A Highly Configurable Cache for Low Energy Embedded Systems

CHUANJUN ZHANG
San Diego State University
and
FRANK VAHID and WALID NAJJAR
University of California, Riverside

Energy consumption is a major concern in many embedded computing systems. Several studies have shown that cache memories account for about 50% of the total energy consumed in these systems. The performance of a given cache architecture is determined, to a large degree, by the behavior of the application executing on the architecture. Desktop systems have to accommodate a very wide range of applications and therefore the cache architecture is usually set by the manufacturer as a best compromise given current applications, technology, and cost. Unlike desktop systems, embedded systems are designed to run a small range of well-defined applications. In this context, a cache architecture that is tuned for that narrow range of applications can have both increased performance as well as lower energy consumption. We introduce a novel cache architecture intended for embedded microprocessor platforms. The cache has three software-configurable parameters that can be tuned to particular applications. First, the cache's associativity can be configured to be direct-mapped, two-way, or four-way set-associative, using a novel technique we call way *concatenation*. Second, the cache's total size can be configured by shutting down ways. Finally, the cache's line size can be configured to have 16, 32, or 64 bytes. A study of 23 programs drawn from Powerstone, MediaBench, and Spec2000 benchmark suites shows that the configurable cache tuned to each program saved energy for every program compared to a conventional four-way set-associative cache as well as compared to a conventional direct-mapped cache, with an average savings of energy related to memory access of over 40%.

Categories and Subject Descriptors: B.3.2 [**Memory Structures**]: Design Styles; C.5.4 [**Computer System Implementation**]: VLSI System

General Terms: Design, Performance

Additional Key Words and Phrases: Cache, configurable, architecture tuning, low power, low energy, embedded systems, microprocessor, memory hierarchy

This research was supported by the National Science Foundation (grants CCR-9876006 and CCR-0203829) and by the Semiconductor Research Corporation (grant 2003-HJ-1046G).
F. Vahid is also with the Center for Embedded Computing Systems at UC Irvine.
Authors' addresses: C. Zhang, Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA 92182; email: chzhang@cs.ucr.edu; F. Vahid and W. Najjar, Department of Computer Science and Engineering, University of California, Riverside, CA 92521.

1. INTRODUCTION

Designers of embedded microprocessor platforms have to compromise between performance, cost, and energy dissipation. Caches may consume up to 50% of a microprocessor's power [Malik et al. 2000; Segars 2001]. Creating the best cache architecture is thus important and involves selecting the amount of associativity, the total cache size, and the cache line size, among many other architectural options. These parameters greatly impact the cache's hit rate (the percentage of accesses that find the desired data in the cache) and the energy (power times time) consumed in accessing cache. Energy comes from not only the power consumed when accessing the cache, but also the time and power spent transferring data to/from the next level of memory during a cache miss, plus the power consumed by the idle processor during such a miss.

Associativity divides a cache into several ways, each of which is looked up concurrently during a cache access. A cache with only one way is known as direct mapped. For some programs, increasing the number of ways to two or even four improves a cache's hit rate [Hennessey and Patterson 1996]; beyond four ways, the improvement is typically not as great. However, more ways means more concurrent lookups per access, and hence more energy per access—a direct-mapped cache uses only about 30% of the energy per access as a four-way set-associative cache [Reinman and Jouppi 1999]. Performance-oriented applications want the highest associativity possible. Energy-oriented applications want the associativity such that the energy savings from added ways, due to improved hit rate, outweigh the energy increase per access.

Cache size is the total number of data storage bytes in the cache, independent of the cache's organization. Larger size may improve the hit rate for some programs, at the expense of more power consumed to fetch (dynamic power) and to hold (static power) the data. Performance-oriented applications benefit from a large size cache. Energy-oriented applications want the size such that the energy savings from increased capacity outweigh the energy increase from more power.

Line size is the number of bytes moved to and from the next level of memory during a miss. Typical line sizes are 16, 32, or 64 bytes. When a program exhibits higher spatial locality, then a larger line size cache can reduce cache misses. However, without spatial locality, a large line size fetches many unnecessary bytes, which not only lengthens cache fill time, but may also evict needed bytes from the cache, thus increasing off-chip memory accesses and stalls. Because microprocessors for desktop computers serve many applications, they include caches that are a compromise.

Since an embedded system typically executes just a small set of applications for the system's lifetime (in contrast to desktop systems), we ideally would like to tune the architecture to those specific applications. However, an architecture tuned to a particular set of applications may perform terribly for other applications, representing an important dilemma facing system architects.

One option that microprocessor vendors use to solve this dilemma in embedded systems is to manufacture multiple versions of the same processor, each with a cache architecture tuned to a specific class of applications. Another option

is to provide a synthesizable core rather than a physical chip, so that an embedded system designer can synthesize a cache architecture tuned to the intended application. Both options increase the microprocessor unit cost. The second option also suffers from a longer time to market [Semiconductor Industry Association 1999]. The variety of cache architectures found in modern embedded microprocessors, summarized in Table I, illustrates that the dilemma of deciding on the best cache architecture for mass produced microprocessors has yet to be solved.

We introduce a novel configurable cache architecture that largely reduces the dilemma by incorporating three configurable cache parameters, which are configured by setting a few bits in a configuration register. The cache can be configured in software as either direct mapped, two-way, or four-way set associative, while still utilizing the full cache capacity. We achieve such configurability using a new technique we call *way concatenation* [Zhang et al. 2003a]. The cache's ways can also be shut down, to adjust total size. The cache line size can be configured, using a technique we call *line concatenation* [Zhang et al. 2003b], to be 16, 32, or 64 bytes, with an underlying physical line size of 16 bytes.

All these configurable features are achieved at the cost of a very small amount of performance overhead, and negligible size overhead, compared to a regular four-way set-associative cache, as verified not only by our estimates using the CACTI model [Reinmann and Jouppi 1999], but also by our own physical layout of the cache in a 0.18 micron CMOS technology.

In this paper, we provide the details of our configurable cache, including way concatenation, way shutdown, and line concatenation, discussing the performance and area overhead imposed by such configurability. We demonstrate significant energy savings compared to nonconfigurable caches, for applications drawn from the Powerstone [Malik et al. 2000], MediaBench [Lee et al. 1997] and the Spec2000 [SPECBENCH 2002] benchmark suites.

The paper is organized as follows. In Section 2, we examine the relation between performance and energy consumption. In Section 3, we discuss previous work. We introduce the way concatenate cache architecture in Section 4. We discuss way shutdown in Section 5. We present line concatenation in Section 6. In Section 7, we explain how to use a configurable cache. Section 8 provides conclusions.

2. CACHE ENERGY VERSUS PERFORMANCE

2.1 Energy Evaluation

There are two main components that result in energy dissipation in a CMOS circuits, namely static energy dissipation due to leakage current, and dynamic energy dissipation due to logic switching current and the charging and discharging of the load capacitance. Dynamic energy per cache access equals the dynamic power of all the circuits in the cache multiplied by the time per cache access; the static energy of a cache equals static power multiplied by the time.

Table I. Instruction and Data Cache Sizes, Associativities, and Line Sizes of Popular Embedded Microprocessors

	Instruct. Cache			Data Cache				Instruct. Cache			Data Cache		
Processor	Size	As.	Line	Size	As.	Line	Processor	Size	As.	Line	Size	As.	Line
AMD-K6-IIIE	32K	2	32	32K	2	32	Motorola MPC8540	32K	4	32/64	32K	4	32/64
Alchemy AU1000	16K	4	32	16K	4	32	Motorola MPC7455	32K	8	32	32K	8	32
ARM 7	8K/U	4	16	8K/U	4	16	NEC VR4181	4K	DM	16	4K	DM	16
ColdFire	0–32K	DM	16	0–32K	N/A	N/A	NEC VR4181A	8K	DM	32	8K	DM	32
Hitachi SH7750S (SH4)	8K	DM	32	16K	DM	32	NEC VR4121	16	DM	16	8K	DM	16
Hitachi SH7727	16K/U	4	16	16K/U	4	16	PMC Sierra RM9000X2	16K	4	N/A	16K	4	N/A
IBM PPC 750CX	32K	8	32	32K	8	32	PMC Sierra RM7000A	16K	4	32	16K	4	32
IBM PPC 7603	16K	4	32	16K	4	32	SandCraft sr71000	32K	4	32	32K	4	32
IBM750FX	32K	8	32	32K	8	32	Sun Ultra SPARC Iie	16K	2	N/A	16K	DM	N/A
IBM403GCX	16K	2	16	8K	2	16	SuperH	32K	4	32	32K	4	32
IBM Power PC 405CR	16K	2	32	8K	2	32	TI TMS320C6414	16K	DM	N/A	16K	2	N/A
Intel 960IT	16K	2	N/A	4K	2	N/A	TriMedia TM32A	32K	8	64	16K	8	64
Motorola MPC8240	16K	4	32	16K	4	32	Xilinx Virtex IIPro	16K	2	32	8K	2	32
Motorola MPC823E	16K	4	16	8K	4	16	Triscend A7	8K/U	4	16	8K/U	4	16

As means associativity, *DM* means direct mapped. Size is total cache size in bytes (K means kilobytes). *U* means instruction and data caches are unified. *Line* is line size in bytes.

