Appears in 33rd International Symposium on Microarchitecture, Monterey, CA, December 2000

Dynamic Zero Compression for Cache Energy Reduction

Luis Villa[*], Michael Zhang, and Krste Asanović
MIT Laboratory for Computer Science, Cambridge, MA 02139
{luisv|rzhang|krste}@lcs.mit.edu

Abstract

Dynamic Zero Compression reduces the energy required for cache accesses by only writing and reading a single bit for every zero-valued byte. This energy-conscious compression is invisible to software and is handled with additional circuitry embedded inside the cache RAM arrays and the CPU. The additional circuitry imposes a cache area overhead of 9% and a read latency overhead of around two FO4 gate delays. Simulation results show that we can reduce total data cache energy by around 26% and instruction cache energy by around 10% for SPECint95 and MediaBench benchmarks. We also describe the use of an instruction recoding technique that increases instruction cache energy savings to 18%.

1 Introduction

Cache accesses consume a significant fraction (30–60% [7, 11]) of total energy dissipation in modern processors. A large portion of cache energy is dissipated in driving the bitlines, which are heavily loaded with multiple storage cells, and so most cache energy reduction techniques have concentrated on reducing bitline energy. One approach is to reduce the bitline capacitance switched on each access, by using a combination of sub-banking, segmented bitlines, and hierarchical bitlines [6]. Another complementary approach is to limit the voltage swing on the bitlines during a read access by pulsing word line drivers [1, 5].

In this paper, we introduce a novel technique for cache energy reduction, *dynamic zero compression* (DZC), which exploits the prevalence of zero bytes stored in the cache. DZC adds an additional *zero indicator bit* (ZIB) to each cache byte that indicates whether the byte contains all zero bits. On a read access, we prevent bitline discharge by disabling the local word line for each byte when the ZIB is set. If the ZIB is clear, the eight data bits are read normally. On a write access, only the ZIB is written if the byte is zero, otherwise both the data bits and the ZIB are written. The ZIB is also used to disable bus drivers connecting the CPU datapath with the cache sub-banks to further reduce cache access energy.

Our initial simulations revealed that over 70% of the bits that are read from or written to the data cache are zeros. Previous work has shown how to exploit this asymmetry in the distribution of ones and zeros to reduce energy for storage arrays with single-ended read bitlines. A single-ended read bitline is typically precharged high and conditionally discharged based on the stored data bit. Register files are often built with single-ended read ports to reduce total wiring cost. The register file bit cell can be modified so that reading a zero on any port causes no bitline discharge, resulting in a savings of 27% of total register file energy [10]. Chang et. al. [12] describe another scheme for ROMs and small RAMs with single-ended bitlines that conditionally inverts stored words to reduce the total number of bitline discharges, and achieves an average of 30% energy reduction for a RAM array.

For larger SRAM array designs, differential bitlines are preferred over single-ended bitlines because they provide greater noise immunity and faster sensing. In a differential design, one of the two bitlines must be discharged for each access regardless of the stored data value. The energy penalty for reads can be reduced by employing a pulsed-word-line technique to turn off the word lines when a sufficient voltage differential has developed on the bitlines [1, 5]. Writes to an SRAM are also usually performed differentially but typically require a full voltage swing on the bitlines and hence consume considerably more energy than low-voltage-swing reads.

Our DZC technique allows us to retain the benefit of differential bitlines while taking advantage of the asymmetric distribution of 1's and 0's to reduce energy dissipation. Instead of treating each bit independently, we group multiple bits together and attach an extra zero indicator bit to indicate when the whole bit field is zero. This scheme also allows us to save energy on write accesses as well as reads since we only write the ZIB rather than the whole bit field. We similarly make use of the ZIB to reduce energy when driving the data bus between the processor and cache.

Several other techniques have been proposed that also

[*] On leave from the Centro de Innovación y Desarrollo Tecnológico en Cómputo, Instituto Politécnico Nacional - México D.F.

