Energy Reduction in Queues and Stacks by Adaptive Bitwidth Compression

Vasily G. Moshnyaga
Department of Electronics Engineering and Computer Science, Fukuoka University
8-19-1 Nanakuma, Jonan-ku, Fukuoka 814-0180, Japan
vasily@fukuoka-u.ac.jp

ABSTRACT

A new micro-architectural technique to reduce energy dissipated by queues and stacks is proposed. Similarly to related research which targets the transition activity in bit-lines, the technique is based on bitwidth compression. However unlike them, it utilizes the fixed accessing order embodied in queues and stacks to exploit input data correlation. The technique dynamically adjusts the required bitwidth to the number of bits which changed in comparison to the last access. It is neither restricted to specific bit-patterns such as zero-byte or precharging value and works efficiently on read and write without large area, timing or power overhead. Simulations show that using this technique, we can save the energy of instruction queue by up to 30% and the energy of video data queue by 20%.

1. INTRODUCTION

1.1 Motivation

Queue and Stack are two specific register file organizations used in modern general purpose processors and DSPs to implement First-In-First-Out (FIFO) and Last-In-First-Out (LIFO) data accessing policies, respectively. In microprocessors, for example, queues are emploied for instruction issue [1],[2] register renaming[3], instruction/data buffering[4], etc, while stacks are put to use in instruction reordering, address generation[5]. The HP3000, for instance, is keyed to stack processing. In DSPs and special architectures, queues and stacks are basic foreground memory blocks [6], [7],which store and rearrange input data and keep results before writing them into background storage. With the trend towards wider instruction issue and larger instruction windows, these application specific register files grow in size, consuming a substantial amount of power. A 32-bit 128-entry instruction queue alone can take as much as 25% of the total energy dissipated by a typical superscalar processor [1]. Therefore, lowering the energy consumption in queues and stacks is important.

ISLPED'01, August 6-7, 2001, Huntington Beach, California, USA.

To first order, the dynamic energy dissipated by a queue or a stack may be expressed as the sum of energy consumed per access (E_i) over all memory accesses (N), or $Energy = \sum_i^N E_i$. In contemporary SRAM-based register file, most of the energy per access E_i is burned when driving the bit-lines which are heavily loaded with multiple storage cells[8]. Given the average physical capacitance, switched per bit (C), the supply voltage (V), the signal transition probability per bit (p), and the bitwidth per access (n_i), this energy can be expressed as, $E_i = n_i \times p \times C \times V^2$. Thus, we have $Energy = \sum_i^N n_i \times p \times C \times V^2$. Though reducing energy dissipation amounts to all the above factors, the energy saving obtained by lowering the number of memory accesses (N), the bitwidth (n) and the bit transition probability (p) is fairly independent of integration technology and less expensive.

In this paper we focus on power reduction through bitwidth truncation and propose a new micro-architectural technique that dynamically adjusts the bitwidth of queues and stacks per access according to data input data variation.

1.2 Related research

Several approaches to transition activity reduction in register files have been recently proposed. A common one is to partition the file into multiple sections, and selectively gate the clock signal to those sections which are not addressed in the current clock cycle[9]. Such clustered register file organization have been embodied in commercial processors such as the Alpha 21264[10]. To support variable data representation and instruction formats, the bitwidth of the registers, queues and stacks is usually fixed to a maximal value, regardless of the actual data variation on inputs.

In reality, the transition activity varies along the bitwidth. Figure 1 shows the transition profile of the Alpha-21264's instruction queue simulated on SPEC95int *mpeg_encoder* benchmark. Because of multiple instructions with both operands less than or equal to 16-bit, the instruction queue makes more transitions in the first half of the bitwidth than in the second. Activity profile for the register file input has a completely different shape (see RF curve). Due to massive arithmetic with subword parallelism and high data correlation, the bit activity is unevenly distributed along the subwords. The Most Significant Bits (MSBs) make 2-3 times less transitions than 31-st or 63-d bits. Leveraging this bit uneveness by storing the minimal number of changed bits can significantly reduce energy consumption.