Sources: Microprocessor Report and data sheets of various microprocessors.

Dynamic energy constitutes the main part of the energy dissipation at micron-scale technology, but static energy dissipation is going to account for an increasing portion of total energy in nanoscale technology. We consider both of them in our energy evaluation.

Energy consumption due to accessing off-chip memory should not be disregarded, since fetching instruction and data from off-chip memory is energy costly because of the high off-chip capacitance and large off-chip memory storage. Also, when accessing the off-chip memory, the microprocessor stalls while waiting for the instruction and/or data and this waiting still consumes some energy. Thus, our equation for computing the total energy due to memory accesses is as follows:

$$energy\_mem = energy\_dynamic + energy\_static \tag{1}$$

where

$$\begin{aligned} energy\_dynamic &= \underline{cache\_hits} * \underline{energy\_hit} + \underline{cache\_misses} * energy\_miss, \\ energy\_miss &= \underline{energy\_offchip\_access} + \underline{energy\_uP\_stall} \\ &\quad + \underline{energy\_cache\_block\_fill} \\ energy\_static &= \underline{cycles} * \underline{energy\_static\_per\_cycle}. \end{aligned}$$

The underlined terms are those we obtain through measurements or simulations. We compute *cache_hits* and *cache_misses* by running SimpleScalar [Burger and Austin 1997] simulations for each cache configuration. We compute *energy_hit* of each cache configuration through simulation of circuits extracted from our layout (which happened to reasonably match earlier work we did using the CACTI model to compute such energy).

Determining the *energy_miss* term is challenging. The *energy_offchip_access* value is the energy of accessing off-chip memory, and the *energy_uP_stall* is the energy consumed when the microprocessor is stalled while waiting for the memory system to provide an instruction or data. *energy_cache_block_fill* is the energy for writing a block into the cache. The challenge stems from the fact that the first two terms are highly dependent on the particular memory and microprocessor being used. To be "accurate," we could evaluate a "real" microprocessor system to determine the values for those terms. While accurate, those results may not apply to other systems, which may use different processors, memories, and caches. Therefore, we chose instead to create a "realistic" system, and then to vary that system to see the impacts across a range of different systems. We examined the three terms of *energy_offchip_access, energy_uP_stall*, and *energy_cache_block_fill* for typical commercial memories and microprocessors. The *energy-cache-fill* is the energy of filling instruction/data to the cache, which we measure using SPICE simulation. The *energy_uP_stall* is the energy consumed by a stalled microprocessor. We estimated the *energy_uP_stall* as the standby energy of a microprocessor. The *energy_offchip_access* includes the energy consumed by off-chip bus and off-chip DRAM memory. We calculated the off-chip bus energy considering the voltage, capacitance and switching of the off-chip bus. Energy consumed by off-chip DRAM energy depends on technology and manufacturer. We surveyed typical commercial SDRAM providers in the

markets and decided to use a range of the DRAM instead of a specific product of a DRAM manufacturer. We found that *energy_miss* ranged from 50 to 200 times bigger than *energy_hit*. Thus, we redefined *energy_miss* as

$$energy\_miss = k\_miss\_energy * energy\_hit.$$

Based on our examination of various real systems, we considered situations of *k_miss_energy* equal to 50 and 200.

Finally, *cycles* is the total number of cycles for the benchmark to execute, as computed by SimpleScalar, using a cache with single cycle access on a hit and using 20 cycles on a miss. *Energy_static_per_cycle* is the total static energy consumed per cycle. This value is also highly system dependent, so we again consider a variety of possibilities, by defining this value as a percentage of total energy including both dynamic and static energy:

$$energy\_static\_per\_cycle = k\_static * energy\_total.$$

k_static is a percentage that we can set. Low-power CMOS research has typically focused on dynamic power, under the assumption that static energy is a small fraction of total energy—perhaps less than 10%. However, for deep submicron, the fraction is increasing. For example, Agarwal et al. [2002] reports that leakage energy accounts for 30% of L1 cache energy for a 0.13-micron process technology. To consider this CMOS technology trend, we evaluate the situations where *k_static* is 30% and 50% of the total energy.

In this paper, all energy plots use *k_miss*_energy = 50 and *k_static* = 30%. We discuss the impact of the larger values for those constants, while the plots for those larger values can be found in Zhang et al. [2003a, 2003b].

To verify that our estimation method is reasonable, we present actual values for a particular system. We use the low-power 64-Mbit SDRAM manufactured by Samsung (model K4S643233E) operating at 2.5 V, 55 mA, and 100 MHz. The total number of bytes read from off-chip memory is 32 bytes, using a cache line size of 32 bytes. It takes 20 cycles to send out the address and 4 cycles to read one word (4 bytes). Then it takes the total of 52 cycles to read 32 bytes from the off-chip memory. The energy per memory access is 2.5 V $\times$ 50 mA $\times$ 520 nS = 65 nJ. We calculate the energy consumed by off-chip bus as follows. The capacitance load is 30 pF per pin [Smith 1997] and the voltage is 2.5 V. The bus is 32 bits wide and roughly half the 32 bits switch during a transmission. The energy per switch is $1/2 \times V^2 \times C$. Then the energy consumption of 32 bytes of data and 32 bits address is $1/2 \times 2.5^2$ V $\times$ 30 pF $\times$ $(32 \times 8 + 32) \times 1/2 = 13.5$ nJ. We use an ARM920T processor [Segars 2001], which has active power of 100 mW at 100 MHz. Assume the processor consumes 10% of the active power when the microprocessor is stalled. The stalled energy consumed during access to off-chip memory is 100 mW $\times$ 10% $\times$ 520 nS = 5.2 nJ. We measured the cache read and refill energy as 0.827 nJ and 0.451 nJ, respectively. We can calculate the *k_miss_energy* as *k_miss_energy* = (65 nJ + 13.5 nJ + 5.2 nJ + 0.445 nJ)/0.827 nJ = 101.7. Thus, we see that our ranging *k_miss_energy* from 50 to 200 does cover the actual value for this particular system of 101.7.

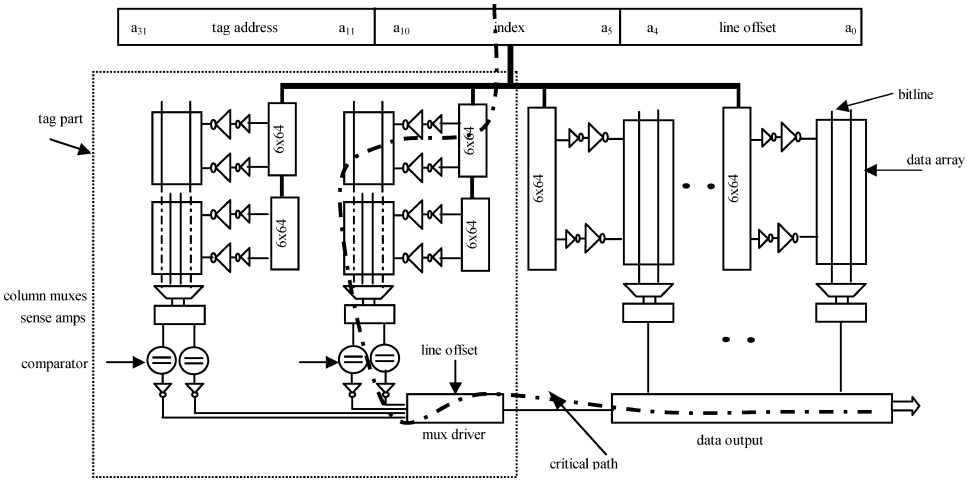

Fig. 1. A four-way set-associative cache architecture with the critical path shown.

2.2 Base Cache Architecture

After examining typical cache configurations of several popular embedded microprocessors, summarized in Table I, we chose to use a base cache of 8 Kbytes having four-way set-associativity and a line size of 32 bytes. The base cache is the cache architecture that we will later extend to be configurable.