Benchmark Name	Instructions Executed (millions)		Symbol		Description
compress{test}	28		`comp`		An in-memory version of a common UNIX utility
li{test}	1,120		`li`		xlisp interpreter
ijpeg{test}	577		`jpeg`		JPEG 24-bit image compression standard
go{test}	17,280		`go`		An internationally ranked go-playing program
vortex{test}	10,058		`vor`		An object oriented database
m88ksim{test}	519		`m88k`		Motorola 88100 microprocessor simulator program
gcc{test}	1,497		`gcc`		Based on the GNU C compiler version 2.5.3
perl{test}	369		`perl`		A Perl Interpreter
adpcm	*enc*: 17	*dec*: 13	*enc*:`ade`	*dec*:`add`	A speech compression and decompression program
epic	*enc*: 3,417	*dec*: 176	*enc*:`epe`	*dec*:`epd`	An image data compression utility in C
g721	*enc*: 134	*dec*:1,620	*enc*:`g7e`	*dec*:`g7d`	Adaptive differential PCM voice compression
mpeg2	*enc*:14,432	*dec*:9,724	*enc*:`mpe`	*dec*:`mpd`	A player for MPEG video
pegwit	*enc*: 33	*dec*: 18	*enc*:`pee`	*dec*:`ped`	A program for public key encryption and decryption

Table 1: Benchmarks and descriptions. Each benchmark from the MediaBench Suite has two separate programs: encoding and decoding.

exploit the prevalence of zero values in data streams. Dynamic ALU width adjustment [3] exploits the zeros present in the high order bits of machine words to switch off portions of the ALU, and RAM compression schemes (e.g., [2]) exploit the large number of zero values created by the operating system clearing memory pages prior to allocating them to a new process.

The rest of this paper is structured as follows. Section 2 presents motivation and simulation results for the data cache which reveal that compressing zeros at the byte granularity gives the optimal energy reduction ratio for our benchmark set. Section 3 presents an overview of the structure of a conventional cache array and describes the necessary circuitry changes to implement the DZC technique. It also analyzes the energy consumption in both the conventional cache and the DZC cache. Section 4 presents the resulting data cache energy reduction for our benchmarks. Section 5 shows the energy savings from using DZC on the instruction cache. It then describes how we can use a recoding scheme for MIPS RISC instructions to increase the number of zero bytes, thus increasing the energy savings. Section 6 concludes the paper.

2 Data Cache Accesses

In this section, we present the distribution of zero values in data cache accesses and show how to maximize the energy savings with minimal area overhead. All our simulation numbers are for SPECint95 [4] and the integer programs from MediaBench [8] programs compiled with `gcc` version 2.7.0 for a MIPS-II-compatible processor using optimization level `-O3` and linked with a version of the `newlib` standard C library. We have extended our MIPS processor simulator to gather statistics on the presence of zero values in cache accesses. Refer to Table 1 for our benchmark workload.

We conducted a study to see how various granularities of zero compression would affect the possible energy savings. Figure 1 shows the reduction in bitline swings observed when we apply zero compression to various sized bit fields for read accesses (write accesses have a similar pattern). We show results for 32-bit, 16-bit, 8-bit, and 4-bit groups, where the figures include the bitline swings of the additional ZIBs in each group. We see that the 8-bit grouping gives the greatest savings overall. The 32-bit grouping actually increases the total number of bitline swings for the `pee` and `ped` benchmarks because there are insufficient zero words to compensate for the additional ZIB accesses required on every word. Adopting a word or half-word granularity would be difficult for a processor that allowed stores to individual bytes, while a 4-bit granularity would almost double the RAM area overhead. Thus, we will only consider the byte granularity design for the data cache.

3 Cache Circuit Design

In this section we present the circuitry we have developed for the DZC caching scheme. We first briefly discuss the design of a conventional low-power cache and each component of its energy dissipation. We then present the necessary circuit changes required to implement the DZC technique and discuss the effect of the circuit changes on area and delay.

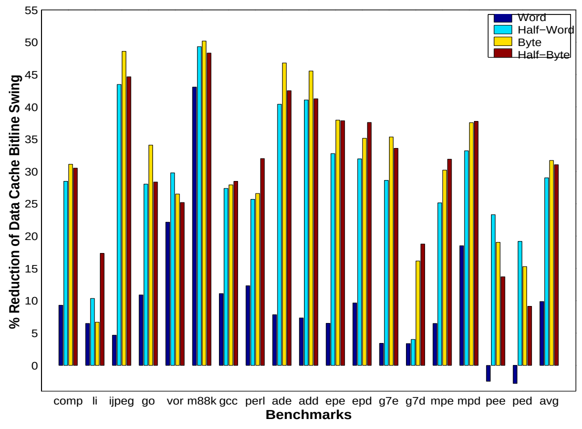

Figure 1: Reduction in the number of bitline swings for read accesses when applying dynamic zero compression to various sized bit fields.

