TECHNICAL REPORT: TR-01-01-02
COMPUTER GROUP, ELECTRICAL AND COMPUTER ENGINEERING, UNIV. OF TORONTO

Asymmetric-Cell Caches: Exploiting Bit Value Biases to Reduce Leakage Power in Deep-Submicron, High-Performance Caches

Navid Azizi Andreas Moshovos Farid N. Najm Babak Falsafi‡

Electrical and Computer Engineering
University of Toronto
{nazizi, moshovos}@eecg.toronto.edu
najm@toronto.edu

‡*Electrical and Computer Engineering*
Carnegie Mellon Univeristy
babak@cmu.edu

www.eecg.toronto.edu/~aenao

Abstract

We propose a novel approach that leverages circuit- and architecture-level techniques to drastically reduce leakage power dissipation in high-performance caches even when most of the cache cells are actively used. We observe that the cache resident memory values of ordinary programs exhibit a strong bias towards zero or one at the bit level. We introduce a family of high-speed dual-Vt SRAM cell designs that exploit this bit-level bias to reduce leakage power while maintaining low access latency. The key characteristic of this cell family is asymmetry: Leakage power dissipation depends on the actual bit value stored. In the preferred state, the leakage power is smaller by as much as approximately 10x and comparable to that of an high-Vt cell. Asymmetry is also key to maintaining high performance reads (the main disadvantage of an high-Vt cell). We propose two asymmetric-cell cache (ACC) designs. The first is statically biased towards the zero bit value. The other uses run-time selective inversion to increase the number of zero-holding bits. We evaluate our designs using the SPEC2000 benchmarks and for a commercial 0.13µ, 1.2V CMOS technology. We find that for most programs the majority of memory bits are zero with the actual fraction varying from 52% to 88% for the level one data cache. We also find that this bias is less evident in the instruction stream (around 60% on the average). Using selective inversion, it is possible to further increase the fraction of zero-holding bits by another 6% and 11% for the level one data and instruction caches respectively. Overall, for one cell design, leakage power is reduced by 96% and 94% for the level one data and instruction caches compared to conventional caches. Finally, we demonstrate that our method is orthogonal to existing techniques that selectively disable cache blocks.

1 Introduction

Power dissipation has emerged as a first class design concern in all segments of the computer system market from handheld/portable computers to desktops and servers. Improvements in chip density accompanied by architectural innovations to exploit the abundance of transistors and boost processor performance have led to designs that are increasingly power demanding. Circuit designers have historically relied on scaling down the transistor supply voltage every generation to reduce voltage swing and the *switching power* dissipated in capacitive loads when transistors switch. Unfortunately, to maintain an increase in transistor switching speeds, supply voltage scaling has been accompanied by transistor threshold voltage scaling, giving rise to a *subthreshold leakage* current when the transistor is off [5,14,20]. The resulting *leakage* (or static) power dissipation is a significant fraction of overall chip power dissipation in deep-submicron (< 0.18µ) processes, and expected to grow by a factor of five every chip generation [5]. It is estimated that in a 0.10µ technology leakage power will account for about 50% of all power dissipated on chip [13].

Unlike switching power, which is primarily a function of transistor activity, leakage power is proportional to the total number of transistors on chip. As such, a number of recent proposals for reducing leakage power have focused on-chip SRAM-based memory structures — such as multiple levels of instruction and data caches and TLBs, register file, etc. — accounting for a dominant fraction of high-end chip's transistors. Unfortunately, many of these techniques are circuit- and technology-centric and fundamentally trade off performance for lower leakage in less performance critical circuits by using higher-threshold voltage transistors. Other techniques combine both circuit (e.g., supply gating) and architectural methods to exponentially reduce leakage in those parts of the cache that remain unused for long periods of time [14,20,21]. These techniques only benefit the unused cache parts and have limited applicability for applications or application phases that place a large demand on cache storage. Moreover, the mechanisms to identify

the unused cache parts and trigger leakage saving incur high enough switching power and delay overheads to limit the granularity of leakage saving to every thousands of cycles.

This paper makes the observation that the distribution of *bit values* (zeros or ones) in SRAM cells is highly skewed towards a higher number of *zeros* both in the data and the instruction stream. Data bitstreams are more likely to have zeros than ones for a variety of reasons. For example, zero and relatively small positive integers (such as loop iteration counts) are common in computation and are often stored as words. Also heap objects are typically fragmented and are always zeroed at allocation time leading to large fraction of zeros in cache blocks, and linked data structures in many systems (e.g., various flavors of Unix) often have zeros in the upper pointer address bits because heap grows downwards. Similarly, the distribution of values in the instruction streams are skewed towards a higher number of zero bits. For example, immediate values, address displacement offsets, and branch displacements are often small positive integers [11]. Also, modern instruction sets use a sparse encoding of opcodes to accelerate decode time.

In this paper, we present a family of cache designs called the *Asymmetric-Cell Caches* (ACCs) that drastically reduce leakage power even when *most* of the cache is *actively used*. ACCs exploit the skewed distribution of bit values to reduce leakage while minimizing the impact on performance. Conventional SRAM designs use a "symmetric" configuration of transistors with identical threshold voltage, speed, and leakage characteristics and are designed for either fast access time or low leakage but not both [10]. ACCs use asymmetric SRAM cell designs in which a selected set of transistors are "weakened" — i.e., have higher threshold voltages — to reduce leakage (in the common case) when the cell contains a zero, with none or a minimal increase in cell access time.

We propose two ACC designs: The first is statically biased towards the zero bit value. The second, uses *dynamic selective inversion* to increase the fraction of cache bits that hold a zero. The contributions of this paper are: (1) We present a family of asymmetric SRAM cells that drastically reduce leakage power compared to conventional SRAM cells. The two best designs offer different performance/leakage characteristics. (2) We present two asymmetric cache cell designs that are statically biased towards zero or that dynamically invert cache data so that at least 50% of the stored bits are zeros. (3) We study the bitstream value characteristics of most the SPEC2000. (3) We evaluate the various ACC and asymmetric cell designs for a commercial 0.13µ, 1.2v CMOS technology.

We find that the asymmetric cell designs reduce leakage power by at least 2x and by as much as about 70x compared to a conventional SRAM cell. By comparison, the all high-Vt cell reduces leakage by about 70x. The majority of cache resident bits are zeros for the programs we studied. Moreover, by using selective inversion we can increase the zero bits by 11% on the average. Compared to a conventional cache, a statically biased level one data ACC dissipates only 4.2% leakage power. This comes at the expense of a 15% increase in cell read latency (only a fraction of overall cache access latency). By comparison, the all high-Vt conventional cache dissipates 1.4% of the leakage power of the conventional all low Vt cache. However, it is 60% slower. An alternative ACC, dissipates 22% of the leakage power with no impact on read latency.

The rest of this paper is organized as follows: In section 2, we present our asymmetric cell family. In section 3, we discuss two cache designs that use asymmetric cells. We present our experimental results in section 4 that includes an analysis of the bit values of the SPEC2000 programs, an analysis of the leakage power, performance and stability of the various asymmetric cells. Finally, we report results on the leakage power dissipation of the data portion of the level one data and instruction caches. In section 5, we review related work. We summarize our findings in section 6 and offer concluding remarks.

2 Asymmetric-Cell SRAM

Ideally, an SRAM cell would be fast and would dissipate low leakage power. This is increasingly at odds with a fundamental technology trade off between transistor speed and leakage power: the lower the threshold voltage of a transistor the faster it becomes and the more leakage power it dissipates. Conventional high-performance SRAM-cells use six transistors of identical threshold voltage to achieve fast access latency. Until very recently, the leakage power of such cells was not a concern, hence building high-performance caches with relatively low leakage power dissipation was possible. As we move towards deep-submicron, subvolt processes maintaining high performance requires scaling transistor threshold voltage too. As a result, leakage power increases rapidly. One possibility for confining leakage power is to use high-Vt transistor cells. Unfortunately, high-Vt cells are unsuitable for high-performance caches (i.e., level one data or instruction caches) since they are much slower than the regular Vt cell.