Figure 1 depicts the architecture of our base cache. The memory address is split into a line-offset field, an index field, and a tag field. For our base cache, those fields are 5, 6, and 21 bits, respectively, assuming a 32-bit address. Being four-way set-associative, the cache contains four tag arrays and four data arrays (only two data arrays are shown in Figure 1). During an access, the address' index field is decoded to simultaneously read out the appropriate tag from each of the four tag arrays, while the index field is decoded to simultaneously read out the appropriate data from the four data arrays. The decoded lines are fed through two inverters to strengthen their signals. The read tags and data items are fed through sense amplifiers. The four tags are simultaneously compared with the address' tag field. If one tag matches, a multiplexor routes the corresponding data to the cache output.

2.3 Cache Parameter Impact on Energy and Performance

Using multiple ways increases energy substantially, since the tag and data arrays of every way are accessed simultaneously. Yet increasing the associativity improves the cache hit rate and hence performance. For example, the average miss rate for the SPEC92 benchmarks is 4.6% for a one-way (in the remainder of this paper, we will sometimes refer to a direct-mapped cache as a one-way cache) 8 Kbytes cache, 3.8% for a two-way 8 Kbytes cache, and only 2.9% for four-way 8 Kbytes cache [Hennessey and Patterson 1996]. Though these differences may appear small, they in fact translate to big performance differences, due to the large cycle penalty of misses (which may be dozens of cycles). Thus, although energy per cache access may be higher for a four-way cache than

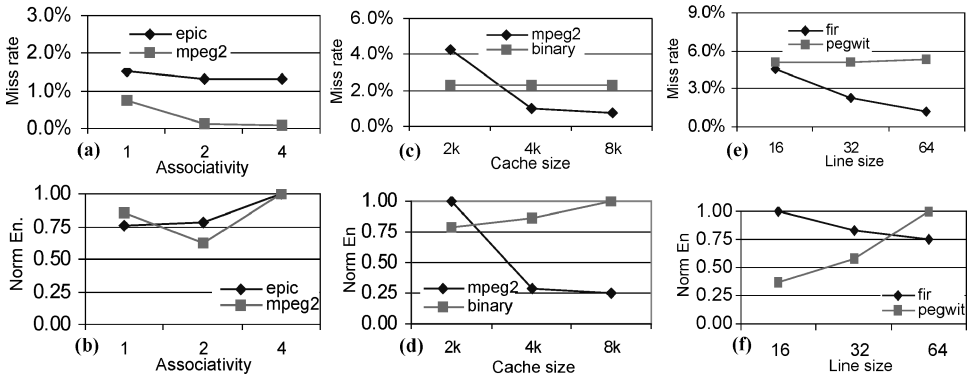

Fig. 2. Energy consumption for different associativities, cache sizes, and line sizes, for a selection of examples. *Norm En.* stands for normalized memory access energy.

for a one-way cache, that extra energy may be compensated for by the reduction in energy from reduced accesses, due to fewer misses, to the next level of memory.

Although greater associativity may increase hit rates on the average across numerous benchmarks, for particular programs, the greater associativity may have little improvement on hit rate, thus resulting in extra energy without a corresponding performance benefit. For example, Figure 2(a) shows the miss rates for two MediaBench benchmarks, *epic* and *mpeg2*, measured using SimpleScalar [Burgar and Austin 1997] and configured with an 8-Kbytes data cache, with a line size of 32 bytes, and having one, two, or four-way set-associativity. Note that the hit rates for both are better for two ways than for one way, while the additional improvement using four ways is very small. Figure 2(b) shows memory access energy (as computed by Eq. (1) for these two examples, demonstrating that a two-way cache gives lowest energy for *mpeg2*, while a one-way cache is best for *epic*. Thus, note that a lower miss rate does not always translate to lower energy. Also note that the energy differences between different cache configurations for a single program can be quite large—up to 40% in these examples.

A larger sized cache consumes more power (both dynamic and static) than a smaller one. If an application does not need all the cache capacity, then shutting down part of the cache will save power. On the other hand, if a smaller cache results in a significant increase in the miss rate, then the savings from the smaller cache will be outweighed by the extra off-chip memory access power dissipation. Figure 2(c) shows the data cache miss rate of two benchmarks, *mpeg2* and *binary*, for a direct mapped, line size of 32 bytes, and a size of 2, 4, or 8 Kbytes cache.The miss rate of *binary* remains almost constant for the different cache sizes, which means a 2 Kbytes cache is enough for that benchmark. However, the miss rate for *mpeg2* increases sharply when the cache size is decreased from 4 to 2 Kbytes. Figure 2(d) shows the normalized energy consumption of the data cache for the two benchmarks. The 2 Kbytes data cache is obviously the best for *binary*, and the 8 Kbytes cache is the best for *mpeg2*. The energy difference for *mpeg2* is significant—up to 75%.

Cache line size also plays an important role in energy dissipation. Figure 2(e) shows the data cache miss rate of two benchmarks *fir* and *pegwit* for an 8-Kbytes data cache, with one way, and line sizes of 16, 32, and 64 bytes. Increasing the cache line size does not decrease the miss rate of *pegwit*. Because a larger cache line size will incur extra off-chip memory accesses, *pegwit* consumes the highest energy at line size 64 bytes, as shown in Figure 2(f). On the other hand, the miss rate of *fir* decreases significantly with the increase of cache line size. The energy dissipation is also the least at a line size of 64 bytes for *fir*.

From the above three examples, we can see that the basic three cache parameters—cache associativity, cache size, and cache line size—have a significant impact on both performance and energy. No particular set of values for those parameters is the best for all the benchmarks. Thus, a configurable cache that would allow an embedded system designer to choose the cache parameters based on a particular application's specific characteristic could result in significant energy savings.

3. PREVIOUS WORK

Previous work can be categorized into three areas: cache architectures that save dynamic energy, cache architectures that save static energy, and configurable cache architectures.

Many cache architectures that save dynamic energy do so by modifying the lookup procedure in a set-associative cache to reduce the number of internal memory arrays accessed. Phased-lookup cache [Edmonson and Rubinfield 1995; Hasegawa et al. 1995] uses a two-phase lookup, where all tag arrays are accessed in the first phase, but then only the one hit data way is accessed in the second phase, resulting in less data way access energy at the expense of longer access time. Way predictive set-associative caches [Inoue et al. 1999; Powell et al. 2001] access one tag and data array initially, and only access the other arrays if that initial array did not result in a match, again resulting in less energy at the expense of longer average access time. Reactive-associative cache (RAC) [Batson and Vijaykumar 2001] also uses way prediction and checks the tags as a conventional set-associative cache, but the data array is arranged like a direct-mapped cache. Since data from the RAC proceeds without any way-select multiplexor, the RAC achieves the speed of a direct-mapped cache, but consumes less energy than that of a conventional set-associative cache. A pseudo set-associative cache (PSAC) [Calder et al. 1996] is physically organized as a direct-mapped cache. Upon a miss, a specific index bit is flipped and a second access is made to the cache using this new index. Thus, each location in the cache is part of a "pseudo-set" consisting of itself and the location obtained by flipping the index bit. A PSAC thus achieves the speed of a direct-mapped cache and the hit rate of a two-way cache, at the expense of slower access time than a two-way cache due to the sequential accessing of the ways. Dropsho et al. [2002] discussed an accounting cache architecture that is based on the resizable selective ways cache proposed by Albonesi [1999]. The accounting cache first accesses part of the ways of a set-associative cache, known as a primary access. If there is a miss, then the cache accesses the other ways, known as a secondary

access. A swap between the primary and secondary accesses is needed when there is a miss in the primary and a hit in the secondary access. Energy is saved on a hit during the primary access. Filter caching [Kim et al. 1997] introduces an extremely small (and hence low power) direct-mapped cache in front of the regular cache. The idea of a filter cache is that if most of a program's time is spent in small loops, then most hits would occur in the filter cache, so the more power costly regular cache would be accessed less frequently—thus reducing overall power, at the expense of performance loss that occurs when the filter cache misses.

Other cache architectures that save dynamic energy do so by adjusting the cache's line size (although most of that work actually seeks to improve performance). Some prefabricated microprocessor chips support static line size configuration. For example, the MIPS R3000/R4000 [MIPS 2002] has a configurable cache line size. Actually, the hardware architecture uses a fixed physical line size [Veidenbaum et al. 1999], but the number of words replaced on a miss could be varied. Some recent work focuses on the advantages of dynamically sizing cache lines. Witchel and Asannovic [2001] proposed a software-controlled cache line size. A compiler specifies how many data to fetch on a data cache miss. Two hardware implementations are given to support the compiler-controlled cache. Veidenbaum et al. [1999] proposed a dynamic mechanism to adapt cache line size to a specific application's behavior during the execution of applications. Based on monitoring the accesses to the cache line, a hardware-based algorithm decides the future cache line size. They achieved 50% reductions in memory traffic compared to a 32-byte line size. Inoue and Kai [2000] proposed a dynamic variable line size cache, which exploits the high memory bandwidth of on-chip merged DRAM/logic chips by replacing a whole cache line in one cycle. They improve performance and save energy, achieving a 75% energy delay product reduction over a conventional memory path model, taking advantage of on chip memory. We note that this high bandwidth on-chip memory may not be available in typical embedded systems.