3.1 Baseline Cache Design

We have implemented a low-power 16 KB cache in a TSMC 0.25 μm CMOS process with a nominal 2.5 V supply. The cache is direct mapped and is structured as eight sub-banks of 2 KB each. Tags for each sub-bank are kept in a separate RAM array of 24 bits for each of the 64 cache lines. Figure 2 shows a 2 KB sub-bank. All bitlines are shielded between power and ground rails to minimize capacitative coupling. On any memory access, only the appropriate sub-bank is enabled. Each sub-bank is organized as 128 rows of 128 bits. Each cache line is 32 bytes long, and is held in two consecutive rows of the sub-bank. We reduce the number of active bitlines with local word lines that enable only the required 32-bit segment of any row on any CPU access; we only enable all 128 bit columns during cache refills. In addition, a self-timed circuit is used to limit the voltage swing of bitlines during read accesses by pulsing the word lines [1, 5]; this reduces bitline swing to around 15% of full rail. The CPU to cache interface is a 32-bit bus, while the CPU datapath contains the circuitry to align and sign-extend appropriate bytes for a byte or half-word load, and also the logic to align byte or halfword store data on to the appropriate bytes within the 32-bit bus for stores. The bus employs pulsed differential low-voltage swing drivers to reduce data I/O energy.

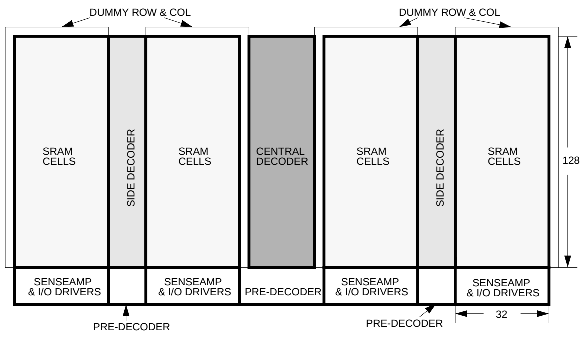

Figure 2: Structure of one 2 KB cache sub-bank.

	Read		Write	
	(pJ)	(%)	(pJ)	(%)
Total	44.4	100.0	99.1	100.0
Decoder	5.5	12.4	5.5	5.5
word lines	1.1	2.5	1.1	1.1
Tag bitlines and sense-amp	3.0	6.2	3.0	3.0
Data bitlines and sense-amp	14.5	32.7	69.2	69.9
I/O buses	12.1	27.3	12.1	12.2
Other	8.4	18.9	8.4	8.5

Table 2: Breakdown of energy consumption for 32-bit accesses in base line cache design.

Energy consumption figures for the baseline cache design were obtained from HSpice simulations of extracted layout. Table 2 shows a breakdown of the cache energy consumption for 32-bit reads and writes. Writes take over twice the energy of reads primarily because of the greater energy expended in driving the bitlines full swing. Most energy is dissipated in the bitlines and the I/O drivers, which are areas where we expect to obtain savings with the DZC scheme. We next show how we can change the cache circuitry to implement DZC.

3.2 Circuit Modifications

The primary modification to the cache circuitry is to add the zero indicator bit for each byte in the cache as shown in Figure 3. On a write to the cache, we need to check whether the eight data bits are all zero, and if so, we write only the ZIB and disable the write of the eight data bits. If the byte is not zero, we clear the ZIB and write the eight data bits normally. On a read, we want to disable the word line for each byte to avoid swinging the bitlines, thus we add local byte word line gating circuitry controlled by the ZIB. We also add control logic into the drivers connecting cache sub-banks to the CPU to avoid driving the I/O busses for zero bytes. The following sections explain each circuit modification in detail.

3.2.1 CPU Zero-Detect and Store Bus Drivers

During a cache write, the CPU detects whether each byte is zero. If so, it disables the store data bus drivers and

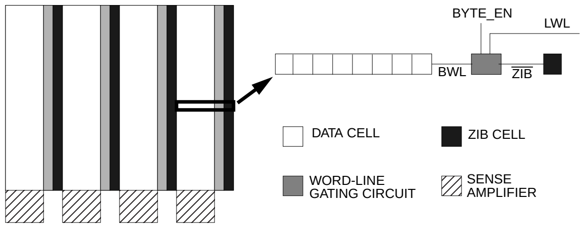

Figure 3: Organization for one 32-bit wide segment of a cache with DZC.