In this work, we propose a family of asymmetric SRAM cells that use a mix of regular- and high-Vt transistors.

Common to all cells is *asymmetric* leakage power behavior: When the cell stores the preferred value, leakage power is significantly lower (by about 3x or 10x depending on the design). This cell family drastically reduces leakage power since ordinary programs exhibit a strong bias in their cache-resident bit values. Compared to the conventional six-transistor cell, our cells also exhibit asymmetric access behavior. We can exploit this behavior to maintain fast access latency that is identical or comparable to that of the regular-Vt[1] cell.

In this section, we start by reviewing the conventional regular-Vt transistor cell and focus on where leakage power is dissipated. We then explain how we can reduce leakage power with no or little impact on access latency by selectively "weakening" some of the transistors (i.e., replacing some transistors with high-Vt ones). Initially, we ignore stability issues and present a simple to understand asymmetric cell design. Once the basic premise is understood, we discuss stability and present two asymmetric cells that offer comparable or better stability than the regular-Vt cell with little or no access latency overhead.

2.1 Conventional SRAM Cell

Figure 1 shows a conventional six transistor SRAM cell. The cell comprises two inverters (MP2, MN2) and (MP1,MN1) and two pass transistors MN3 and MN4. At 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).

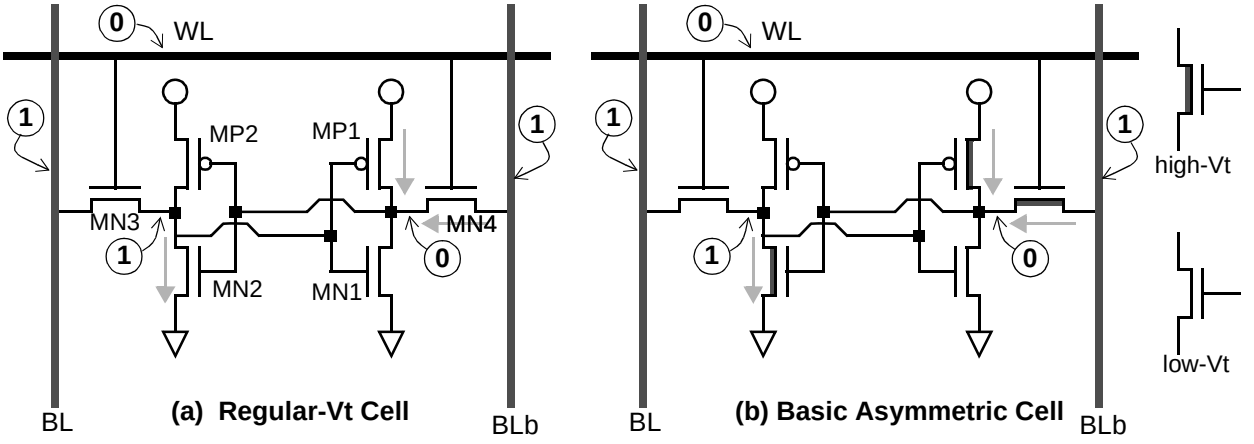

Figure 1: *(a) Conventional Regular-Vt SRAM cell. (b) An asymmetric, dual-Vt cell. Arrows indicate transistors with high leakage power dissipation. Circled numbers indicate logical voltage levels.*

To read the cell's contents, the wordline is set to one (Vdd). This activates the two pass transistors MN3 and MN4 and the side holding a zero starts discharging the corresponding bit line. A sense amplifier amplifies the voltage differential between BL and BLb. Once sufficient voltage differential exists (e.g., 200mv for a 1.2V supply) the access is complete. The read access latency heavily depends on the cell's ability to quickly discharge the corresponding bit line. This discharge happens via one of the pull down chains (MN3, MN2) or (MN4,MN1). Figure 1(a) shows the cell storing a one on its left side. Consequently, on a read, it is the right side pull-down chain that will be discharging BLb. To change the cell's contents we first set one of the bitlines to one and the other to zero. The exact values depend on what is the value we want to write in the cell. Then we activate the wordline and the cell is forced to the desired value.

2.2 An Asymmetric SRAM-Cell

Our asymmetric cell family is best understood by first explaining where leakage power is dissipated in the conventional regular-Vt cell. The vast majority of cells spend most of their time in the inactive state where the wordline is pulled low and the two bitlines are charged at Vdd. In this state, most of the leakage power 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. In figure 1(a) we show the cell storing a one on the left side. In this state, the leaking transistors are MP1, MN4 and MN2. If the cell was storing a zero on the left side, then the leaking transistors would

1. Note that typical, commercial CMOS processes offer also ultra-low Vt transistors. However, these are not suitable for SRAMs due to stability considerations.

be MP2, MN3 and MN1. One way of reducing leakage power would be to use an *high-Vt* cell where all transistors are replaced with high-Vt ones. Unfortunately, as we show in section 4.6, this degrades bitline access latency by as much as 60%. High-Vt transistors are not as strong and hence require a lot more time to discharge the relatively large capacitance of the bitlines.

As we show in section 4.2, ordinary programs exhibit a strong bias in the cache-resident bit values. Optimizing for the common case, our asymmetric cell family is built on the following premise: *Select a preferred stored value and weaken only those transistors necessary to drastically reduce leakage power when this value is stored.* Figure 1(b) shows an asymmetric cell where the MP1, MN4 and MN2 have been replaced with high-Vt transistors.The preferred state is the one where the output of the left side is one. As we show in section 4.6, the leakage power of this cell is comparable to the high-Vt cell in the preferred state and comparable to the regular-Vt cell otherwise.

Having explained how the asymmetric cell reduces leakage power, we have to also consider access latency and stability.

2.2.1 Access Latency

In the asymmetric cell of figure 1(a) the pull-down chains that are critical to performance are made of both high- and regular-Vt transistors. On the left side we have a regular-Vt pass transistor chained with a high-Vt pull-down and the reverse on the right side. Consequently, this cell is not as fast as the regular-Vt cell. Still, it is much faster than the high-Vt cell. Moreover, because the two pull-down chains are different, the cell's access latency depends on its value. In particular, the left side pull-down chain is faster. It is possible to achieve fast, value independent access latency by exploiting the fast pull-down chain. We describe two alternative read configurations that do so: *single-ended* and *pseudo-differential self-timed*.

Single-Ended Reads: Rather than using differential signaling, we could use single-ended reads [10]. In this case, the sense amplifiers at each column only observe one of the bitlines. To ensure noise immune operation, single-ended reads typically require that the bitline is discharged fully. For this reason, single-ended reads are typically slower than differential reads. However, it is expected that the relative difference between differential and singled-ended reads will be small in future processes. This is because the number of rows per cache sub-array has been decreasing steadily.

Pseudo-Differential Self-Timed Reads: Even in processes where single-ended sensing is not fast enough, it is possible to leverage the fast side of the asymmetric cell to achieve fast overall access latency. In this design, we still look at the voltage differential across BL and BLb. There are two possibilities. In the first case, the fast left pull down chain quickly discharges BL and access latency is comparable to that of a regular-Vt cell. In the second case, the slower pull down chain starts discharging BLb. Ordinarily, if we had to wait until BLb is sufficiently discharged (e.g., 200mV for a 1.2V supply), access latency will be much slower. The key idea is to *time* the access so that either BL is sufficiently discharged, or there is little voltage differential between BL and BLb (with BLb being slightly discharged). In the second case, we effectively make the determination that should we have waited longer, BLb would have been discharged. To maintain correct operation we cannot rely on the BL and BLb lines to determine how long we should wait. Instead, we have to establish a reliable *time reference.* For this purpose we use a dummy cell column. Using dummy cells as time reference generators was recently proposed to reduce dynamic power in SRAMs [2]. The dummy cell column contains a single cell per row and holds the value that discharges BL. We access the regular cells and the dummy cell in parallel. As soon as the dummy cell discharges its BL we can "sample" the output of the BL and BLb lines of the other cells and check whether BL is approx. 200mv lower than BLb. Compared to differential sensing, this method produces almost identical sensing time (there is some overhead associated with using the dummy cell for timing). Since we still use both BL and BLb this sensing method maintains most of the noise immunity properties of conventional differential sensing: noise should affect BL and BLb similarly. Moreover, since BLb does not discharge as much as BL, this method may also improve dynamic power dissipation. Dummy cells are commonly used in DRAMs, however, there they are used for voltage reference generation.