Other work focuses on cache architectures that save static energy. Due to VLSI technology advances, static energy dissipation is accounting for an increasingly larger portion of total microprocessor energy consumption. Apart from multiple threshold (MTCMOS) and dual-Vt circuit level techniques, Ye et al. [1998] proposed to use more than one turned off transistors connected in series to reduce the standby leakage power dissipation, such as in a 2 inputs NAND gate, the leakage current is smaller when both nMOS transistors are turned off than that when only one nMOS transistor is turned off. Using this technique, gated-V_{DD} [Powell et al. 2000] inserts an extra pMOS transistor between the voltage source and the SRAM cells to shut off the unused on-chip cache lines, achieving 62% energy-delay reductions with minimal performance degradation. Because much of a chip's area and transistors are devoted to on-chip cache (e.g., 60% in the StrongArm [INTEL 2002]), gated-V_{DD} has been widely used by many researchers to shut off part of the cache. DRI [Yang et al. 2001] cache dynamically resizes a cache by monitoring the miss rate of the cache and shutting off part of the sets of the cache for a particular application. Tadas and Chakrabarti [2002] proposed to shut off subbanks of an instruction cache

and microblocks of both instruction and data caches, achieving a 22–81% static energy reduction for an instruction cache and a 17–65% for a data cache for SPECJVM98 benchmarks. Cache line decay [Kaxiras et al. 2001] dynamically turns off cache lines that have not been visited for a designated period, reducing the L1 cache leakage energy dissipation by 4× in SPEC2000 applications without impacting performance. Zhou et al. [2001] proposed to dynamically determine the time interval for deactivating the cache lines, achieving an average of 73% instruction cache lines and 54% of data cache lines put into sleep mode with an average instruction per cycle impact of only 1.7% for 64 Kbytes caches. In contrast with the shutting off mechanism that loses the information in the cache, a drowsy cache [Flautner et al. 2002] keeps the unused cache line in a low-power mode by lowering the SRAM source voltage while retaining the contents of the cache. 80 to 90% of the cache lines can be put into drowsy mode without affecting the performance more than 1% from benchmarks of SPEC2000.

Researchers have recently begun to suggest configurable cache architectures. Ranganathan et al. [2000] proposed a configurable cache architecture for a general-purpose microprocessor. When used in media applications, a large cache may not yield benefits due to the streaming data characteristics of media applications. In this case, part of the cache can be dynamically reconfigured to be used for other processor activities, such as instruction reuse. Kim et al. [2001] proposed a multifunction computing cache architecture, which partitions the cache into a dedicated cache and a configurable cache. The configurable part can be used to implement computations, for example, FIR and DCT/IDCT, which takes advantage of on-chip resources when an application does not need the whole cache. Smart memory [Mai et al. 2000] is a modular reconfigurable architecture, which is made up of many processing tiles, each containing local memories and processor cores, which can be altered to match the different applications.

One work closely related to ours is that of a configurable cache architecture whose memory hierarchy can be configured for energy and performance trade-offs, proposed by Balasubramonian et al. [2000]. The associativity, size, and latency of their cache can be dynamically configured based on different applications or the same application at different phases. Their work targets general-purpose microprocessors that may require different cache hierarchy architectures. Another efforts closely related to ours are way shutdown cache methods, proposed independently by both Albonesi [1999] and by the designers of the Motorola M*CORE processor [Malik et al. 2000]. In those approaches, a designer would initially profile a program to determine how many ways could be shut down without causing too much performance degradation. Albonesi also discusses dynamic way shutdown and activation for different regions of a program. We showed that our way concatenation approach is superior to way shutdown in reducing dynamic power [Zhang et al. 2003a].

Our way concatenate method is complementary to phased lookup, way predictive, pseudo-set-associative, and filter caching methods. Unlike those methods, ours does not result in multicycle cache accesses during a hit, but those methods could be combined with ours to reduce the number of misses and hence off-chip memory accesses further. The way shutdown of our method also reduces static power, which the other methods above do not.

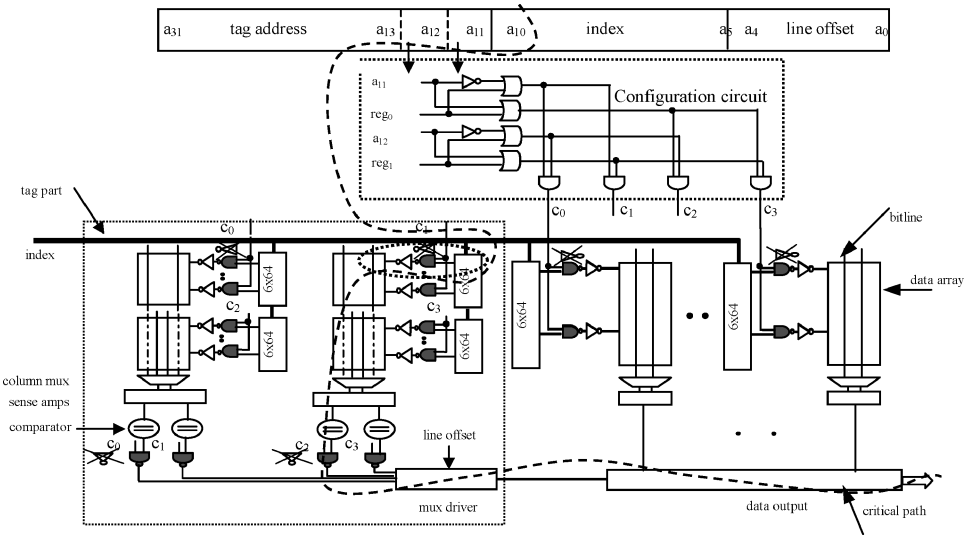

Fig. 3. A way-concatenatable four-way set-associative cache architecture with the critical path shown. We will examine the portion indicated by a dashed circle in more detail in Figure 5.

Compared with memory hierarchy configurable cache, our configurable cache can have some ways shut down and tuned to fit the size of the cache to the specific application. Compared with way shutdown caches, our way concatenation can have different ways given a fixed size of cache, which we will show to be superior in reducing dynamic power.

Compared with cache line size configurable only cache architectures [Inoue et al. 2000; MIPS 2002; Witchel and Asannovic 2001; Veidenbaum et al. 1999], our cache line size is configured under varied cache sizes and associativities.

We have also introduced on-chip hardware implementing an efficient cache tuning heuristic that can automatically, transparently, and dynamically tune the cache to an executing program [Zhang et al. 2004]. That heuristic seeks not only to reduce the number of configurations that must be examined, but also traverses the search space in a way that completely avoids costly cache flushes.

4. WAY CONCATENATION

4.1 Architecture

Because every program has different cache associativity needs, we sought to develop a cache architecture whose associativity could be configured as one, two, or four ways, while still utilizing the full capacity of the cache. Our main idea is to allow ways to be concatenated. The hardware required to support concatenation turned out to be rather simple.

Our way concatenatable cache is shown in Figure 3. *reg0* and *reg1* are two single-bit registers that can be set to configure the cache as four, two, or one-way set-associative. Those two bits are combined with address bits *a11* and *a12* in

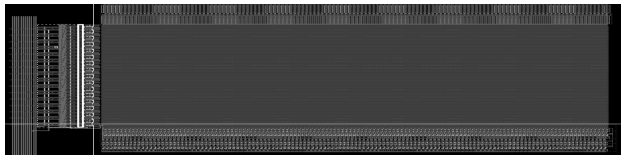

Fig. 4. Layout of one way of cache data.

a configuration circuit to generate four signals *c0*, *c1*, *c2*, *c3*, which are in turn used to control the configuration of the four ways.

When *reg0* = 1 and *reg1* = 1, the cache acts as a four-way set-associative cache. In particular, *c0*, *c1*, *c2*, and *c3* will all be 1, and hence all tag and data ways will be active.

When *reg0* = 0 and *reg1* = 0, the cache acts as a one-way cache (where that one way is four times bigger than the four-way case). Address bits a_{11} and a_{12} will be decoded in the configuration circuit such that exactly one of *c0*, *c1*, *c2*, or *c3* will be 1 for a given address. Thus, only one of the tag arrays and one of the data arrays will be active for a given address. Likewise, only one of the tag comparators will be active.

When *reg0* = 0 and *reg1* = 1, or *reg0* = 1 and *reg1* = 0, then the cache acts as a two-way cache. Exactly two of *c0*, *c1*, *c2*, and *c3* will be 1 for a given address, thus activating two tag and data arrays, and two tag comparators.