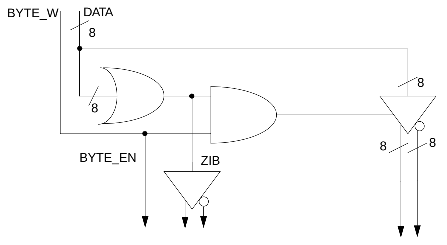

Figure 4: CPU store data driver.

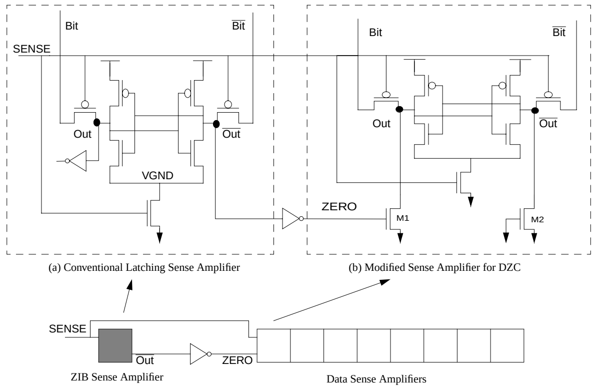
Bit
SENSE
Out
VGND
ZERO
M1
M2
(a) Conventional Latching Sense Amplifier
(b) Modified Sense Amplifier for DZC
ZIB Sense Amplifier
Data Sense Amplifiers

Figure 6: (a) Conventional Senseamp, only used for ZIB. (b) Modified Senseamp for DZC Scheme, used for data bits. The dummy inverter at node Out in (a) is only used for capacitance balancing.

only sends a one on the ZIB bus to the cache, otherwise, it enables the store data drivers and sends a zero for the ZIB. The circuit change for write access is shown in Figure 4. If the byte is not zero, it enables the tristate drivers to drive the data onto the bus. The CPU also emits the usual byte write (BYTE_W) enables for each byte. At the cache side, the byte write enables are combined with the ZIB bit to control whether the eight data bits are written (BYTE_EN).

3.2.2 Word Line Gating Circuitry

Extra word line gating circuitry is used to disable the byte word line when reading or writing zero bytes. Figure 5 shows the modified byte word line gating circuitry. On a cache write access, the cache determines whether to enable the writing of a byte based on the byte write enable signal, BYTE_EN, and the local word line, LWL. If BYTE_EN and the local word line LWL are both asserted, the byte level word line BWL will be asserted.

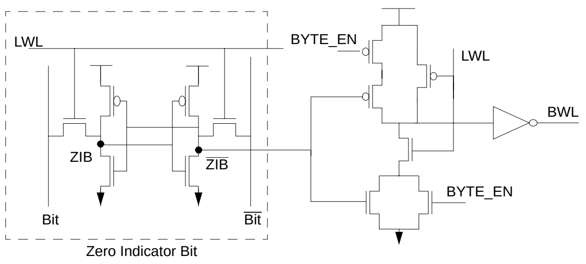

Figure 5: Modified byte word line gating circuit for DZC scheme.

On a cache read access, the byte level word line will be turned on only if the byte is not zero, which is indicated by the ZIB storage output. The CPU keeps BYTE_EN low during a read cycle. The word line gating circuitry adds little delay to read accesses because we have simply replaced the usual per 32-bit word gating circuitry with per 8-bit byte gating circuitry and resized buffers in the new fanout tree.

3.2.3 Sense Amplifier Modification

Our baseline senseamp design is a conventional latching sense amplifier with isolated bitlines as shown in the dotted box (a) in Figure 6. While not sensing, nodes Out and $\overline{\text{Out}}$ follow the values of Bit and $\overline{\text{Bit}}$. When the SENSE signal is asserted, it turns off the p-type transistors connecting Bit to Out and $\overline{\text{Bit}}$ to $\overline{\text{Out}}$ and grounds VGND. If there is a small voltage differential between Nodes Out and $\overline{\text{Out}}$, the inverter pair will amplify this differential to a full rail-to-rail signal.