Although there have been several attempts to exploit this data asymmetry for processing elements[11]-[13], data and

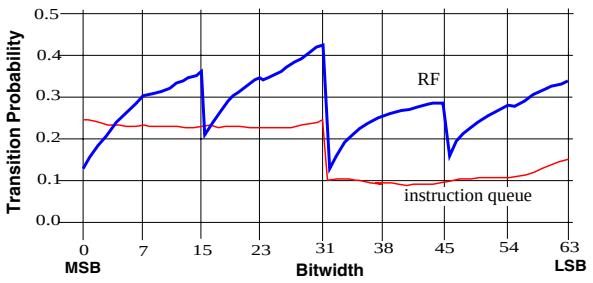

Figure 1: Activity profile for the Alpha-21264 instruction queue and data queue on the SPEC95int benchmark

address busses[14]-[16], only a few works have targeted memory structures. Hu and Martonosi[17] exploit temporal locality of accesses by buffering the register file with a "value aging buffer". Since the buffer is smaller than a typical register file, it has better power characteristics. Park et al[18] propose to conform memory data to the precharging value. Depending on the value, a decision is made to store a datum in either true or complement form, so as to minimize the bit-line discharging during the memory read. To indicate the form in which values are being stored, an extra line is added to each memory byte. Villa et al. [19] advocate a Dynamic Zero Compression, which attaches an extra bit to each data byte to indicate whether the byte contains all zero bits. The technique allows to shrink the accessing bitwidth to only one bit if the byte is zero. Its performance, however, strongly depends on the number of zero bytes in accesses. Canal, et al.[20] describe a compression scheme that maintains only significant bytes with 2 or 3 extension bits appended to indicate significant byte positions. The efficiency of this scheme also is conditional to amount of bytes with all zeros or ones. Yet timing overhead of the bit-encoder may be a problem.

The above techniques proved to be useful in general purpose applications, which involve representation of short operands in the full word size. However, their application for the special-purpose hardware is not effective. The problem is that the number of bytes which have all zeros or ones is small. In video streams, for example, the percentage of such bytes is less than 1%.

1.3 Contribution

In this paper we present a novel approach for bitwidth reduction in application specific register files, such as queues and stacks. Similarly to existing techniques, the approach is based on bit compression. However unlike them, it exploits the unique accessing features of queues and stacks in order to minimize the transition activity in bitlines. The method dynamically adjusts the active bitwidth in queues and stacks to those bit segments which vary in comparison to the previous access. It is neither restricted to the special bit-patterns such as zero-byte or value of precharging and works efficiently on read and write without a large area, timing or power overhead.

The rest of this paper is structured as follows. Section 2 presents the approach and describes the necessary circuitry changes to implement it. Section 3 shows the simulation results. Conclusions are given in Section 4.

2. ADAPTIVE BITWIDTH COMPRESSION

2.1 Main idea

The approach we propose is based on two key features of queues and stacks: (1) the fixed accessing order and (2) the uneveness of bit-activity along the bitwidth. In queue, the order of read repeats the order of writing, while in stack, the order of reading and writing are inverse to each other. Due to high correlation of data, adjacent words (or subwords) in a stream differ by a few bits. So, reading and writing all the bit-lines per each access is unnecessary. Moreover, it is prohibitly expensive from the energy perspective, since each bit-line activation dissipates a large amount of power.