Note that we are essentially using six index bits for a four-way cache, seven index bits for a two-way cache, and eight index bits for a one-way cache. Also, note that the total cache capacity does not change when configuring the cache for four, two or one way.

4.2 Cache Layout

While we initially used the CACTI model to determine the impact of the extra circuitry on cache access and energy, we eventually created an actual layout to determine the impact as accurately as possible. Figure 4 shows our layout of the data part of one way of the cache. We used Cadence layout tools [CADENCE 2002] and we extracted the circuit from the layout. The technology we used was TSMC 0.18, the most advanced modern CMOS technology available to universities through the MOSIS program [MOSIS 2002]. The dimensions of our SRAM cell were 2.4 μm $\times$ 4.8 μm, using conventional six-transistor SRAM cells. We used Cadence's Spectra to simulate the netlist of the extracted circuits. We measured the access time and energy consumption of the cache from the outputs of the simulation. We measured the energy of the various parts of a conventional four-way set-associative cache during a cache access, and compared that energy with our configurable way-concatenatable cache configured for four, two, and one-way, using the cache layout. The access energies and savings of our configurable cache are shown in Table II. These energies include dynamic power only, not static. *Cnv* stands for conventional cache, *Cfg* stands for configurable cache. The numbers after *Cnv* and *Cfg* are the size and associativity of the cache, for example, *8K2W* means an 8 Kbytes, two-way set-associative cache. The energy savings of the configured two-way

Table II. Dynamic Access Energy of a Configurable Cache Compared with Conventional Four-Way Set-Associative Cache

	Cnv	Cfg					
	8K4W	8K4W	8K2W	8K1W	4K2W	4K1W	2K1W
Energy (pJ)	827.1	828.1	471.5	293.8	411.9	234.1	219.0
Savings		−0.1%	42.7%	64.0%	50.0%	71.5%	73.4%

and one-way caches come primarily from the fact that fewer sense amplifiers, bit lines, and word lines are active per access in those configurations.

4.3 Time and Area Overhead

Perhaps the most pressing question regarding a way-concatenatable cache is how much the configurability of such a cache increases access time compared to a conventional four-way cache. This is especially important because the cache access time is often on the critical path for a microprocessor, and thus increased cache access time may slowdown the system clock. However, note that the configuration circuit in Figure 3 is not on the cache access critical path, since the circuit executes concurrently with the index decoding. Based on our layout, we can set the sizes of the transistors in the configure circuit such that the speed of the configure circuit is faster than that of the decoder. Such resizing is reasonable because we only have four OR and four AND gates in the configure circuit. From our cache layout, the configure circuit area is negligible.

However, we have changed two inverters on the critical path into NAND gates. NAND gates are slightly slower than inverters. One might think this replacement of inverters by NAND gates would increase the cache access time, but in fact access time need not be increased. In the following, we will analyze how to select the transistor size of the NAND gate to make the access time of the cache as fast as before.

Normally, the cache critical path is on the tag side of a set-associative cache. From Figure 3, we can see the cache critical path includes a tag decoder, a word line, a bit line (including the mux), a sense amplifier, a comparator, a mux driver that selects the output from four ways, and an output driver. We measured the critical path delay to be 1.28 ns.

Of the two inverters we are going to change to NAND gates on the critical path, one is the inverter after the decoder, and another is the inverter after the comparator. Let us consider in more detail the inverter after the decoder. In Figure 5, we show part of the critical circuit, which we will refer to as the tag decoder circuit. The tag decoder circuit includes a 6 to 64 decoder, the decoder inverter that we are going to change into a NAND gate, and the word line driver (inverter). The figure includes two transistors, P2 and N2, to form a NAND gate from the original inverter, which was composed from transistors P1 and N1. The number beside each transistor in the figure is the transistor's size; for example, P1's 3.75/0.2 means the width and the length of the transistor are 3.75 microns and 0.2 microns, respectively, in our 0.18 micron technology. We label four signals in the figure: signal (1) is the address, signal (2) is the

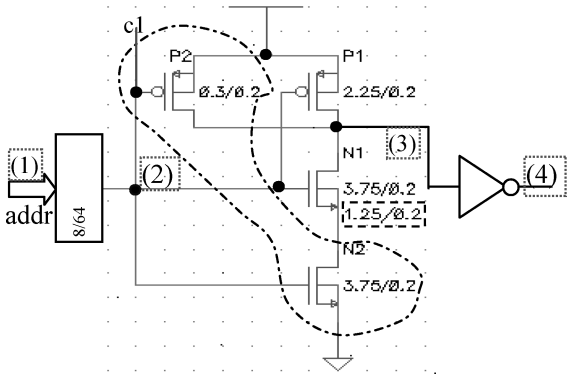

Fig. 5. Structure of the tag decoder circuit circled in Figure 3. *addr* is the address, and *c1* is the output of the configure circuit. Note that the circuit is flipped horizontally compared to Figure 3.

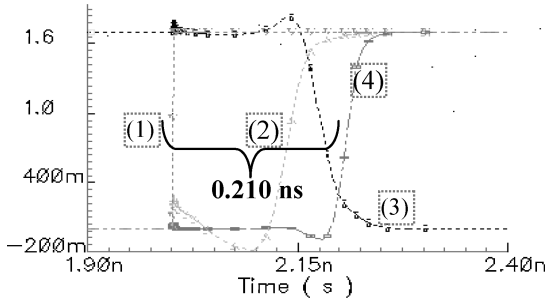

Fig. 6. Transient time response of the tag decoder circuit.

decoder output, signal (3) is the inverter/NAND gate output, and signal (4) is the word line driver output.

Our design goal is to ensure that signal (4) will not be prolonged after we change the inverter to a NAND gate. Figure 6 shows the transient time responses of the four signals *before* the inverter has been changed to a NAND gate. We can see the delay from the address (1) to the word line driver output (4) is 0.210 ns. Case 1 of Figure 7 shows this tag decoder circuit delay of 0.210 ns in the context of the complete critical path delay of 1.28ns. We see that the tag decoder circuit accounts for less than 20% of the total delay.

Replacing the inverter in the tag decode circuit by a NAND gate lengthens the (1) to (4) delay from 0.210 ns to 0.241 ns, as illustrated by Case 3 in Figure 7. The lengthening of 0.031 ns represents a critical path lengthening of 2.4%. Because we replace two such inverters on the critical path, the total lengthening would be 4.8%.

By resizing the NAND gate transistors to three times their original size in the inverter, we can compensate for the lengthened delay. The new NAND gate results in a (1) to (4) delay of 0.210 ns again, just like when using the original inverter, yielding a total delay the same as the original cache, as illustrated by Case 4 in Figure 7.

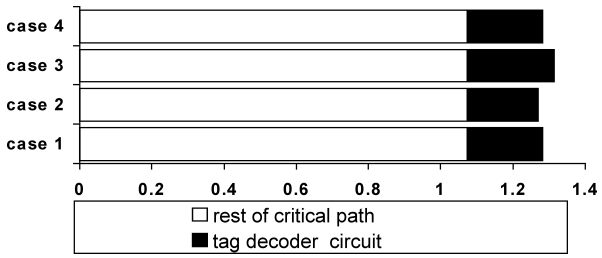

Fig. 7. The cache access time under four cases Case 1: original circuit (total delay is 1.28 ns); case 2: transistor N1 is three times as large of the original to the benefit of charging transistor; case 3: change the inverter to NAND gate without changing the size of the transistor; case 4, change the inverter to NAND gate with three times size of the original transistor.

One might ask why we did not increase the original inverter's transistors to three times their original size, to achieve a shorter original critical path. The reason is because the original inverter's contribution to overall delay was quite small, and only became significant when changed to a NAND gate. Increasing the original inverter's transistors to three times their original size would decrease the (1) to (4) delay by only 0.013 ns (from the original 0.210 ns down to 0.197 ns), representing a mere 1% change in critical path delay (2% when considering the two inverters on the path), as illustrated by Case 2 in Figure 7.

In short, we can resize transistors to ensure that way concatenation does *not* incur performance overhead. The cache's physical layout is such that resizing is indeed possible, and the size overhead by such resizing is negligible compared to the size of the cache.

4.4 Experiments

To determine the benefits of our configurable cache with respect to reducing dynamic and static power consumption and hence energy, we simulated a variety of benchmarks for a variety of cache configurations by using SimpleScalar. The benchmarks included programs from Motorola's Powerstone suite [Malik et al. 2000] (*padpcm, crc, auto2, bcnt, bilv, binary, blit, brev, g3fax, fir, pjpeg, ucbqsort, v42*), MediaBench [Lee et al. 1997] (*adpcm, epic, jpeg, mpeg2, pegwit, g721*) and some programs from Spec 2000 [SPECBENCH 2002] (*art*, *mcf, parser, vpr)*. We included only a subset of benchmarks from each suite due to time constraints, as simulations are very time consuming; we report data for every benchmark that we simulated. We used the sample test vectors that came with each benchmark as program stimuli.