The word line gating circuitry disables bitline discharge for a read of a zero byte. To avoid having the senseamp hang and burn static current, or swing due to threshold voltage mismatches or noise, we add two n-type transistors (M1 and M2) to the senseamps of the data bits to force them towards zero if ZIB is set. The modifications to the senseamp are shown in the dotted box (b) in Figure 6. The zero transistors are only used here to force a known state on the senseamp; the resulting zero value is not driven to the CPU and is not on any critical path. Shown in Figure 6, the zero transistors of the data bit senseamps are driven from the conventional senseamp used for the ZIB. The ZIB and data bits are read at the same time, i.e., they share the same sense signal. The data bits will not discharge the bitlines

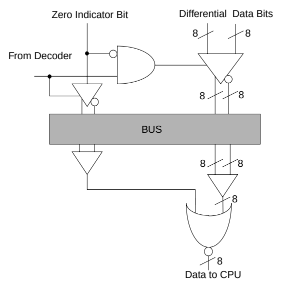

Figure 7: Cache I/O Driver

for a zero byte. If the byte is not zero, the `ZERO` signal from the ZIB senseamp will stay low, and the data bit senseamps behave exactly the same as the original one and sense the non-zero data byte. If the byte is zero, the ZIB senseamp will assert the `ZERO` signal to push the data bit senseamps into a stable zero state. Transistor `M2` balances the capacitances at the differential nodes $\overline{\texttt{Out}}$ and `Out`. Note that for a non-zero byte, the sensing time will only increase slightly due to the additional capacitative load of the drains of the zeroing transistors.

3.2.4 Cache Read Drivers

The circuit change for read access is shown in Figure 7. On the cache side, the ZIB controls the tristate enables for the data bus drivers. The ZIB begins set and conditionally clears after sensing. If it is cleared, we enable the bus drivers in the cache, otherwise they remain tristated. When the data is received in the CPU datapath, a bank of NOR gates produces a zero byte in the case of a set ZIB, or allows the data byte to propagate through if the ZIB is clear.

3.3 Energy Breakdown in DZC Cache

In order to obtain energy consumption figures, we again used HSpice simulation results from extracted DZC cache layout. In the DZC cache the energy consumption can be separated into two major components. First, there is a fixed cost that all accesses must incur regardless of data patterns, caused by the peripheral circuitry including decoders and the self-timing circuitry. The second component, caused by the discharging of the bitlines and I/O buses, varies according to the data patterns. Table 3 shows the energy consumption figures for the DZC cache. The energy figures are lumped into the read or write energy for either a zero byte or a non-zero byte. As expected, writes give much larger savings than reads, because of the greater energy used to swing the bitlines full rail.

	Read (pJ)	Write (pJ)
Zero Byte	5.0	6.7
Non-Zero Byte	11.4	26.9

Table 3: Breakdown of energy consumption for DZC cache design for zero byte or non-zero byte.

3.4 Area and Delay Overhead

Most area overhead is introduced by the ZIBs; there is very little area overhead in the senseamps and word line gating circuitry. The changes in the I/O bus drivers also add insignificant area. In total for the entire cache, the extra circuitry imposes around a 9% area overhead.

We consider read and write delay overhead separately. For a cache write access, write data is usually held in a pipeline buffer for a cycle while the cache tags are checked. The zero check can occur while the write data is waiting in the buffer and hence we expect no visible delay penalty.

For reads, the ZIB is read out in parallel with all the data bits. On the cache side, the data bits are delayed by the need to gate their tristate enables with the zero bit which adds around one FO4 gate delay. On the CPU side, a NOR gate is necessary to reconstruct zeros when the ZIB is set. In total, we estimate that DZC will add around two FO4 gate delays on a read access. The performance impact of this additional read latency depends on the degree of pipelining in the machine and on the amount of instruction-level parallelism present in the code. As a pessimistic example, for a classic RISC five-stage pipeline with an aggressive 16 FO4 delay clock cycle, the two gate delays could be spread over the two cycles of memory instruction execution (address calculation plus cache access) to give an overall cycle time penalty of under 7%. In practice, the machine pipeline structure or cache size might be modified so as not to incur as large a cycle time penalty.

4 Data Cache Results

Figure 8 presents the energy savings for the data cache using dynamic zero compression. The energy savings vary from around 12% for `li` to close to 40% for `m88ksim`, with an average of 26%. The energy reduction is less than the average bitline swing reduction of around 33% shown in Figure 1 because of the fixed peripheral circuit costs.