2.2.2 Stability

A major consideration with cell design is its stability over all possible variations allowed by the semiconductor process and operating conditions. There are two interrelated issues: read stability and noise margins [10,18]. Intuitively, read stability indicates how likely it is to invert the cell's stored value when accessing it. Noise margins suggest how immune is the cell to noise appearing on the bitlines, wordline and supply lines. We have performed stability analysis on all the cells we report in this paper. We found that they offer comparable or better stability than that of the conven-

tional regular-Vt cell. The results are summarized in section 4.6.

2.3 Two Improved Asymmetric SRAM Cells

It is possible to further optimize the asymmetric cell of figure 1(b) improving stability and access latency. We have investigated a total of 9 meaningful variations of asymmetric cell designs. In the interest of space, we present the two best designs. These are shown in figure 2. The *leakage enhanced* (LE) cell in part (a) offers better leakage behavior than that of figure 1(b). The LE-cell exhibits strong asymmetric power behavior, yet it dissipates reduced power compared to the regular-Vt cell in both possible states. The *speed enhanced* (SE) cell in part (b) dissipates higher leakage power compared the LE-cell. This is because the left-side pull-down chain is made of regular-Vt transistors. In either state, it still dissipates lower leakage power than the regular-Vt cell time. The main advantage of this cell is that its access latency is virtually identical to that of the regular-Vt cell. Also, notice that the pull-down chains employ either all regular-Vt or all high-Vt transistors. As we show in section 4.6, this improves cell stability.

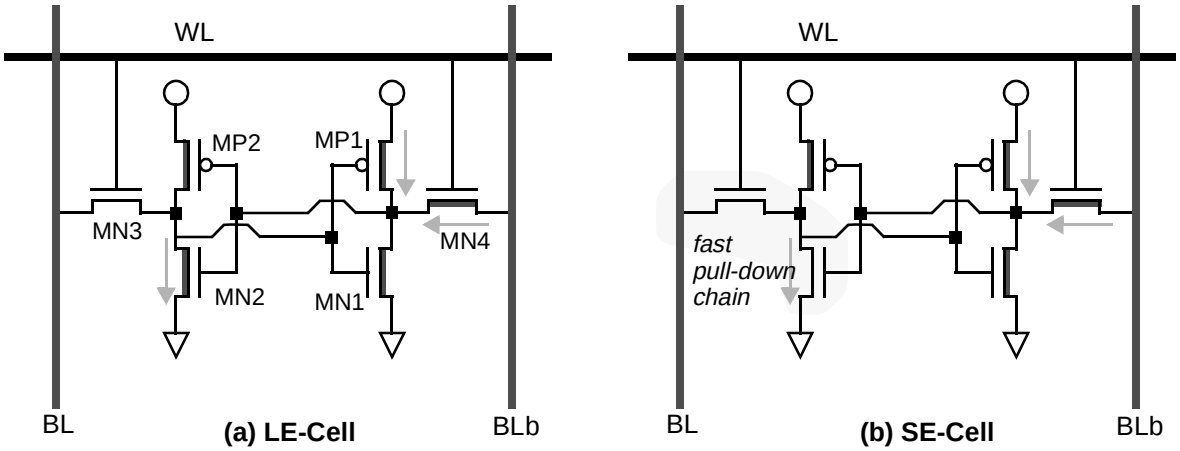

Figure 2: *Two improved asymmetric cells. Compared to cell in figure 1(b): (a) Leakage Enhanced Cell (LE): Reduced leakage power dissipation. (b) Speed Enhanced Cell (SE): Higher leakage dissipation but access latency virtually identical to the regular-Vt cell.*

3 Asymmetric-Cell Caches

We investigated two cache organizations that use asymmetric cell designs: *statically biased* and *dynamic inversion*. In the *statically biased* cache the cells of a conventional cache are replaced with asymmetric ones. This cache is *statically biased* to dissipate low leakage power only when it stores the preferred bit value. What makes this cache successful is *typical* program behavior: as we show in section 4.2, the SPEC2000 programs we studied exhibit a strong bias towards zero. In principle, it is possible that a program may exhibit the reverse behavior reducing the power benefits of this design. However, it still dissipates much lower leakage power compared to the regular-Vt cell cache.

In *selective inversion*, the values stored within a block can be inverted. For example, in one design inversion is possible at the byte level. In this case, if a byte contains five or more ones it is inverted prior to storing it in the cache. For correct operation it is necessary to also hold information on which bytes were inverted. To do so, we introduce an *inversion flag* per byte. This is just an extra cell. Inversion happens at write time. Since stores are typically buffered in a writebuffer and are only sent to the data cache on commit, there is plenty of time to decide and apply inversion if necessary.

Selective inversion allows us to further increase the fraction of cells that are in the preferred low-leakage power state. Another advantage of this method is that it reduces the maximum leakage power as it guarantees that at most 50% of the cells will be in the high-leakage power state. However, there are area and potential performance overheads associated with this technique. Inversion flags are required per block of cells that may be inverted. These inversion flags add to the wordline length and hence may impact read and write access latencies. Moreover, upon reading a cache value we may have to invert it. This inversion will increase overall access latency.

Thus far, we have assumed that the granularity of inversion is a byte. Other possibilities exist. For example, we

could invert every word (32 bits) or a whole block. This reduces the number of inversion flags and hence improves access time. However, if we use anything else than a byte, partial writes will have to be converted to a read-modify-write access to maintain appropriate inversion. Moreover, using a larger than a byte granularity increases the complexity of the circuit that determines whether a value should be inverted or not. In this work, we restrict our attention to byte-level inversion.

4 Evaluation

The leakage power reduction possible with asymmetric cells increases with the fraction of cache bits that are zero. In section 4.2, we study the values of the cache bits for a set of ordinary programs. We consider, different instruction set architectures and cache sizes. Moreover, we also present a breakdown of the zero bits in terms of the part of the address space they belong to. In section 4.3, we consider only those blocks that hold live data. We do so, to demonstrate that our method can reduce leakage power significantly even if it was possible to perfectly predict and disable those cache blocks that do not hold live data. In section 4.4, we look at individual bytes rather than at the whole cache. This analysis provides additional insight on what causes the bias towards zero bits. Moreover, it demonstrates that while zero bytes are quite frequent, often most of the zero bits belong to non-zero values. In section 4.5, we consider selective inversion. We present a circuit-level analysis of the performance, leakage power and stability of the various asymmetric cells in section 4.6. Finally, in section 4.7 we report the overall leakage power reduction possible with various caches that use asymmetric cells.

4.1 Methodology

To evaluate the bit value distribution in the caches, we use Simplescalar v3.0 to simulate an aggressive wide-issue dynamically-scheduled superscalar processor. Table 1 depicts the processor configuration parameters we assume in this study. In all of our experiments, 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 either 32 or 64 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 on bit value characterization in 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: (a) SPEC2000's tiny instruction footprints allowing the data streams to dominate occupancy in L2, and (b) SimpleScalar v3.0 only simulating a uniprogrammed environment with a single application's footprint in L2. As such, in the interests of space, we only present characterization of bit values in level-one caches. Similarly, we do not present bit value characterization in the cache tag arrays despite finding the values to be highly skewed, because SimpleScalar v3.0 does not simulate address translation of a realistic multiprogrammed environment.