By analogy with the other techniques, which target the amount of transitions occurring on the bit-lines, we attach an additional bit to a group of multiple bits. However in our formulation, this extra bit indicates the equality between the current datum and the datum which was the most recently accessed. Our main idea is to split the bitwidth into m multibit groups and compare on each access, the corresponding groups of the current entry, X, and the entry, Y, accessed recently. (Note that the size of the groups may vary). If in a group j, the values X_j and Y_j have all bits equal in pairs, we prevent writing X_j into the queue (stack), while storing only the equality bit (EB_j) to indicate that the current value of the group (j) equals the value of the last access. Formally, the EB_j is defined as:

$$EB_j = \begin{cases} 1 & \text{if } AND_{i=0}^{i=k-1}(x_{j,i} \odot y_{j,i} \& c_{j,i-1}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where k is the size of the group, $c[j,0] = 1$ and $\odot$ is the Equivalence operator defined by the logic function $F = ab + a'b'$.

When X_j and Y_j are not equal, both the EB_j and all bits of X_j are stored. On a read access, we prevent bitline discharge by disabling the local word line for the group j when the EB_j is zero. If the EB_j bit is set to one, all the bits in the group are read normally. Thus instead of accessing the fixed n-bit width, we use an adaptively one, whose size dynamically is changed from $(n+1)$ bits to $(k+1)$ bits according to input data variation.

2.2 Implementation scheme

2.2.1 Queue

Figure 2 outlines an implementation scheme for 8-bit two-port decoder-based queue, whose bitwidth is segmented in two groups of 4 bits each. (The shifter-based implementation is similar). On a write access, each decision logic compares in pairs the corresponding four bits of the input value X_{in} with the bits of the last data written to the queue (registers I_1, I_2 keep copies of these bits) and generates the EB_j signal. The zero EB_j disables the corresponding segment (j) of the queue's bit-lines, reducing the active bitwidth in the segment to (EB_j). Otherwise, both the four bits of X_{in} and the EB_j are written to the queue. On a read access, the zero EB_j bit disables the local word line to the bit-segment j at the output. The disabled part of the output word is taken from the output register O_j, whose content is renewed when EB_j is set to one.

Figure 3 illustrates how the scheme works, assuming that the data is written to the queue sequentially according to the numbers depicted on the left. As figure shows, only the