4.4.1 *Results.* Figure 8 shows instruction and data cache miss rates for the benchmarks for three configurations of our way-concatenatable cache: 8 Kbytes with 4-way associativity, 8 Kbytes with 2-way associativity, and 8 Kbytes with 1-way associativity (direct mapped).

We see that the miss rates in the figures support our earlier discussions in which we pointed out that most benchmarks do fine with a direct-mapped cache. However, we see that some benchmarks, like *jpeg* and *parser*, do much better

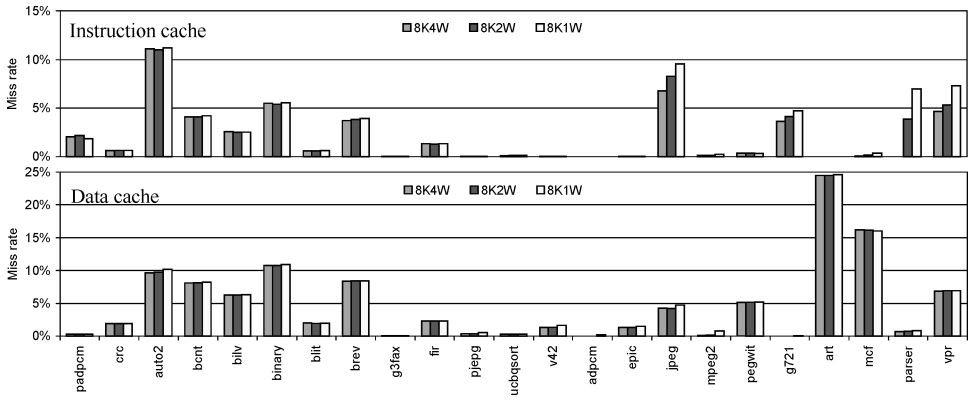

Fig. 8. Miss rates of an 8 Kbytes instruction and data caches when configured as four ways, two ways, and direct mapped.

with higher-associativity caches. *jpeg*'s miss rate is only 6.5% with a four-way instruction cache, but 9.5% with a one-way cache. *parser*'s miss rate is nearly 0% with a four-way instruction cache, but is 7% with a one-way cache.

We computed energy data for the benchmarks, using the method described by Eq. (1), with results summarized using the first three bars in the figure. We compared our 8 Kbytes configurable cache having way concatenation and a 32-byte line size (*cfg8Kwc32B*), with two conventional caches: an 8-Kbytes conventional cache having four-way set associativity and a 32-byte line size (*cnv8K4W32B*), and an 8-Kbytes conventional cache having one-way (direct mapped) and a 32-byte line size (*cnv8K1W32B*). (The other two bars shown for each example will be described in upcoming sections). We normalized all energies to the conventional four-way cache. The energy we display for our configurable cache was determined by simulating all possible configurable cache configurations for a given benchmark, and selecting the lowest energy configuration.

For most benchmarks, the configuration yielding minimal energy has both instruction cache and data cache configured for one way. However, for some benchmarks, such as *jpeg*, *g*721, *parser* and *vpr*, one-way configurations result in more overall energy due to high miss rates. In those cases, higher associativities are preferred. *jpeg*, for example, uses minimal energy with a four-way instruction cache and a two-way data cache. *parser* does best with a four-way instruction cache and a one-way data cache. *Mpeg2* does best with a one-way instruction cache but a two-way data cache.

We should point out a design choice we made for the conventional direct-mapped cache that results in our configurable cache configured as direct-mapped achieving lower energy than a conventional direct-mapped cache. We had to choose between a direct-mapped cache having an 8 Kbytes word array internally, having two 4 Kbytes word arrays, or having four 2 Kbytes word arrays. These latter two cases, known as cache subbanking [Ghose and Kamble 1999] can reduce the power per access, by accessing smaller arrays. The cost of subbanking is the multiplexing logic, which adds delay. We chose to compare with a direct-mapped caching having a single 8 Kbytes word array, giving the

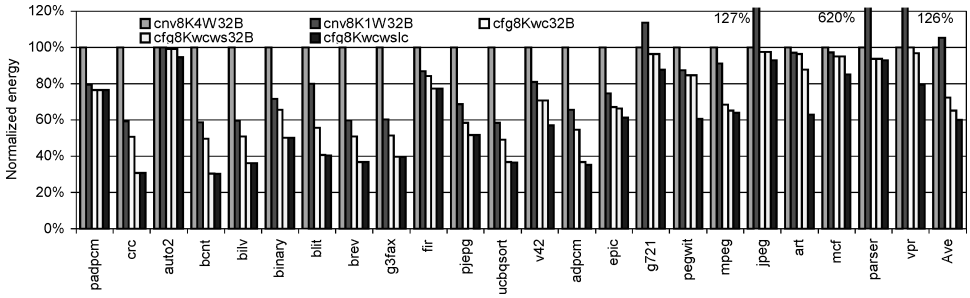

Fig. 9. Normalized energy dissipation when way concatenation, way shut down, and cache line size concatenations are all implemented. cnv stands for conventional, cfg stands for configurable, wc: way concatenation, ws: way shut down; lc: line concatenation.

fastest access time, which is one of the reasons that microprocessor designers choose a direct-mapped cache.

The conclusions from our results would not change substantially if we had chosen to compare to a subbanked direct-mapped cache. The conventional direct-mapped cache energy might be even with or slightly better than our configurable cache for some examples, but the average savings overall would be quite similar.

4.4.2 *Main Observations.* The first observation we make from this data is that a way-concatenatable configurable cache has an average energy savings of 28% compared to a conventional four-way set-associative cache, ranging from 3% savings for *jpeg* to 51% for *ucbqsort*. The energy savings is 33% compared to a conventional direct-mapped cache, but perhaps more importantly, the savings for some examples can be quite large—620% for parser, which results from the highly undesirable performance degradation due to the high miss rate of a direct-mapped cache.

4.4.3 *Impact of k_miss_energy and k_static Ratios.* In our energy calculation equation (Eq. (1)), we included the memory access energy and the processor stall energy. Figure 9 showed results for $k\_miss\_energy = 50$ and $k\_static = 30\%$. We also generated results when increasing k_miss_energy to 200 (off-chip memory accesses are even more expensive), and when modifying k_static to 50% (static energy consumption is even more important). We described those results in [Zhang et al. 2003a], achieving 31% savings compared to a conventional four-way set-associative cache. Compared to a direct-mapped cache, savings increased to 48%, due to the even greater penalty caused by a higher miss rate.

5. WAY SHUTDOWN

We have thus far focused on the impact of cache associativity on energy consumption. We also know that cache size plays an important role in energy dissipation, especially when static energy, which is proportional to the cache size and execution time, begins to account for more of the total cache energy consumption.

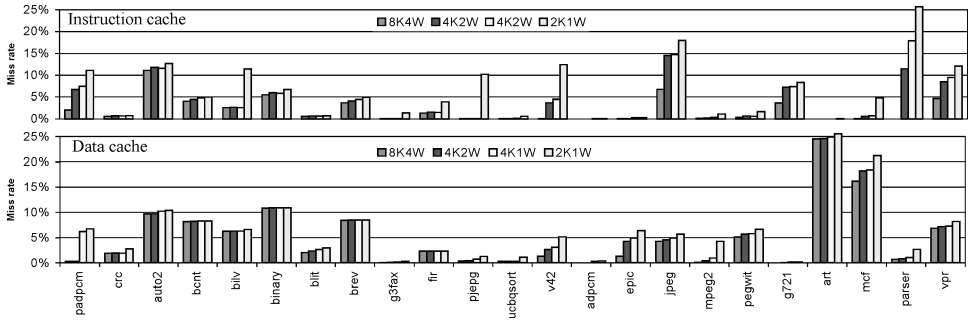

Fig. 10. Miss rate when two ways or three ways of the original four-way 8 Kbytes cache are shut down.

As CMOS technology continues to scale down, transistors with lower threshold voltage are becoming common. Low threshold voltage transistors enable a lower supply voltage, resulting in great reductions in dynamic power, since dynamic power is proportional to the *square* of voltage. However, lower threshold voltage transistors also result in more subthreshold leakage current through the transistors, resulting in increased static power consumption. Thus, static power is becoming a greater concern [Agarwal et al. 2002]. Some researchers are thus working on leakage power reductions, such as DRG-Cache [Agarwal et al. 2002].