5 RISC Instruction Cache Results

In this section, we first present the energy savings for instruction caching using DZC, then we show how we

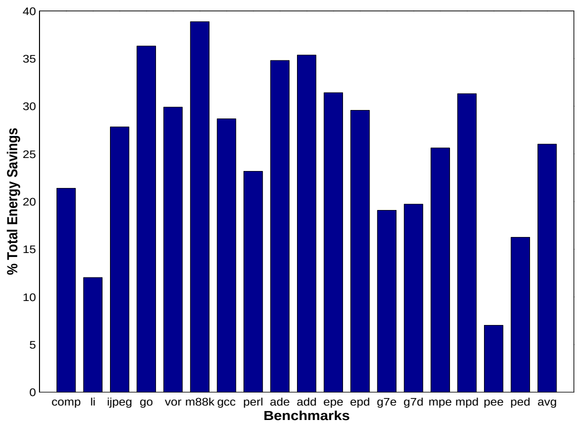

Figure 8: Data cache energy reduction obtained with dynamic zero compression.

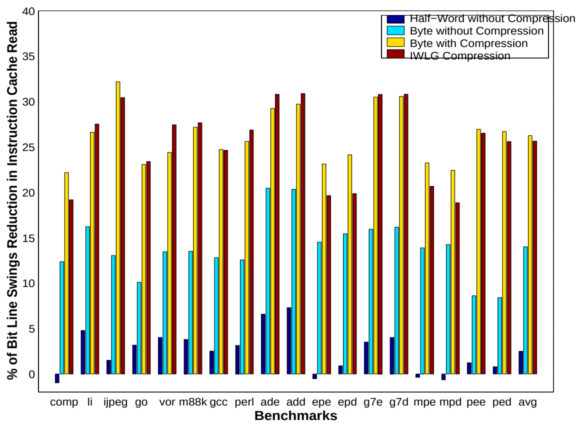

Figure 9: Bitline swing reduction in instruction fetch.

can employ an instruction recoding technique [9] to obtain greater savings in the instruction cache.

The first two bars in each column of Figure 9 show the bitline swing reduction for read accesses with half-word and byte granularities. It is clear that byte granularity gives larger savings, with an average of 15% reduction in bitline swings.

Energy savings are directly proportional to the percentage of zero-valued bytes in cache accesses. To increase the percentage of zero bits, we also experimented with an instruction recoding technique for the MIPS instruction set, previously presented in [9]. This technique compresses commonly used MIPS instructions into fewer bits. For example, many ALU operations use the same register for one source and the destination, and so can be compressed into a two-address form. Another example is that many branches compare against zero and have short offsets, and so can be compressed from the MIPS form which includes a large 16-bit offset and allows comparisons between any two arbitrary registers. The compressed form of these instructions still occupy a full 32-bit slot plus additional encoding bits in the instruction cache, but all unused bits are packed to the low end of the instruction word and set to zero. The instruction recoding takes place at cache refill. The instruction fetch stage is unchanged because the instructions are still addressed as fixed-size units, but instruction decode must be expanded to handle the larger number of instruction types.

The results for instruction word line gating (IWLG) presented in [9] assumed that the instruction word line could be segmented at arbitrary bit positions to disable bitline swings from the unused zero bits. It was found that the greatest reduction in bitline swings was achieved with the 32 instruction bits grouped into three fields of 16 bits, 7 bits, and 9 bits, used to hold three compressed instruction sizes of 16, 23, and 32 bits. The upper 16-bit field is designed to always contain the opcode and the source register specifiers and is never gated so that there is no additional read latency. This scheme allows full speed instruction decode and register access. The remaining 16 bits emerge with some delay, but only contain immediate values or destination register specifiers and hence are not in the decode stage critical path. The bitline swing reduction for the IWLG technique averages around 26%, shown as the last bar in each column in Figure 9.

Although the IWLG scheme gives a large bit swing reduction with effectively no fetch delay penalty, it requires a custom word line gating circuit that prevents using the same cache RAM arrays to store byte-addressed data. A small addition to the DZC compressor circuit allows the same cache RAM array to hold either compressed instructions or data. The modified DZC scheme adds an instruction compressor that is used on cache refills and which employs instruction sizes of 16 bits, 24 bits, and 32 bits to match the byte granularity of word line gating. The DZC instruction recoding scheme changes the meaning of the ZIBs of the two lower bytes to indicate whether the byte is used in the instruction instead of whether the byte is zero-valued. Notice that there exist instances where the byte is used in the instruction but is actually zero-valued, causing bitline swings for zero bytes.