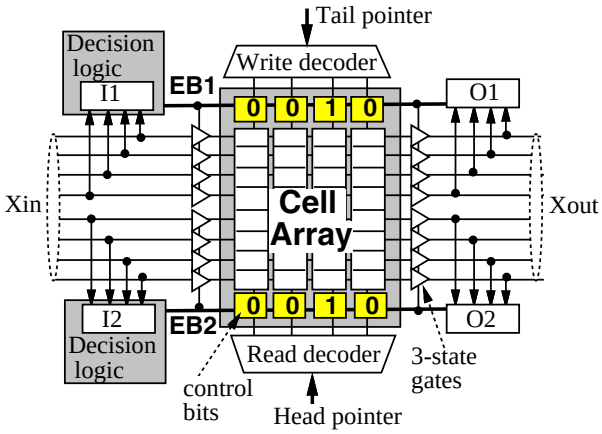

Figure 2: Implementation scheme for queue

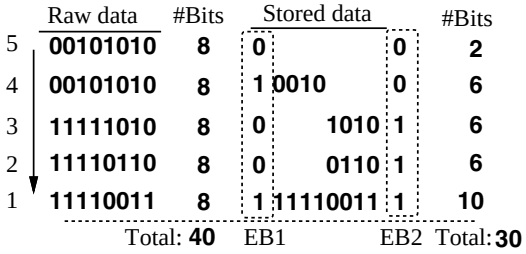

Figure 3: An illustration of data compression in queue

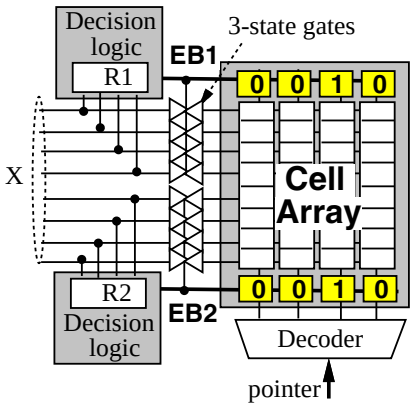

Figure 4: Implementation scheme for stack

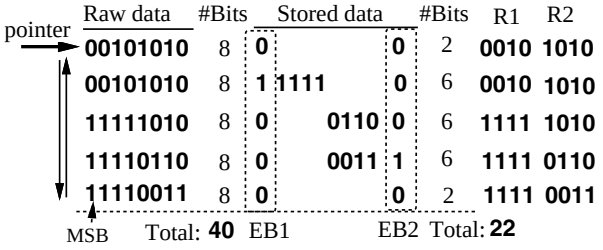

Figure 5: An illustration of data compression in stack

first word of the stream requires full $n+1$-bit representation, while all the others are stored in the compressed form. Numbers at the bottom indicate the total number of bits which have to be activated to store the data stream in the queue.

2.2.2 Stack

In comparison to queue, the application of the proposed technique to stack is rather simpler. Since read and write accesses use the same pointer, the implementation scheme involves m decision units only, as shown in Figure 4. Because data is pushed onto the stack and "popped" off the stack in reverse order (last-in-first-out), and the registers R_j keep the copy of last data accessed, the stack is accessed only when its input is changed. Figure 5 illustrates the compression on the same data stream as in Fig.3. The columns on the right display the content of registers $R1$ and $R2$. We assume that each write operation pushes the stack pointer up, and each read pops it down. When the first word 11110011 of the stream comes to the stack, we save it on registers $R1, R2$, without writing into the stack. Since the MSBs of the next three words are equal to $R1$, we keep the $R1$ unchanged until the forth word appears at the input. When "popping" the data off the stack, we read the registers $R1, R2$, but not the stack. The stack bit-lines are read if and only if either $EB1$ or $EB2$ is set to one, i.e. when the register values are renewed.

2.2.3 Decision logic

Figure 6 depicts the internal circuitry of the Decision Logic for the case of $k = 4$. In this figure, $y0 - y3$ are the flip-flops to store the MSB of the last word written into the queue (stack). The operation proceeds as follows. When CLK is low, the nodes $c1, c2, c3, c4$ are precharged to V_{dd}. During computation, when CLK goes high, the n pull-down transistor on the left and the n pull-up transistor on the right turn on. Upon comparison of input bit signals in pairs, each XNOR generates $g_i, (i = 1, 2, ..)$, making the low potential from the left to propagate through the transistor chain to the right. If x_i and y_i are equal, then g_i is high; in this case, the node c_{i-1} is connected to the node c_i; conditionally discharging it. The low c_i gates the clock signal, clk_i, to the bit, i, discarding it from the further computations. Note that the clock clk_i to the bit i will be gated if and only if all the senior bits $0, 1, ..., i-1$ are also gated.

3. EVALUATION

We chose to evaluate the activity savings of the proposed approach by simulating benchmarks from the SPECint95[21] and MediaBench[22] suites. Table 1 outlines the benchmarks used in the study. The second column in the table shows the number of instructions considered. The benchmarks were simulated using a modified version of SimpleScalar [24] program, compiled for the Alpha instruction set using optimization options as specified by the SPEC Makefile: `-migrate -std1 -O5 -ifo -nonshared`. The simulator has been extended to gather statistics on presents of equal values in instruction queue accesses. The main parameters of the processor architecture are described in Table 2. As can be seen,

Table 1: Benchmarks and descriptions

Benchmark name	Instr. (10^6)	Description
gcc	1497	GNU C compiler version 2.5.3
li	1,120	Lisp interpleter
mpeg(enc)	14,432	MPEG2 video encoder
mpeg(dec)	9,724	MPEG2 video decoder
ijpeg	577	JPEG 24-bit image compression
adpcm	17	Adaptive speech compression

our model architecture closely resembles Alpha 21264[10] with the key difference to that we do not model a clustered organization.