Base Processor Configuration			
Branch Predictor	16k GShare+16K bi-modal with 16K selector	**Fetch Unit**	Up to 8 instr. per cycle. 64-entry Fetch Buffer 10 cycles branch misprediction penalty.
Instruction Window Size	128 entries RUU-like	**Load/Store Queue**	64 entries, 4 loads or stores per cycle Perfect disambiguation
Issue/Decode/Commit Bandwidth	any 8 instructions / cycle	**Functional Unit Latencies**	same as MIPS R10000
L1/UL2 Access Latencies	3/16 cycles	**Main Memory**	Infinite, 100 cycles

Table 1: Base configuration details. We model an aggressive 8-way, dynamically-scheduled superscalar processor having a 128-entry scheduler and a 64-entry load/store queue.

Table 2 depicts the SPEC2000 programs we study. The binaries were compiled for the Alpha 21264 architecture using Compaq's compilers and for the Digital Unix V4.0F. For every application, the table also depicts the memory reference counts and the L1D and L1I miss ratios for the cache configurations we study. To account for variations in bit value distributions across compilations and instruction sets, we also present results for SPEC2000 binaries compiled for PISA, the MIPS-like instruction set simulated by Simplescalar v3.0.

To measure leakage power we used SPICE and the production-strength model provided by a vendor of a commercial 0.13µ, 1.2v CMOS technology. We combine results from SPICE and the timing simulations to obtain a measurement of cell leakage power dissipation and read time. We also present results from CACTI II [17] to gauge the impact

App.	Ab.	Sim. Insts.	References		Skip	Miss Rates					
			Total	Comm.		32k L1			64k L1		
						Instr.	Data	L2	Instr.	Data	L2
164.gzip	*gzp*	2B	968M	634M	1B	< 0.1%	2.6%	2.2%	< 0.1%	2.3%	2.7%
171.swim	*swm*	2B	659M	658M	1B	< 0.1%	17.6%	30.8%	< 0.1%	15.1%	34.4%
173.applu	*apu*	2B	763M	758M	1B	< 0.1%	11.4%	27.7%	< 0.1%	11.4%	11.3%
175.vpr	*vpr*	2B	1,008M	664M	4B	< 0.1%	5.4%	10.3%	< 0.1%	4.6%	12.1%
176.gcc	*gcc*	2B	1,106M	665M	0	0.6%	1.4%	2.0%	0.2%	0.8%	4.4%
177.mesa	*mes*	2B	763M	622M	1B	< 0.1%	0.1%	27.5%	< 0.1%	0.1%	0.28%
179.art	*art*	1.5B	679M	593M	0	< 0.1%	34.6%	27.6%	< 0.1%	34.6%	27.5%
181.mcf	*mcf*	2B	1,603M	736M	2B	< 0.1%	26.0%	35.2%	< 0.1%	24.6%	37.3%
183.equake	*eqk*	1.4B	550M	657M	0	< 0.1%	4.4%	26.9%	< 0.1%	3.8%	31.8%
188.ammp	*amp*	2B	866M	784M	2B	< 0.1%	6.6%	11.2%	< 0.1%	5.7%	12.7%
197.parser	*prs*	2B	1,118M	733M	3B	< 0.1%	3.4%	5.1%	< 0.1%	2.7%	5.9%
254.gap	*gap*	2B	1,209M	789M	4B	< 0.1%	2.0%	24.2%	< 0.1%	1.8%	27.3%
255.vortex	*vor*	2B	930M	862M	4B	0.6%	.8%	4.5%	0.2%	0.5%	9.9%
256.bzip	*bzp*	2B	1,164M	1,048M	3B	< 0.1%	1.1%	33.4%	< 0.1%	1.0%	34.7%
300.twolf	*twf*	2B	1,017M	632M	5B	< 0.1%	9.1%	< 0.1%	< 0.1%	7.9%	< 0.1%

Table 2: *Applications used in our studies. In the interest of space, we will refer to the applications using the three letter abbreviations listed under the "Ab." column. Reported from left to right are the resulting instruction counts, the number of executed and committed memory references, the number of instructions we skipped prior to measuring program behavior and the cache accesses and resulting miss rates.*

on overall cache access time given an increase in cell read time.

4.2 Bit Value Distribution of Ordinary Programs

Figure 3(a) illustrates the bit value distribution in 32-Kbyte L1 caches. The graphs indicate that there is much opportunity to exploit in the distribution of bits in the caches. On average, almost 80% of bit values are zero in the applications we studied. Similarly, there is a considerable skew in the bit value distribution in the instruction cache. The instruction caches results are, however, less dramatic and the fraction of zero bits in L1I is on average slightly over 60%. These results corroborate findings from recent research advocating zero-byte compression in caches especially due to the data stream [19], and zero-compression of operands in the pipeline datapath [8, 7], and value prediction [15].

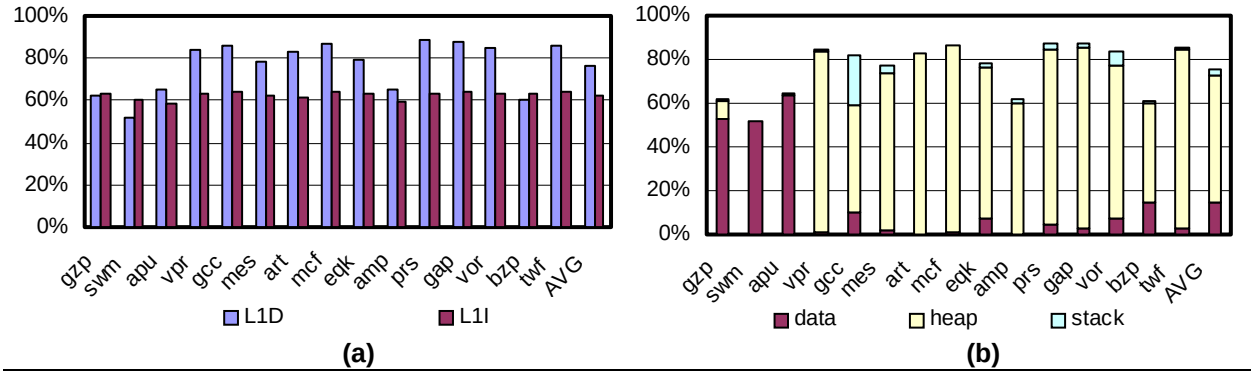

Figure 3: *(a) Fraction of cache bits that are zeroes for L1D and L1I for the ALPHA binaries for 32-Kbyte caches. (b) Memory space breakdown of zero bits for L1D.*

Not surprisingly, the instruction cache does not exhibit much variation in distribution of zeros across applications. We expect common opcodes, zero operand specifiers (i.e., immediate values, branch displacements, memory address

offsets, and register zero) are more or less uniformly distributed across applications. RISC instruction sets also tend to have sparse encoding of opcodes to facilitate and accelerate decode. In Alpha, the most common instructions — such as the integer ALU instructions — have at most two out of 14 opcode and function bits set to one.

Unlike instruction caches, data caches exhibit a large variation in the skew towards zero. Figure 3 (b) illustrates the breakdown of zero distribution across memory addresses into data, heap, and stack segments. The results indicate that except for three applications, *gzip*, *applu*, and *swim*, the majority of zero-bit distributions are from the heap. *Gzip* maintains its main data structures (i.e., the input/output compression buffers), statically in the data segment. *Applu* and *swim* are FORTRAN77 applications and therefore do not use heap. The rest of the applications allocate their data structure dynamically in the heap.