Figure 10 shows the miss rate when cache ways are shut down. We see significant increases in the miss rate for many examples. For example, *v42* has a nearly 0% instruction cache miss rate with all ways on, but has 4% and 12% miss rates when two or three ways are shut down. In contrast, no such miss rate increase occurs when ways were concatenated in Figure 8. We see that shutting down ways is far more likely to increase the miss rate than concatenating ways—which intuitively makes sense since way shutdown decreases the cache size while way concatenate does not. However, we still can see that although way shutdown increases the miss rate for some benchmarks, for other benchmarks, way shutdown has negligible impact, such as for *fir, brev,* and *binary* on data cache. Such negligible impact means that the benchmarks do not need the full capacity (8 Kbytes) of the cache. To save static energy, we want to shut down the unneeded capacity. We choose to use way shutdown for this purpose. Thus, we extend our way-concatenatable cache to include way shutdown also.

5.1 Architecture

Albonesi [1999] originally proposed way shutdown to reduce dynamic power, using an AND gate to shut down cache ways. Since we instead use way concatenation to reduce dynamic power and we want to use way shutdown to reduce static power, we use instead the shutdown method by Powell et al. [2001], involving a circuit level technique called gated-Vdd, shown in Figure 11. When the gated-Vdd transistor is turned off, the stacking effect of the extra transistor reduces the leakage energy dissipation. Because the extra transistor can be shared by an array of SRAM cells, the area increase is thus only about 5%.

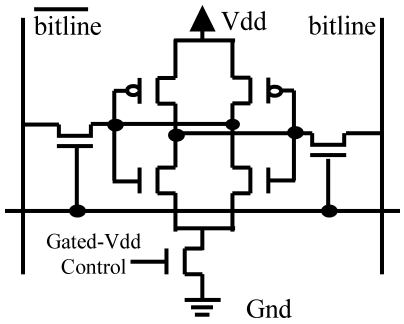

Fig. 11. SRAM cell with an nMOS gated-Vdd control.

However, Powell showed that the performance overhead of the extra transistor is about 8%.

5.2 Experiments

The normalized energy dissipation when both way concatenation and way shut down are implemented is shown in Figure 9 as *cfg8Kwcws32B*. We again determine our configurable cache energy by examining all possible configurations of way concatenation and way shutdown (there are six such configurations). The average savings compared to a conventional four-way cache increase from 28% for a way-concatenate cache to 35% for a way-concatenate way-shutdown cache. Savings compared to a conventional direct mapped were again slightly greater.

Those results were obtained for $k\_miss\_energy = 50$ and for $k\_static = 30\%$. We also obtained results for $k\_miss\_energy = 200$. Again, due to the higher cost of misses, the conventional direct mapped does even worse, with an energy consumption overhead up to 800% for *parser*, for example.

Likewise, we considered the case where $k\_static = 50\%$. For many examples, way shutdown becomes increasingly important since static energy begins to dominate. In most examples, way shutdown alone gained most of the energy savings. However, in some examples, such as *v*42 and *padpcm*, the combination of both concatenate and shutdown was necessary—way shutdown alone increased the miss rate too much for these examples and thus did not save energy. Thus, we conclude that we need both way concatenate and way shutdown to effectively reduce static energy, even in deep submicron technologies.

6. CONFIGURABLE LINE SIZE

6.1 Architecture

We also considered cache line size as a configurable cache parameter. Creating a cache with a configurable line size is relatively straightforward. Our approach is shown in Figure 12. The physical line size of the cache is 16 bytes. A counter in the cache controller specifies how many words to read from the off-chip memory. For a conventional cache, this counter contains a fixed number, like four for a 16-byte line size cache, assuming one word is read from off-chip memory at a time. We make the counter writable to achieve configurability. When the line

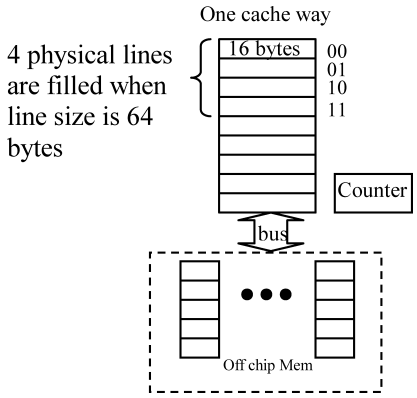

Fig. 12. Architecture of a line size configurable cache.

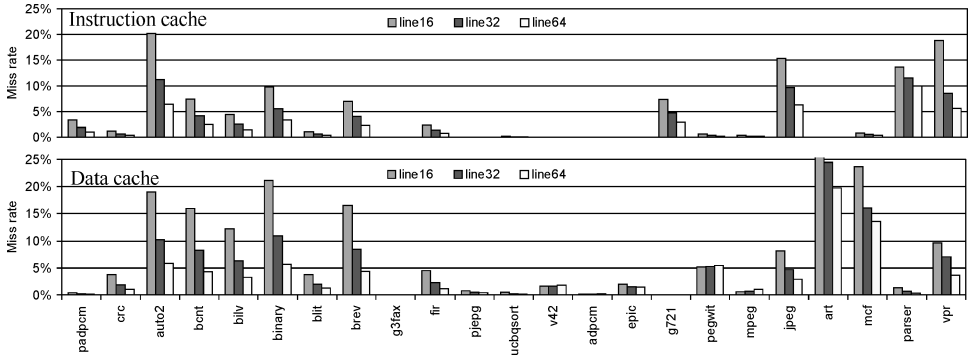

Fig. 13. Miss rates of one-way instruction (top) and data (bottom) caches for 16, 32 and 64 byte line sizes.

size is configured larger than 16 bytes, such as 64 bytes shown as in Figure 12, if there is a miss at physical line 10, then the replace should start from physical line 00.

We assume the use of an interleaved memory organization. Because we configure the cache line size statically, we do not require the off-chip memory to fit for all line size possibilities. When the line size is 16 bytes, the off-chip memory should be organized as 4 banks interleaved, and 8 or 16 banks interleaved for line sizes of 32 bytes and 64 bytes, respectively.

6.2 Experiments

Figure 13 shows the miss rates for various line sizes for the benchmarks using direct-mapped instruction and data caches. We see in some benchmarks that a small line size yields a much higher miss rate than a larger line size, in which case a smaller line size will likely result in higher energy. In other benchmarks, the small line size works better, so will likely save energy. In many cases, the line size has little impact, in which case a smaller line size will likely save energy. The difference in miss rate between line sizes is quite high—more than 15% in some cases. In terms of miss rates, 15% is huge.

In Zhang et al. [2003b], we showed the energy benefits of a configurable line size for a four-way set-associative instruction cache, for line sizes of 16, 32, and 64-bytes. We showed that for most benchmarks, a line size of 64 bytes yielded the least energy. However, several benchmarks, such as *v42*, *g721*, *pegwit* and *jpeg,* yielded the least energy at a line size of 16 bytes. The energy differences were surprisingly significant—over 20% in many cases. A line size of 32 did not yield significant improvements over the other two line sizes in any particular case, but did work well on average and never performed very poorly—thus explaining its popularity in Table I. In contrast, 16 bytes was best for some examples and 64 bytes for others, but each performed very poorly in some examples.

We found that selecting the best line size is even more important for data cache, as the energy differences between line sizes are even greater—up to 50%. The reason is because spatial and temporal locality varies more greatly for data access than instruction access. We also found that the line size becomes even more critical for direct-mapped caches. The differences in miss rates among line sizes are even more pronounced than before. We found a nearly 60% energy difference in some cases of the data cache, just by varying the line size.

The normalized energy dissipation of a cache, whose cache associativity, sizes, and line size can be configured, is shown in Figure 9 (*cfg8Kwcwslc*). We can see the energy savings of the configurable cache is now up to an average of 40% compared with a conventional four-way set-associative cache.

6.3 Overhead of Configurability

The overhead of cache line size configuration is negligible. From Figure 12, we can see that we need to make the counter configurable. This counter will not reside in the critical path. A 16-byte line size should have no overhead. A 64-byte line size could have a few cycles overhead between 16-byte chunks, but these cycles (if any) should be quite small compared to the cycles to read and write the bytes themselves. The size of the counter is also negligible, though making the counter accessible for writes through memory-mapped I/O will require some additional wires and logic.

7. DISCUSSION

Not all three cache parameters are effective for all benchmarks. For example, for benchmarks *crc, bcnt, bilv, binary, bilt, ucbqsort, fir*, and *brev*, way concatenation and way shut down combined together at line size 32 bytes has already reduced the energy dissipation to the extent that line size configuration to either 16 or 64 bytes will not reduce the energy dissipation any further. This means that, when associativity and cache size can be chosen to reduce the energy dissipation, a line size of s32 byte can also be the best line size, which is different from our observations from [Ye et al. 1998], where only cache line size is configurable. For benchmarks *v42, g721, pegwit, jpeg,* and *mcf*, way concatenation and line size concatenation achieve most of the energy dissipation, because when way shut down is incorporated with way concatenation, we cannot see any further energy is reduced. For benchmarks, *vpr, mcf, art, g721*, and *pegwit*, line size concatenation contributes the most of the energy reduction.