We first investigated how various granularities of adaptive bitwidth compression would affect the possible activity savings. Figure 7 shows the reduction in bitline transitions observed when we apply the adaptive bitwidth compression to various sized bit fields of instruction queue on read access (write accesses have a similar pattern). We show the results for byte, half-word, word, and double-word groups. All the figures include the bitline transitions of the additional EBs in each group. We see that the byte granularity gives the greatest savings overall.

In order to obtain energy consumption figures, we designed a prototype 32-word 16-bit queue (Fig.8) in 0.35μm CMOS process (2.5V voltage supply) and used HSPICE simulation of the extracted layout. In total, the extra circuitry imposes around an 8% area overhead. The delay penalty introduced by the 8-bit decision logic is around two FO4 gate. Table 3 shows a breakdown of the queue energy consumption. The main observation that write take over twice the energy of read primarily because of the greater energy burned in driving the bitlines full swing. As we see, most energy is dissipated in the bitlines.

Based on simulations we found that though the proposed approach disables eight lines from a nine-line wide bit slice, there is a fixed cost that all accesses must incur regardless of data patterns caused by the decoders, decision making mechanism and clocking circuitry. Also, discharging of the bitlines and I/O busses depends on the data patterns. Table 4 outlines the energy dissipation numbers in one queue slice for either all the 8-bitlines are disabled or not. As expected,

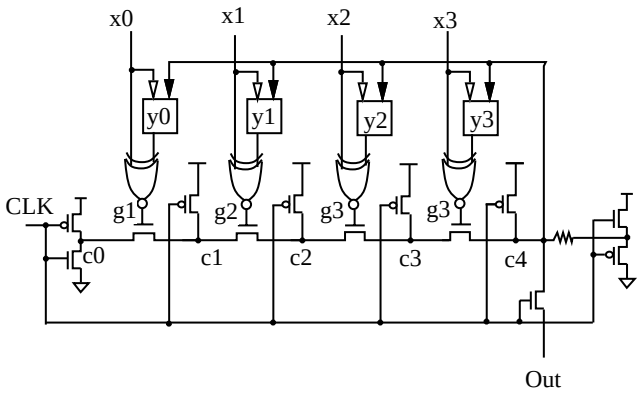

Figure 6: The decision logic circuitry

Table 2: Processor configuration

Parameter	Configuration
Decode width	4 instructions/cycle
Issue width	4 instructions/cycle
Commit.width	4 instructions/cycle
Branch predictor	Combined, Bimodal 4K table 10bit history, 4K Chooser 32 entry Return Stack
I-Queue	80x 64bit, divided in ready queue and first-use queue as in [23]
Issue policy	Out-of-order
Func.Units	4 int.ALU, 1-int.mul/div
	2 FP ALU, 1-FP mul/div
I-cache	64KB, 2-way, 32bit 1 cycle hit, 3 cycles miss
D-Cache L1	64KB, 32-bit, 2-way, 32B blocks 1 cycle hit, 3 cycles miss
U-Cache L2	1M, 64-bit, 4 way, LRU 32B blocks, 12 cycles latency

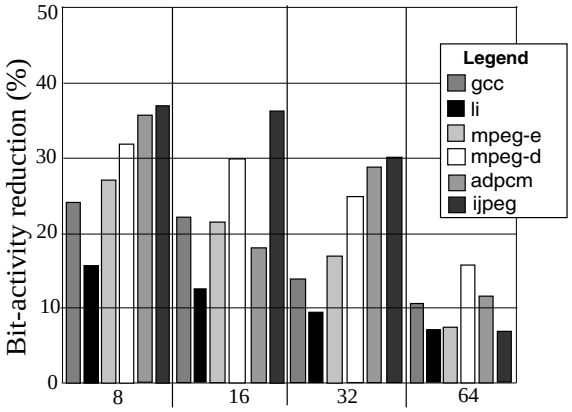

Figure 7: Reduction in bitline transitions vs. various sized bit fields

writes give much larger savings than reads, because of the greater energy used to swing the bitlines full rail.