The data cache results can be divided into three groups: (1) the group of applications with predominantly dynamic data structures and a significant skew towards a high fraction of zero bits, (2) the compression applications, and (3) the dense numerical (floating-point) applications. *Vpr*, *gcc*, *mesa*, *art*, *mcf*, *equake*, *parser*, *gap*, *vortex*, and *twolf* all heavily use dynamic memory allocation. In these applications, we expect the key contribution to the high skew towards zero bits to be due to small positive values occurring frequently in computation. For instance, *art* and *mesa* compute over thermal and graphical images respectively, and benefit from a large distribution of zero image data bytes. However, there are secondary factors that may favor the skew towards zero. Dynamic memory allocation libraries typically allocate heap objects with a spectrum of pre-defined sizes (e.g., power-of-two size objects), irrespective of the data structure requirements in applications. As such, they are likely to exhibit fragmentation in heap objects, resulting in a high skew towards zero bits when these objects are loaded in the data cache. Moreover, compilers often align aggregates of structures to prevent misaligned accesses, padding aggregates with fields that remain zero. On a close inspection of the source code, we found that some of these applications (e.g., *art* and *mesa*) request dynamic allocation for a small number (~16) of bytes, inadvertently allocating a large number of small heap objects with a high probability of fragmentation rather than allocating one large object and dividing it up among a large number of small data structures.

The second group of applications, *bzip* and *gzip*, compress the input streams and as such reduce the longevity of large sequences of zero bits in the data cache during execution. As such, these applications exhibit less skew towards zero. 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.

Figure 4 (a) depicts the bit value distributions for a 64-Kbyte cache. The results are almost identical to a 32-Kbyte cache primarily because the application footprints do not change as corroborated by the small change in miss ratios (table 2) when increasing the sizes from 32 to 64 Kbytes. Figure 4 (b) compares the bit value distributions for L1D caches across Alpha and 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 towards zero. The results indicate a slight increase in the skew towards zero when using the Alpha binaries. Unlike PISA, Alpha uses 64-bit integer and floating-point values. There are many factors that may affect the skew across the binaries including but not limited to the compiler's use of 64-bit register values in Alpha (as compared to 32-bit values in PISA) which would increase the skew of zero bits for small positive integers, and the allocation, alignment, and padding of dynamic data structures in memory.

4.3 Bit Values in Live Blocks

Recent architectural/circuit techniques to reduce cache leakage [14,20] have targeted identifying inactive cache block frames and using supply-gating [16] to reduce leakage in the inactive or "dead" frames. asymmetric cells can be used in conjunction with these techniques to reduce leakage in both the "live" and "dead" cache block frames. In this section, we evaluate bit value distribution in the "live" cache blocks.

Figure 5 (a) illustrates the fraction of zero bits in L1D and L1I in the live cache block frames — i.e., block frames to which a subsequent processor reference will result in a hit. The graphs indicate that, as expected, there is little change in distribution of zeros in the instruction stream. There are small variations towards a higher or lower skew depending on the application, affecting the average fraction of zero bits by 3%. Other applications either exhibit no change, or a slight reduction in the skew towards zero.

Figure 5 (b) illustrates the distribution of zero bits across the segments. The breakdown indicates a significant shift

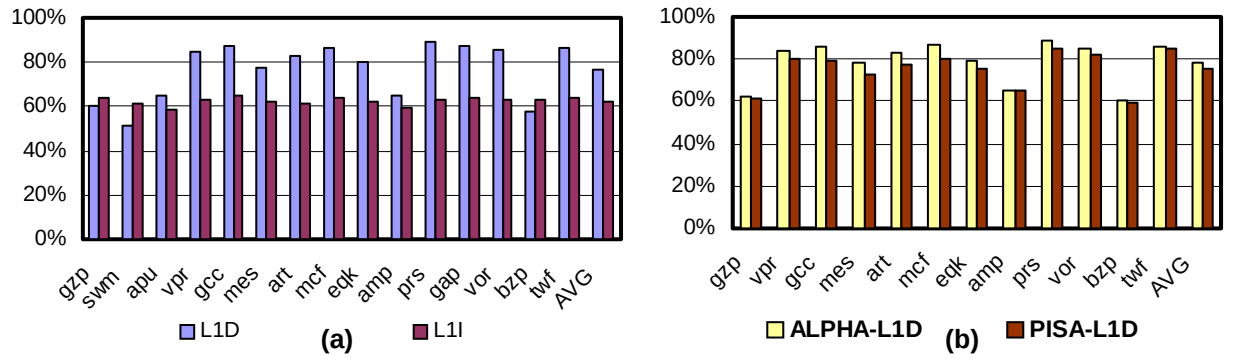

Figure 4: *(a) Fraction of cache bits that are zeroes for L1D and L1I for the ALPHA binaries for 64-Kbyte caches. (b) Comparing the bit value distribution across the ALPHA and PISA binaries for 32-Kbyte L1D caches. These numbers do not include the benchmarks applu, swim and gap, because PISA binaries for these benchmarks were not available at the time of this experiment.*

across the segments that contribute to the zero bit skew. In general the breakdown graphs indicate a higher shift towards the data segment and the stack due to two reasons: (1) the variables and data structures in the data segment are often used for lookup purposes (e.g., pointers to global structures that are actively referenced, constant scalars, tables holding constant values initialized statically the compiler) remaining active and live, (2) stack frames are always used for keeping temporary computation and as such always remain active, and (3) the main data structures, which in most applications are allocated in the heap, are often swept once per application phase to compute new values and otherwise remain inactive or dead in the cache (e.g., updating of image pixels, or compressing data from an input buffer). Moreover, a reduction in live heap data may also result in a slight reduction in the skew towards zero due to the decreased contribution of zeros from fragmented heap objects. *Gzip* makes heavy use of compile-time allocated data structures where it stores the (uncompressed) input stream likely to have a large set of zero bits, while using the heap for scratch storage. *Bzip*, however, primarily uses statically-allocated arrays/buffers for scratch storage and as such exhibits a significant shift towards the data segment.

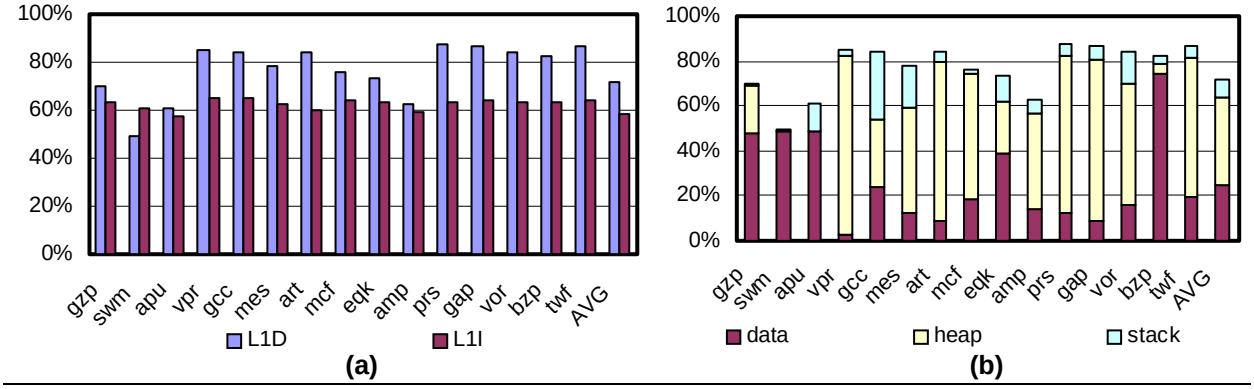

Figure 5: *(a) Zero bit fraction with 32K L1 caches and for only those blocks that hold live data.(b) Memory space breakdown for live data.*

4.4 Per-Byte Bit Distribution

Recent research advocates zero-byte compression to reduce switching energy in caches [19]. While these techniques have been primarily proposed to reduce bitline switching, they can be coupled with supply-gating to reduce leakage in caches. Such variations on zero-byte compression would be orthogonal to, and can be applied in conjunction with asymmetric cell caches to maximize opportunity for leakage saving. In this section, we characterize the distribution of zero bits in bytes to evaluate opportunity for zero-byte compression.

