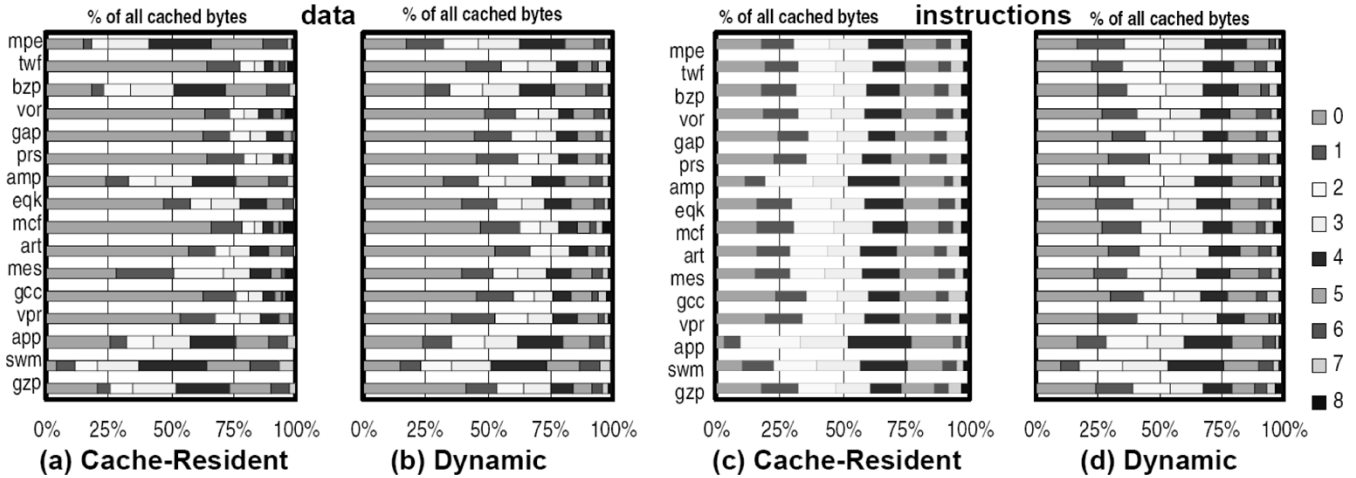


Fig. 2. Per-byte bit distribution. The figures show the breakdown of bytes with respect to the number of their bits that are one (range is 0 to 8). (a) Data cache resident bytes. (b) Data cache bytes as read by the processor. (c) Instruction cache resident bytes. (d) Instruction cache bytes as accessed by the processor.

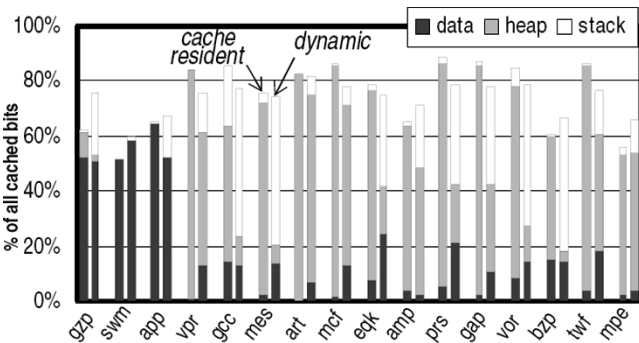


Fig. 3. Which memory segment zero bits come from for ALPHA binaries: left bar is for the cache resident values; right bar is for the dynamic memory reference stream.

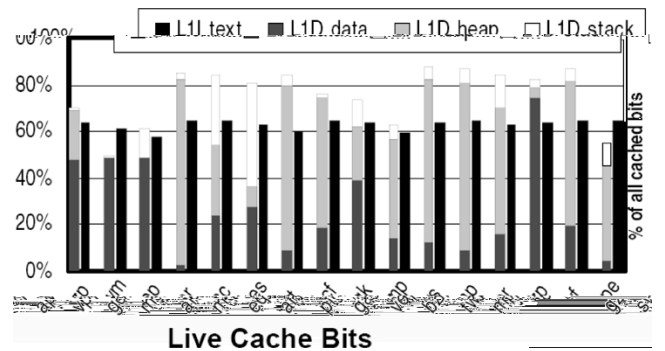


Fig. 4. Fraction of caches bits that are zeroes for L1D and L1I blocks that are “live” and for the ALPHA binaries.

bytes are more uniform in instruction caches than data caches limiting the opportunity for zero-byte compression. Fig. 3 shows a breakdown of the zero bits in the various memory segments for the data cache. Two bars are shown per program. The left bar is for the cache resident values while the right is for the dynamic memory reference stream. The relative importance of various data values is different across the two streams. For example, in virtually all programs, stack data have a negligible impact on the fraction of zero bits in the data cache. In the dynamic stream, however, stack values are responsible for many of the zero bits.