Using the above numbers, we estimated the energy savings for the instruction queue (Figure 9). The savings vary from around 12% for *li* to 30.5% for *jpeg*, with an average of 23%. Due to the fixed peripheral circuit costs, the energy reduction is around 25% less than the average bitline transition activity reduction shown Fig.7).

In order to obtain energy figures for data queue, we simulated the behavior of 8-bit 256-word video input queue of real-time MPEG2 pixel processor[25] using six standard video streams: *Bicycle, Carousel, Football, Cheer Leader, Mobile & Calendar, and Table Tennis* (frame size: 352×240 pixels; frame rate: 30 f/s). The first 150 frames of each stream were considered. In the study, we experimented with three levels of compression granularity: (1) Byte; (2) Half-Byte; and (3) compressing the four most significant bits only.

Figure 10 shows the maximum reduction in bitline swing observed in the data queue for the tested sequences. Figure

Table 3: Breakdown of energy consumption for 16-bit queue

Circuit	Read		Write	
	(pJ)	(%)	(pJ)	(%)
Decoders	6.4	13.8	6.4	6.8
Word lines	1.3	2.8	1.3	1.5
Data lines	16.9	36.5	65.4	68.9
I/O buses	13.6	29.4	13.6	14.3
Decision logic+clock	8.1	17.5	8.1	8.5
Total	46.3	100	94.8	100

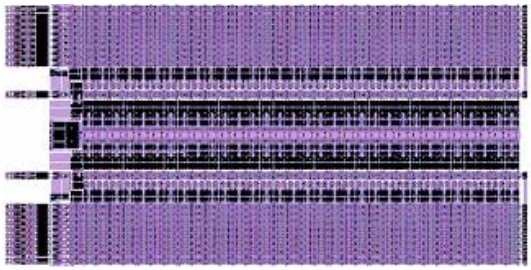

Figure 8: Layout of the prototype queue

11 presents the energy savings in the queue obtained computed for each sequence. We see that the numbers vary with sequences and the granularity level. Due to high transition activity of the Least Significant Bits, the byte compression becomes ineffective. Shrinking the compression granularity to the four Most-Significant Bits (denoted by MSB in the figure) achieves results similar to (1-2% difference) the half-byte compression. However, it almost halves the area and delay overhead in comparison to the double-sliced implementation and, therefore, is more beneficial for the data queue implementation.

The energy savings for the MSB compression vary from 20.2% for *Bicycle* to 14.1% for *Mobile*, with an average of 16.5%. Note, that this is the result of applying the proposed technique to the narrow 8 bit queue. For the wider queues we expect better results. Currently we are investigating this issue for the 16- and 32-bit data queues utilized in video and audio processors.

Finally, to compare the proposed approach with the related techniques, we simulated the performance of the Dynamic Zero Compression[19] (byte granularity). Due to extremely small number of zero-bytes per frame for each video sequence (maximum 109 out of 84480), the use of DZC for the 8-bit data queue was completely ineffective (the activity reduction factor was 0.1%). For the instruction queue, the DZC was able to reduce the bitline activity up to 26% (the