Figure 6 (a) illustrates the fraction all data bytes that have no one bits all the way to seven one bits. The results indicate that the applications are primarily split in their behavior. In nine of the applications about half or over half of the

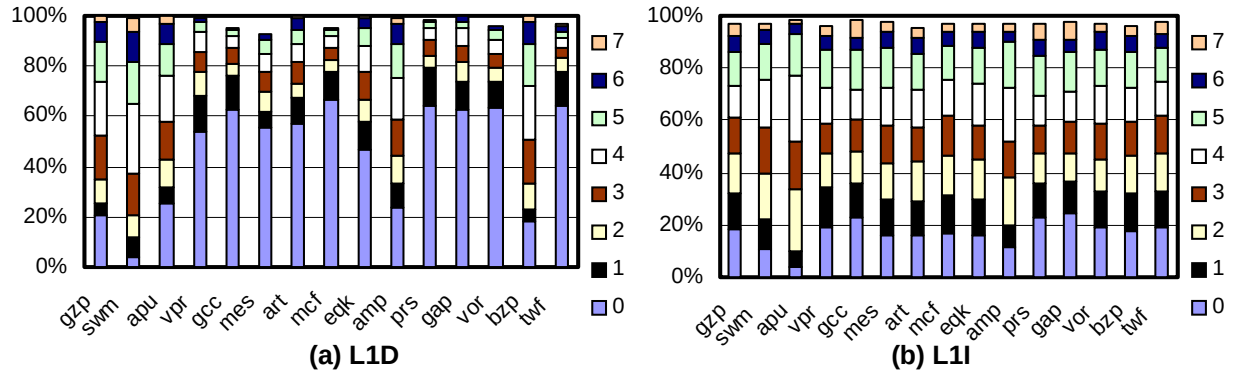

Figure 6: *Per-byte bit distribution. The figures show the breakdown of all cache resident bytes with respect to the number of their bits that are one (range is 0 to 8).*

bytes are candidates for zero-byte compression (i.e., only contain zero bits). In the rest of the five applications, however, only less than a quarter of the bytes are zeros. The compression applications, *gzip* and *bzip*, simply exhibit a more or less uniform distribution of zero bits and as such do not exhibit a large fraction of zero bytes. Similarly, the dense numerical applications *swim*, *applu*, and *ammp* also primarily exhibit bit-level zero distribution in numerical floating-point values, and as such will not benefit from zero-byte compression. However, these applications all benefit from asymmetric cell caches, by allowing asymmetric cells to save leakage at the bit-level.

Figure 6 (b) illustrates the same byte-level distribution of zeros for instruction caches. The graphs indicate that zero distributions across bytes are more uniform in instruction caches than data caches, significantly limiting opportunity for zero-byte compression. Zero bytes are especially low in *swim*, *applu*, and *ammp*, indicating a high fraction of floating-point instructions in these applications. Unlike memory, branch, and integer instructions that benefit from zero immediate values (8 bits), zero branch displacements (16 bits), and zero memory address offsets (16 bits), the floating-point instructions in Alpha mostly have one bits in opcode and function fields, limiting the likelihood of zero bytes in the instruction even if register operand specifiers are skewed toward zero.

4.5 Selective Inversion

In figure 7 we report the increase in zero bits with byte-level selective inversion. Part (a) shows this increase ignoring the inversion flag bits that are necessary. For example, in swim the fraction of zero cells increases from 52% (figure 3(a)), by about 15% (figure 7(a)) to 67%. Selective inversion is typically more effective for instructions than for data. This is expected, since as we show in section 4.2, instructions have less zeros than data. Overall, selective inversion offers *most* of the potential improvement that is possible increasing the fraction of zero bits by 6.2% and 11.5% on the average for the data and the instruction caches respectively.

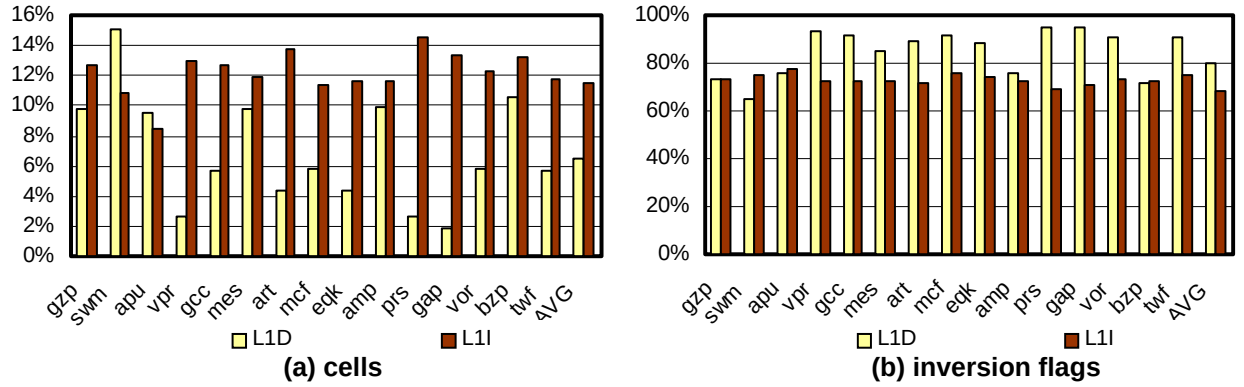

Figure 7: *Selective inversion at the byte level: (a) Increase in the fraction of zero bits ignoring inversion flag overhead. (b) Shown is the fraction of flag bits that hold a zero (measured over all flag bits). Since we introduce a flag bit per eight bits of data storage, flag bits represent an overhead of 12.5% over the original number of bit cells.*

Selective inversion introduces additional inversion flags which impose overheads on power, area and potentially latency. In this design, we introduce one inversion flag per 8 bits of the original cache. Hence, overall we introduce 12.5% more cells. Figure 7(b) shows the fraction of those flag bits that hold a zero (not inverted). On average, 80% and 70% of the inversion bits hold a zero for the data and instruction caches respectively.

Whether selective inversion reduces leakage power also depends on the relative leakage power at the zero and one bit states. As we will see in section 4.7, for the given CMOS technology and applications, the overall benefit is small. Nevertheless, we chose to present selective inversion since it bounds *worst case* leakage.

4.6 Cell Power, Performance and Stability

Using the production strength SPICE model provided by a vendor of a commercial 0.13µ, 1.2V CMOS technology we studied the performance, power and stability characteristics of our SRAM cells. Table 3 reports leakage power and performance (bitline access) under high-activity operating conditions (110C temperature). For reference, we also include measurements with low-activity (30C) and the Vth0 of the regular-Vt and the high-Vt transistors (the process also includes ultra-low-Vt transistors that are unsuitable for SRAM cell design due to stability and process variation considerations).

Temp.	Vth0 Regular	Vt0 High	Reg.-Vt Leak. Power	*Relative Leakage Reduction Compared to Regular-Vt*							*Reg.-Vt Read Latency*	*Bitline Read Latency Overhead*			
				Basic		LE		SE		High Vt		Basic	LE	SE	High Vt
				0	*1*	*0*	*1*	*0*	*1*						
110C	0.23v	0.43v	3.3nW	69.5X	1X	69.5X	6.9X	6.9X	2.0X	69.5X	217ps	12%	12%	0%	60%
30C	0.23v	0.43v	187pW	15.2X	1X	15.7X	6.8X	6.8X	2.3X	15.7X	199ps	15%	15%	0%	68%

Table 3: *Cell behavior. We report the absolute leakage power for the regular-Vt cell and the relative decrease in leakage power for the three asymmetric cells and for both possible stored values and for the all-high-Vt cell. We also report the read latency overheads for the asymmetric cells and the all-high-Vt cell.*

The regular-Vt cell's leakage power[2] at 110C is aproximately 3.3nW. We use this power as our base in reporting the leakage power of all other cells. The high-Vt cell's leakage power is 69.5 times lower. The leakage power of the asymmetric cells depends on the value they store. The LE cell behaves like the high-Vt cell when it stores a zero. Otherwise, it still reduces leakage power by a 6.9x factor. The SE cell also reduces leakage power significantly. Depending on the stored value it reduces leakage power by 6.9x or by 2.0x. As we will explain next, the SE cell is faster than the LE cell. The basic asymmetric cell does not appear to offer any particular advantage over the other two asymmetric cells.