D. Bit Values in Live Data

Recent architectural/circuit cache leakage reduction techniques [10], [15] have used supply-gating [12] or supply-voltage-scaling [6] to reduce leakage in the inactive or “dead” cache lines. Asymmetric cells can be used in conjunction with these techniques to reduce leakage in both the “live” and “dead” cache lines. Fig. 4 illustrates the fraction of zero bits in L1D and L1I in the live cache lines—i.e., lines that will be accessed again prior to being evicted.

IV. ASYMMETRIC SRAM

The key enabling mechanism for value-based leakage optimization is a set of asymmetric SRAM cells. Here we review their principle of operation. A detailed, circuit-level analysis of the complete asymmetric SRAM family appears in [1] and [2]. Let us first review where leakage

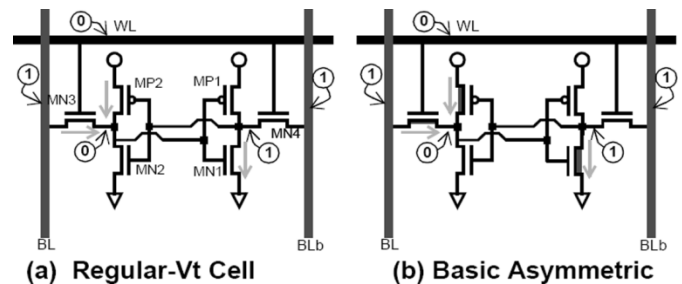


Fig. 5. (a) Conventional regular-Vt SRAM cell. (b) Asymmetric, dual-Vt cell.

is dissipated in the conventional high-performance six regular Vt transistor cell shown in Fig. 5(a). The cell comprises two inverters (MP2, MN2) and (MP1, MN1) and two pass transistors MN3 and MN4. In the inactive state, the wordline (WL) is held at “0” so that the two pass transistors are off isolating the cell from the two bitlines BL and BLb. At this stage the bitlines are also typically charged at Vdd (e.g., logical one). The cells spend most of their time in the inactive state. In this state, most of the leakage is dissipated by the transistors that are off and that have a voltage differential across their drain and source. Which are those transistors depends on the value stored in the cell. When the cell is storing a zero (left side) the leaking transistors are MP2, MN1, and MN3 [Fig. 5(a)]. Fig. 5(b) shows an asymmetric cell where the MP2, MN1 and MN3 have been replaced with high-Vt transistors. The leakage of this cell is comparable to the high-Vt cell in the preferred state and comparable to the regular-Vt cell otherwise. The basic

TABLE II
CHARACTERISTICS OF ASYMMETRIC CELLS

	Leakage (0)	Leakage (1)	Δ Delay	Δ SNM	$\Delta I_{trip}/I_{read}$
Regular-Vt	100%	100%	0%	0%	0%
LE	1%	14%	5%	7%	-5%
SE	14%	50%	0%	-6%	15%
SLE	14%	43%	5%	23%	-7%
SSE	50%	53%	0%	9%	13%

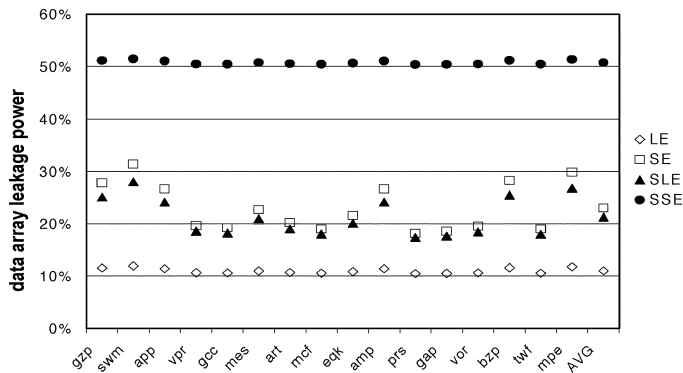


Fig. 6. Data cache leakage power with the asymmetric cells relative to the regular-Vt conventional cache. Lower is better.

asymmetric cell of Fig. 5(b) serves to illustrate the idea behind asymmetric cells but suffers in terms of stability and access latency (high-Vt transistor are marked with a gray line). A family of asymmetric cells with various performance, leakage and stability characteristics is presented in [2]. The characteristics of the better cells are summarized in Table II normalized over the Regular-Vt cell (first row). We consider four cells: 1) leakage-enhanced (LE); 2) speed-enhanced (SE); 3) stability-leakage enhanced (SLE); and 4) stability-speed enhanced (SSE). These cells offer different performance versus leakage reduction ratios.