Table 4: Energy consumption of the bitline slice

Function	Read (pJ)	Write (pJ)
Disabled	8.4	12.9
Enabled	23.5	50.4

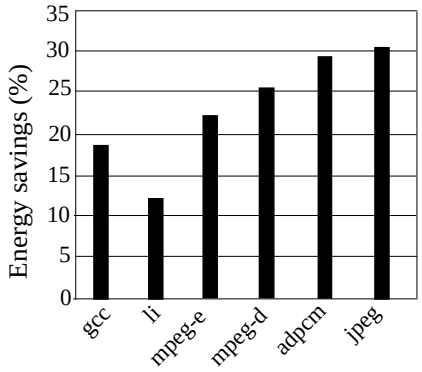

Figure 9: Energy savings for the instruction queue

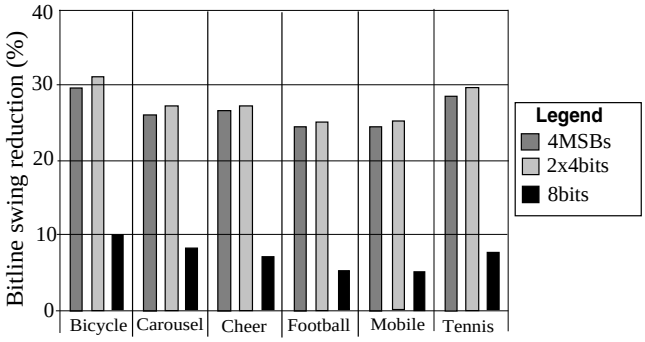

Figure 10: The maximum bitline swing reduction in the data queue

adpcm benchmark). However, it is less than the activity reduction factor (37%) obtained by our technique.

4. CONCLUSIONS

A new approach has been presented for reducing power consumption of queues and stacks. In particular, we targeted the reduction of the number of transitions occurring on the bitlines when similar values are sequentially accessed. Our approach eliminates unnecessary bitline activation by dynamically adjusting the bitwidth to the pixel variation. The proposed adaptive bitwidth compression allows us to retain the benefit of differential bitlines while taking advantage of the asymmetric distribution of transition activity along the bits to reduce energy dissipation. Instead of treating each bit independently, we group multiple bits together and attach an equality bit to indicate when the whole bit field repeats the value previously accessed. This approach also enables us to save energy on write accesses as well as reads since we only write the equality bit rather than the whole bit field. Simulations have shown that it can save as much as 30% of energy consumed by the queues in comparison to the full resolution access, without affecting the quality of results.

The proposed power reduction scheme is strongly application oriented. In fact, it is based on sequential feature of the read/write accesses which may not be a case for a general-purpose register files, where the accesses are of a random nature.

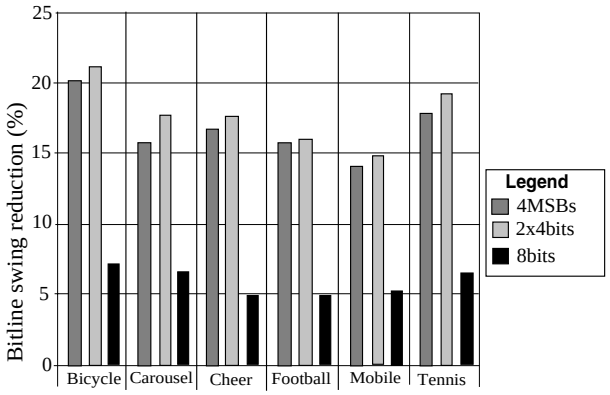

Figure 11: Energy reduction for the data queue

5. ACKNOWLEDGMENTS

This work was sponsored in part by The Ministry of Education, Science and Culture of Japan, Grant No.10650335 and Mitsubishi Electric Corporation, Grant T000666TK.

6. REFERENCES

[1] D. Folegnani and A. Gonzalez, 'Reducing power consumption of the issue logic. *Proc. Workshop on Complexity-Effective Design*, (held in conjunction with ISCA-27) June 2000.

[2] A. Buyuktosunoglu, S. Schuster, D. Brooks, P. Bose, P. Cook, D. Albonesi. An adaptive issue queue for reduced power at high performance, *Proc. Workshop on Power Aware Computing Systems (PACS)*, (held in conjunction with ASPLOS-IX) November 2000.

[3] B. Bishop, T. Kelliher, M. J. Irwin. The design of a register renaming unit", *Proc. Great Lake Symp.*, March 1999.