The bitline reduction for the DZC instruction compression scheme is shown as the third bar of each column in Figure 9 and is actually slightly larger than the IWLG scheme at 27%. The DZC achieves slightly greater savings on average because it can compress zero bytes in the top 16 bits of each instruction while the IWLG scheme avoids gating these bits to avoid any read delay.

We experimented with all three different designs, a DZC

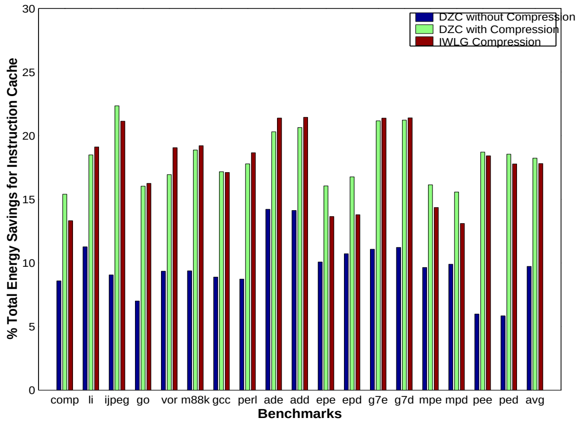

Figure 10: Energy savings in instruction fetch.

cache without instruction compression, DZC cache with instruction compression, and the IWLG cache with custom wordline gating. The energy savings are summarized in Figure 10. The DZC cache without compression averages 10% saving. The optimized IWLG cache achieves energy savings of 17.8%. The DZC cache with instruction compression achieves a slightly greater energy saving of 18.2%.

The DZC cache without compression would be suitable for a unified primary cache system. The IWLG scheme is appropriate for systems with a dedicated instruction cache, as it gives large energy savings and no fetch delay. The DZC compressed cache is suitable for systems with a single cache system that can be variably partitioned between primary instruction and data cache.

6 Conclusion

We proposed a dynamic zero compression technique to reduce cache energy by taking advantage of the high occurrence of zero-valued bytes in the cache. This technique uses a small amount of additional hardware embedded in the RAM array to detect and eliminate the reading and writing of zero bytes. Simulation results show a 26% energy reduction on data cache accesses and 10% on instruction cache accesses when applied to a low-power cache design with sub-banking and low-swing bitlines. The area overhead is about 9% and the latency overhead is around two gate delays. For partitionable primary caches, instruction cache savings can be improved to 18% by using an instruction recoding scheme.

Although this paper has concentrated on the energy savings possible in the primary caches, the zero indicator bits can be propagated throughout the lower levels of the memory hierarchy to provide additional energy savings in memory access and bus transfers.

7 Acknowledgments

Thanks to members of the MIT SCALE and RAW groups and the anonymous reviewers for feedback and comments on earlier drafts of this paper. This work was partly funded by DARPA PAC/C award F30602-00-2-0562 and by a Instituto Politécnico Nacional COFAA grant.

References

[1] B. Amrutur and M. Horowitz. Techniques to reduce power in fast wide memories. In *Symposium on Low Power Electronics*, volume 1, pages 92–93, October 1994.

[2] C. Benveniste, P. Franaszek, and J. Robinson. Cache-memory interfaces in compressed memory systems. In *Solving the Memory Wall Problem Workshop, ISCA-27*, Vancouver, Canada, June 2000.

[3] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *HPCA-5*, January 1999.

[4] Standard Performance Evaluation Corporation. Spec95, 1995. `http://www.spec.org`

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

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

[7] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid State Circuits*, 31(9):1277–1284, September 1996.

[8] C. Lee, M. Potkanjak, and W. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communication systems. In *Micro-30*, North Carolina, December 1997.

[9] M. Panich. Reducing instruction cache energy using gated wordlines. Master's thesis, Massachusetts Institute of Technology, August 1999.

[10] J. Tseng and K. Asanović. Energy-efficient register access. In *Proc. XIII Symposium on Integrated Circuits and Systems Design*, Manaus, Brazil, September 2000.

[11] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimization using SimplePower. In *ISCA-27*, Vancouver, Canada, June 2000.

[12] B.-I. Park Y.-S. Chang and C.-M. Kyung. Conforming inverted data store for low power memory. In *ISLPED*, pages 91–93, August 1999.