We report leakage reduction in both states (20% means a reduction of $5\times$), increase in latency and two metrics of stability: signal-to-noise margin (SNM) and the current necessary to trip the cell's value. In the last two characteristics, a positive percentage suggests an increase in stability while a negative percentage a decrease. There is no clear best asymmetric cell if we consider all characteristics. Overall, the stability differences are minor and all cells are usable under worst-case assumptions and for the commercial 0.13- μm process that we used.

A. Leakage Power Reduction

Fig. 6 shows the leakage power dissipation for the four ACCs. Due to space limitations we report results only for the data cache. We report leakage power as a fraction of the leakage power dissipation of the regular-Vt cell cache. This metric includes only the power dissipated by the data array cells. We also measured leakage in the decoder and found that it represents a very small fraction of overall cache leakage power. Furthermore, the leakage dissipated by the decoder array is the same for the conventional cache and the ACCs. As expected, on the average, LE offers the highest average savings (11% or about $10\times$) but these come at the expense of a 5% increase in cache latency. SE and SLE perform similarly (22.9% and 21.2%, respectively). Finally, SSE that has the best stability characteristics offers only a $2\times$ reduction in leakage.

V. CONCLUSION

Contrary to existing architectural techniques for reducing cache leakage power that focus on the time and space characteristics of the memory reference stream we targeted its value content. The key enabling technology for our work is a set of asymmetric SRAM cells that dissipate drastically reduced leakage when they store a "zero." We demonstrated that the memory value behavior of programs is such that asymmetric cell caches can effectively reduce leakage power without impacting performance. One of the major insights of this work is that the behavior exploited by ACCs is not the same as the one exploited by recent zero value related work for reducing dynamic power in various parts of processors.

REFERENCES

- [1] N. Azizi, A. Moshovos, and F. Najm, "Low-leakage asymmetric-cell SRAM," in *Proc. 2002 Int. Symp. Low Power Electronics and Design*, Aug., pp. 48–51.
- [2] N. Azizi, F. Najm, and A. Moshovos, "Low-leakage asymmetric-cell SRAM," *IEEE Trans. Very Large Scale (VLSI) Integr. Syst.*, vol. 11, no. 4, pp. 701–715, Aug. 2003.
- [3] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, Jul. 1999.
- [4] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. 5th IEEE Symp. High-Performance Computer Architecture*, Jan. 1999, pp. 13–22.
- [5] R. Canal, A. Gonzalez, and J. E. Smith, "Very low power pipelines using significance compression," in *Proc. 33rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2000, pp. 181–190.
- [6] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *Proc. 29th Annu. Int. Symp. Computer Architecture*, May 2002, pp. 148–157.
- [7] F. Hamzaoglu, Y. Ye, A. Keshavarzi, K. Zhang, S. Narendra, S. Borkar, M. Stan, and V. De, "Dual-Vt SRAM cells with full-swing single-ended bit line sensing for high-performance on-chip cache in 0.13 μm technology generation," in *Proc. 2000 Int. Symp. Low Power Electronics and Design (ISLPED)*, Jul. 2000, pp. 15–19.
- [8] S. Heo, K. Barr, M. Hampton, and K. Asanovic, "Dynamic fine-grain leakage reduction using leakage-biased bitlines," in *Proc. 29th Annu. Int. Symp. Computer Architecture*, May 2002, pp. 137–147.
- [9] T. Kam, S. Rawat, D. Kirkpatrick, R. Roy, G. S. Spirakis, N. Sherwani, and C. Peterson, "EDA challenges facing future microprocessor design," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1498–1506, Dec. 2000.
- [10] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *Proc. 28th Annu. Int. Symp. Computer Architecture*, Jul. 2001, pp. 240–251.
- [11] M. Lipasti, "Value prediction and speculative execution," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, Apr. 1997.
- [12] M. D. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd: a circuit technique to reduce leakage in cache memories," in *Proc. 2000 Int. Symp. Low Power Electronics and Design (ISLPED)*, Jul. 2000, pp. 90–95.
- [13] G. Reinman and N. Jouppi, "An integrated cache timing and power model," Compaq Corp., Palo Alto, CA, Western Res. Lab. Tech. Rep. 2000/7, 1999.
- [14] L. Villa, M. Zhang, and K. Asanovic, "Dynamic zero compression for cache energy reduction," in *Proc. 33rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2000, pp. 214–220.
- [15] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar, "An integrated circuit/architecture approach to reducing leakage in deep-sub-micron high-performance i-caches," in *Proc. 7th IEEE Symp. High-Performance Computer Architecture*, Jan. 2001, pp. 147–157.
- [16] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte, "Adaptive mode control: a static-power-efficient cache design," in *Proc. 9th Int. Conf. Parallel Architectures and Compilation Techniques*, 2001, pp. 61–70.