[4] S. Wallace, N. Dagli, and N. Bagherzadeh. Design and implementation of a 100MHz reorder buffer", *Proc. Workshop on Power Aware Computing Systems (PACS)*, (held in conjunction with ASPLOS-IX) November 2000.

[5] K. Yeager. The MIPSR10000 superscalar microprocessor. *IEEE Micro*, 16(2):28–40, April 1996.

[6] A. Ferrari, et al., An ASIC chip set for parallel fuzzy database minning. *IEEE Micro*, 16(4):60–67, December 1996.

[7] I. Verbauwhede, etc. In-place memory management of algebraic algorithms on application specific processors, *Algorithms and Parallel VLSI Architectures*. Elseiver Sc. Publ., pages 353–362, 1991.

[8] B. Amrutur, Techniques to reduce power in fast wide memories. *Proc. 1994 Int.Symp.on Low Power Electronics*, pages 92–93, October 1994.

[9] K. Farkas, N. Jouppi and P. Chow, Register file design consideration in dynamically scheduled processors", *Technical report*, Compaq Western Research Lab., 1995.

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

[11] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. *Proc. 5-th Int. Symp. on High Performance Computer Architecture*, January 1999.

[12] V. G. Moshnyaga. An MAB truncation scheme for low-power video processors. *Proc. IEEE Int. Symp. Circuits and Systems*, (4), pages 291–294, June 1999.

[13] J. Y. Tong, D. Nagle, and R. Rutenbar. Reducing powery optimizing the necessary precision range of floating point arithmetic. *IEEE Trans. VLSI Systems*,8(3):273–286, June 2000.

[14] M. R. Stan and W. P. Burleson. Low-power encodings for global communication in CMOS VLSI. *IEEE Trans. VLSI Systems*, 5(4): 444–455, December 1997.

[15] E. Musoll, T. Lang, and J. Cortadella. Working zone encoding for reducing the energy in microprocessor address busses. *IEEE Trans. VLSI Systems*, 6(4): 568–572, December 1998.

[16] L. Benini, G. De Micheli, E. Macii, M. Poncino, S. Quer. Reducing power consumption of core based systems by address bus encoding. *IEEE Trans. VLSI Systems*, 6(4): 554–562, December 1998.

[17] Z. Hu and M. Martonosi. Reducing register file power consumption by exploiting value lifetime characteristics. *Workshop on Complexity Effective Designs (WCED)*, (held in conjunction with ISCA-27), June 2000.

[18] B. -I. Park Y. -S. Chang and C. -M. Kyung. Conforming in-verted data store for low power memory. *Proc. Int. Symp. on Low Power Electronics and Design*, pages 91–93, August 1999.

[19] L. Villa, M. Zhang and K. Asanovic Dynamic zero compression for cache energy reduction. *Proc.33rd Int. Symp. on Microarchitecture*, December 2000.

[20] R. Canal, A. Gonzalez, and J. E. Smith. Very low power pipelines using significant compression. *Proc. 33rd Int. Symp. on Microarchitecture*, December 2000.

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

[22] C. Lee, M. Potkanjak, and W. Mangione-Smith. Media-bench: A tool for evaluating and synthesizing multimedia and communication systems. *Proc. 30-th Int. Symp. on Microarchitecture*, December 1997.

[23] R. Canal and A. Gonzalez. A low-complexity issue logic. *Proc. ACM Int. Conference on Supercomputing*, pages 327–335, June 2000.

[24] D. Burger, T. Austin, and S. Benett. Evaluating future microprocessors: the SimpleScalar Tool Set. Technical report TR-1308, Univ. Wisconsin-Madison CS Dept., July 1996.

[25] T. Matsumura, H. Segawa, S. Kumari, et al. A chip set for programmable real time MPEG2 MP@ML processor. *IEICE Trans. Electronics*, E-81-C (5): 680–694, May 1998.