In the final four columns we report the bitline access latency for the regular-Vt cell and the overhead incurred by the other cells. The *bitline access latency* is the time required to discharge the bitline by 200mV to 1.0V. The high-Vt cell is 60% slower than the regular-Vt cell. Using CACTI II, we estimated this delay to be approximately 16% of the overall cache access latency. This is an unacceptable increase for typical, high-performance caches. The LE cell's bitline access is 12% higher than that of the regular-Vt cell. This translates to a 3% increase in overall access latency. Finally, the SE is as fast as the regular-Vt cell. Consequently, the SE and LE cells offer different power/performance characteristics. The LE cell drastically reduces leakage power with a small performance penalty. The SE also reduces leakage energy but to a lesser extent, however, it performs as fast as the regular-Vt cell.

Besides power and performance, important considerations are stability and noise margins. Following the methodology of Hamzaoglou *et. al.* [10], and Seevinck *et al.* [18] we studied both aspects and found that they are acceptable for the regular-Vt, LE, SE and high-Vt cells. In this paper, we summarize our findings, We measured stability as the ratio *Iflip/Iread*, where *Iflip* is the current necessary to flip the cell and *Iread* is the maximum current passing through the transistors during a read. We measured this ratio under worst case assumptions for process variation and cache state. Normalizing stability over the regular-Vt cell, we measured that the LE cell is slightly less stable (0.91x), while the SE and high-Vt cells are more stable (1.07x and 1.4x respectively). We have also performed static noise margin analysis and found that all cells offer acceptable noise margins. Specifically, normalizing noise margins with respect to the regular-Vt cell we found that: The high-Vt cell's noise margins are 13% higher. The noise margins of the asymmetric cells depend on the value they store. For the LE cell, they are 15% and 18% higher when it stores a zero or one respectively. For the SE cell, they are 27% higher when it stores a zero and 4% lower when it holds a one. Overall, the

2. In contrast to dynamic power dissipation, leakage power is not associated with a dynamic event. Accordingly, we report power and not energy.

stability and noise margins of the asymmetric cells are comparable or better to those of the conventional, regular-Vt cell.

4.7 Leakage Power Reduction

Figure 8 shows the leakage power dissipation of various asymmetric cell caches. 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 cells of the data array. This is appropriate given that Simplescalar does not model a multiprogrammed environment and as a result tag bits are strongly skewed towards zero. In this section, we restrict our attention to the SE and LE cells since the basic asymmetric cell does not offer any advantage over them.

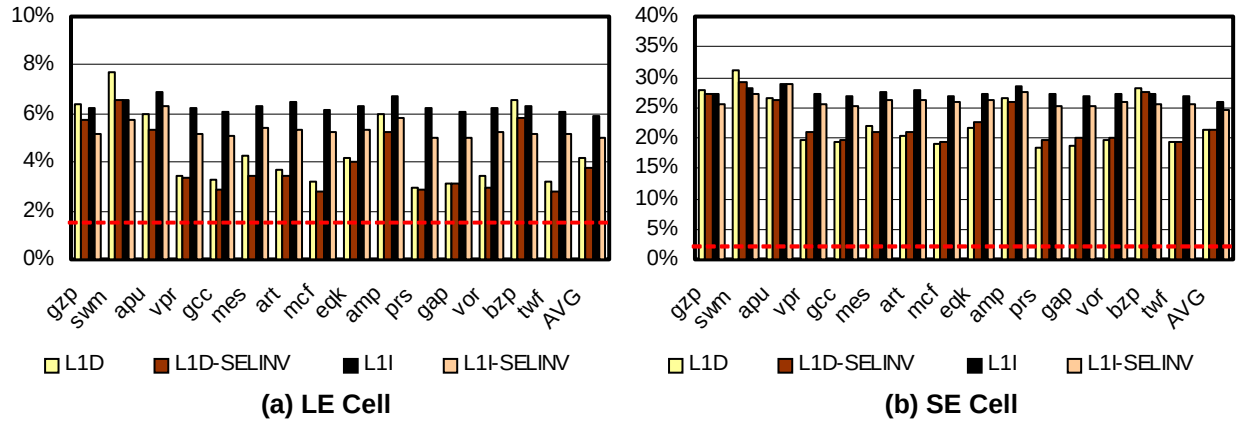

Figure 8: *Data and instruction cache leakage power with the asymmetric cells **relative** to the regular-Vt conventional cache (20% is equivalent to a 5x reduction). This is for the 1.2v Vdd technology. The leakage power dissipation of the high-Vt cell caches is 1.4% (dashed line). Lower is better.*

Part (a) shows the leakage power with the LE cell for the statically-biased cache and the cache that uses selective inversion. In all cases, leakage power is less than 7.8% of that of the conventional regular-Vt cell. On the average, the statically-biased cache reduces leakage to 4.2% and 6% of the regular-Vt cell cache for the data and instruction caches. By comparison, the high-Vt cell cache reduces leakage to about 1.4%. Selective inversion, reduces leakage to 3.9% and 5% respectively. Compared to the leakage power of the regular-Vt cell this additional reduction is unimportant. However, in relative terms it is significant bringing the asymmetric cell cache closer to the all high-Vt one.

Part (b) reports leakage power for the SE cell cache. Even for the data stream of swim that exhibited the lowest fraction of zeros, leakage power is reduced to 32% of the regular-Vt cell cache (i.e., a reduction of about 3.1x). On the average, the SE cell cache's leakage power is only 23% and 26% of that of the regular-Vt cell data and instruction caches respectively. Selective inversion does not produce a discernible advantage. In some cases, it actually increases leakage power. The leakage power for the SE cell is only 3.2x higher when it stores a one as compared to when it stores a zero. The resulting reduction in overall leakage power with selective inversion does not compensate for the overhead introduced by the inversion flags. Performing inversion on a larger than byte granularity would help. Nevertheless, the expected benefits are rather small for the technology we used. In future technologies, where the difference in leakage power between regular-Vt and high-Vt cells is much larger (e.g., 10^4x [16]) selective inversion may be useful even with the SE cell.

5 Related Work

Numerous architectural techniques for reducing power in high-performance caches have been proposed. We restrict our attention to those proposals that target leakage power. An architecture oriented, general discussion of leakage power and of its implications is given by Butts and Sohi [6]. The majority of proposals fore reducing leakage power of caches leverage a supply-gating mechanism to essentially "turn off" parts of the cache (these mechanisms use transistor stacking to exponentially reduce leakage in the inactive state, e.g., [16]). The motivation is that, ideally, only those cache cells that hold live data should be kept on. Yang *et al.*, proposed a dynamically resizing I-cache where sets are turned off [20]. Kaxiras *et al.*, proposed cache-decay where blocks are disabled when they have not been

accessed for sufficiently long periods of time [14]. Zhou *et al.*, proposed adaptive mode control that also disables blocks that have not been accessed recently where the actual interval of inactivity is dynamically adjusted [21]. 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 (misses will be serviced by the much larger higher levels of the memory hierarchy). As we have shown in section 4.3, our technique is orthogonal to all aforementioned methods. Even if it was possible to perfectly predict which blocks are not live, our asymmetric cell caches can reduce leakage significantly. Moreover, even when most cache blocks hold live data our techniques can reduce leakage power significantly.

To the best of our knowledge, no previous proposals on using bit values to reduce leakage power dissipation exists. Villa *et al.*, present dynamic zero compression where zero values are exploited to reduce dynamic power dissipation [19]. Our method attacks leakage power dissipation and relies on bit values. As we have shown in section 4.4, even if zero bits were completely disabled, our method could still reduce leakage power significantly.

Several circuit-level only proposals for reducing leakage power in caches exist. These exploit multiple or dynamic threshold voltages and/or supply voltages and transistor stacking. In their majority, these techniques trade-off performance for reduced power dissipation and are oblivious to application behavior. Previous work on multi-Vt transistor cells focused on *symmetric* cells [10,12]. A. Bhavnagarwala *et. al.*, present dynamic threshold SRAM cells [3]. Additional information on low power SRAM design and their performance and power scaling can be found in [1,4]. Gonzalez *et al.*, present an analysis of supply and threshold voltage scaling [9]. To the best of our knowledge no previous work on dual-Vt asymmetric cells exists.

6 Conclusion

Current technology voltage scaling trends are leading to high levels of leakage power dissipation in microprocessors. Because cache memories account for a large fraction of on-chip transistors, much recent research has focused on reducing leakage in caches. 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: fast transistors tend to dissipate high leakage power. Accordingly, while high-Vt cells reduce leakage power drastically they are unacceptably slow for high-performance deep-submicron caches. Consequently, previous work at the architectural-level has focused on using regular-Vt cell caches and effectively turning off those parts of a cache that do not hold live data.

In this paper, we proposed a novel approach that combines both circuit- and architecture-level techniques. Our approach drastically reduces leakage power dissipation even when most of the cache is holding live data most of the time. The key observations behind our approach are that: (1) cache-resident memory values of ordinary programs exhibit a strong bias towards zero or one at the *bit level*, and (2) leakage power dissipation depends on the actual bit value stored in the cell.

We introduced a family of high-speed asymmetric dual-Vt SRAM cell designs that exploit this bit-level bias to reduce leakage power while maintaining high performance. The two best asymmetric cells offer different performance/leakage power characteristics. The *speed enhanced* cell reduces leakage power by at least 2x and by 7x in the preferred state. It is as fast as the conventional, regular-Vt SRAM cell. By comparison, the *leakage enhanced* cell reduces leakage by at least 7x and by about 70x in the preferred state. Its bitline latency is 15% higher than the speed enhanced and the regular-Vt cells. By comparison, the all high-Vt cell reduces leakage power by about 70x while it is 60% slower than the speed enhanced and the regular-Vt cell. We proposed two cache organizations that used either a static bias towards zero, or dynamic, selective inversion to maximize the number of cache bits that are zero. The latter bounds worst case leakage power while the reduction possible with either technique depends on application behavior.

We studied the cache resident bit value behavior of most of SPEC2000 and found that there is bias towards zero. Except for dense FORTRAN applications, often more than 70% of data cache bits exhibit a strong bias towards zero. The instruction cache bits are less biased with about 60% being zero. We studied the cache resident bit behavior for two different instruction set architectures, for different caches sizes while taking into account both resident and live-only cache bits. This analysis allowed us to identify why programs tend to exhibit a strong bias towards zero bits. Moreover, we have shown that our techniques can reduce leakage even if was possible to perfectly disable all dead cache blocks and those bytes that hold a zero. We have used a production-strength SPICE model for a commercial 0.13µ CMOS technology and studied leakage, performance and stability for the asymmetric cells. Finally, we have

shown that our asymmetric cells can drastically reduce leakage power compared to the conventional, regular-Vt caches. We have found that for the given process, using selective inversion does not offer a significant advantage. Nevertheless, selective inversion bounds leakage power under worst case assumptions for the bit values stored in the caches.

While we have presented two asymmetric cells, further possibilities exist. Also, asymmetric SRAMs could be useful in other SRAM-based structures in high-performance processors. Moreover, while we used threshold voltage to selectively weaken some of the cell's transistors other alternatives may exist (e.g., resizing). Finally, it may be possible to use similar concepts in other parts of high-performance processors lowering leakage power while maintaining high-performance.

References

[1] B. S. Amrutur and M. A. Horowitz, Speed and power scaling of SRAM's, *IEEE Journal of Solid-State Circuits*, vol. 35, February 2000.

[2] B. S. Amrutur and M. A. Horowitz, A replica technique for wordline and sense control in low-power SRAM's, *IEEE Journal of Solid-State Circuits, Vol. 33, No. 8*, pp. 1208-1219, August 1998.

[3] A. Bhavnagarwala A. Kapoor and J. Meindl, Dynamic-threshold CMOS SRAM cells for fast, portable applications, In *Proc. 13th Annual IEEE Intl. ASIC/SOC Conference*, Sept. 2000.

[4] A. Bhavnagarwala and J. Meindl, Limits on CMOS SRAM Scaling, *TECHCON 2000*, September 2000.

[5] S. Borkar, Design challenges of technology scaling, *IEEE MICRO*, 19(4):23-29, July 1999.

[6] J. A. Butts and G. Sohi, A Static Power Model for Architects, In *Proc. 33rd Annual IEEE/ACM Intl. Symposium on Microarchitecture*, December, 2000

[7] D. Brooks and M. Martonosi, Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance. In *Proc. 5th Intl. Symposium on High-Performance Computer Architecture*, January 1999.

[8] R. Canal, A. Gonzalez, and J. E. Smith. Very low power pipelines using significance compression. In *Proc. 33rd Annual IEEE/ACM Intl. Symposium on Microarchitecture*, December 2000.

[9] R. Gonzalez, B. Gordon, and M. A. Horowitz, Supply and threshold voltage scaling for low power CMOS, *IEEE Journal of Solid-State Circuits*, vol. 32, August 1997.

[10] F. Hamzaoglu, Y. Ye, A. Keshavarzi, J. Zhang, S. Narenda, 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 0.13um technology generation. *In Proc. 2000 Intl. Symposium on Low Power Electronics and Design*, July 2000.

[11] John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach (2nd edition), *Morgan Kaufman*, 1996.

[12] Itoh, K., A. R. Fridi, A. Bellaouar, and M. I. Elmasry. A Deep Sub-V, Single Power-Supply SRAM Cell with Multi-Vt , Boosted Storage Node and Dynamic Load. In *Proc. of IEEE Symposium on VLSI Circuits*, June 1996.

[13] T. Kam, S. Rawat, D. Kirkpatrick, R. Roy, G. S. Spirakis, N. Sherwani and C. Peterson, EDA challenges facing future microprocessor design, *IEEE Transactions on Computer-Aided Design*, 19(12), December 2000.

[14] S.Kaxiras, Z.Hu and M.Martonosi. Cache decay Exploiting generational behavior to reduce leakage power. *In Proc. of the 27th Intl. Symposium on Computer Architecture*, July 2001.

[15] M. Lipasti, Value Prediction and Speculative Execution, *Ph.D. Thesis, Carnegie-Mellon University*, April 1997.

[16] 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 Intl. Symposium on Low Power Electronics and Design*, July 2000.

[17] G. Reinman and N. Jouppi, An Integrated Cache Timing and Power Model, *Unpublished document*, COMPAQ Western Research Lab., 1999.

[18] E. Seevinck Sr., F. J. List, and J. Lohstroh, Static-noise margin analysis of MOS SRAM cells, *IEEE Journal of Solid-State Circuits, vol. 22*, pp. 748 - 754, October 1987.

[19] L. Villa, M. Zhang, and K. Asanovic, Dynamic Zero Compression for Cache Energy Reduction, In *Proc. 33rd Intl. Symposium on Microarchitecture*, December 2000.

[20] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches. *In Proc. 7th Intl. Symposium on High-Performance Computer Architecture*, January 2001.

[21] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte. Adaptive Mode Control: A Static-Power-Efficient Cache Design. In *Proc. 2001 International Conference on Parallel Architectures and Compilation Techniques*, September 2001.