Table III. The Best Configuration in Terms of Energy Dissipation of Instruction and Data Cache for All Benchmarks

	Best Configuration			Best Configuration	
Ben.	ICACHE	DCACHE	Ben.	ICACHE	DCACHE
padpcm	8K1W32B	8K1W32B	pjepg	4K1W32B	4K2W64B
crc	2K1W32B	4K1W64B	ucbqsort	4K1W16B	4K1W64B
auto	8K2W16B	4K1W32B	v42	8K1W16B	8K2W16B
bcnt	2K1W32B	2K1W64B	adpcm	2K1W16B	4K1W16B
bilv	4K1W32B	2K1W32B	epic	2K1W64B	8K1W16B
binary	2K1W32B	2K1W32B	g721	8K4W16B	2K1W16B
blit	2K1W16B	8K2W32B	pegwit	4K1W16B	4K1W16B
brev	4K1W32B	2K1W32B	mpeg2	4K1W32B	8K2W16B
g3fax	4K1W32B	4K1W16B	art	2K1W32B	2K1W16B
fir	4K1W32B	2K1W32B	parser	8K4W16B	8K2W64B
jpeg	8K4W32B	4K2W32B	mcf	8K4W16B	8K1W16B
vpr	8K4W32B	2K1W16B			

Clearly, though, we need all three parameters to account for the spectrum of applications.

The best configurations of all the benchmarks are shown in Table III. From the table, we can see that any value of the three cache parameters, cache associativity, size, and line size, is possible to be the best for some benchmarks.

Note in Figure 9 that our configurable cache is not only best on average, but is also best for every example. This phenomenon is easily explained by the fact that conventional caches are designed as a compromise.

A configurable cache could be configured by a designer, or possibly dynamically. In the former scenario, an embedded system designer would have a fixed program that would run on the microprocessor platform having the configurable cache. Based on simulations or actual executions on the platform, the designer would determine the best configuration for that program. The designer would then modify the boot or reset part of the program to set the cache's configuration registers to the chosen configuration. We have also developed a method for dynamically configuring the cache [Zhang et al. 2004].

One limitation of our work is that our direct-mapped configuration is not as fast as a conventional direct-mapped cache could be. Thus, system clock frequency using our configurable cache may be slightly slower than a direct-mapped cache.

One area of future investigation involves use of our configurable cache for desktop applications. Another area involves use of multiple levels of configurable cache.

8. CONCLUSIONS

We have introduced novel configurable cache architecture for embedded computing platforms. By incorporating simple configure circuits, we can configure the associativity, size, and line size of an embedded system's cache architecture. We obtained average energy savings of over 40% compared with conventional four-way set-associative and conventional direct-mapped caches, with savings as high as 70% compared to a four-way cache, and as high as 90% compared

to a direct-mapped cache. Since caches may consume half of a microprocessor system's power, such savings can significantly reduce overall system power.

REFERENCES

AGARWAL, A., LI, H., AND ROY, K. 2002. DRG-Cache. A data retention gated-ground cache for low power. In *Design Automation Conference*.

ALBONESI, D. H. 1999. Selective cache ways: On-demand cache resource allocation. In the *32nd Annual ACM/IEEE International Symposium on Microarchitecture*.

BALASUBRAMONIAN, R., ALBONESI, D., BUYUKTOSUNOGLU, A., AND DWARKADAS, S. 2000. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In the *33rd International Symposium on Microarchitecture*.

BATSON, B. AND VIJAYKUMAR, T. N. 2001. Reactive-associative caches. In *International Conference on Parallel Architectures and Compilation Techniques*.

BURGER, D. AND AUSTIN, T. M. 1997. *The SimpleScalar Tool Set, Version 2.0*. University of Wisconsin-Madison Computer Sciences, Department. Technical Report #1342.

CADENCE. 2002. http://www.cadence.com.

CALDER, B., GRUNWALL, D., AND EMER, J. 1996. Predictive sequential associative cache. In *International Symposium on High Performance Computer Architecture*.

EDMONDSON, J. H. AND RUBINFIELD, P. I. 1995. Internal organization of the Alpha 21164 a 300-MHz 64-bit quad-issue CMOS RISC microprocessor. *Digital Technical Journal 7*, 1, 119—135.

DROPSHO, S., BUYUKTOSUNOGLU, A., BALASUBRAMONIAN, R., ALBONESI, D. H., DWARKADAS, S., SEMERARO, G., MAGKLIS, G., AND SCOTT, M. L. 2002. Integrating adaptive on-chip storage structures for reduced dynamic power. In the *11th International Conference on Parallel Architectures and Compilation Techniques*.

FLAUTNER, K., ET AL. 2002. Drowsy caches: Simple techniques for reducing leakage power. In the *35th Annual ACM/IEEE International Symposium on Microarchitecture*.

GHOSE, K. AND KAMBLE, M. B. 1999. Reducing power in superscaler processor caches using subbanking, multiple line buffers and bit-line segmentation. In *International Symposium on Low Power Electronics and Design*.

HANSON, H. 2000. Static energy reduction for microprocessor caches. In the *International Conference on Computer Design*.

HASEGAWA, A., KAWASAKI, I., YAMADA, K., YOSHIOKA, S., KAWASAKI, S., AND BISWAS, P. 1995. SH3: High code density, low power. *IEEE Micro 15*, 6, 11–19.

HENNESSY, J. L., AND PATTERSON, D. A. 1996. *Computer Architecture Quantitative Approach*, 2nd ed. Morgan-Kaufmann, Menlo Park, CA.

INTEL. 2002. http://www.developer.intel.com/design/strong/.

INOUE, K., ISHIHARA, T., AND MURAKAMI, K. 1999. Way-predictive set-sssociative cache for high performance and low energy consumption. In *International Symposium on Low Power Electronic Design*.

INOUE, K. AND KAI, K. 2000. A high-performance/low-power on-chip memory-path architecture with variable cache-line size. *IEICE Trans. Electron. E83-CV, 11* (Nov.).

KAXIRAS, S., HU, Z., AND MARTONOSI, M. 2001. Cache decay: Exploiting generational behavior to reduce cache leakage power. In the *28th Annual International Symposium on Computer Architecture*.

KIM, H., SOMANI, A. K., AND TYAGI, A. 2001. A reconfigurable multi-function computing cache architecture. *IEEE Transactions on VLSI Systems 9*, 4 (Aug.), 509–523.

KIN, J., GUPTA, M., AND MANGIONE-SMITH, W. 1997. The filter cache: An energy efficient memory structure. In *International Symposium on Microarchitecture*. 184–193.

LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. 1997. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitecture*.

MAI, K., PAASKE, T., JAYASENA, N., HO, R., DALLY, W. J., AND HOROWITZ, M. 2000. Smart memories: A modular reconfigurable architecture. *ACM SIGARCH Computer Architecture News 28*, 2.

MALIK, A., MOYER, B., AND CERMAK, D. 2000. A low power unified cache architecture providing power and performance flexibility. In *International Symposium on Low Power Electronics and Design*.

MIPS. 2002. http://www.mips.com.

MOSIS. 2002. http://www.mosis.org.

POWELL, M., YANG, S. H., FALSAFI, B., ROY, K., AND VIJAYKUMAR, T. N. 2000. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In the *ACM/IEEE International Symposium on Low Power Electronics and Design*.

POWELL, M. D., AGARWAL, A., VIJAYKUMAR, T. N., FALSAFI, B., AND ROY, K. 2001. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In the *34th International Symposium on Microarchitecture*.

RANGANATHAN, P., ADVE, S., AND JOUPPI, N. P. 2000. Reconfigurable caches and their application to media processing. In the *27th Annual International Symposium on Computer Architecture*.

REINMAN, G. AND JOUPPI, N. P. 1999. *CACTI2.0: An Integrated Cache Timing and Power Model*. COMPAQ Western Research Lab.

SEGARS, S. 2001. Low power desin techniques for microprocessors. In *IEEE International Solid-State Circuits Conference Tutorial*.

SEMICONDUCTOR INDUSTRY ASSOCIATION. 1999. *International Technology Roadmap for Semiconductors: 1999 edition.* International SEMATECH, Austin, TX.

SMITH, M. J. S. 1997. *Application-Specific Integrated Circuits*. Addison-Wesley Longman, Reading, MA.

SPECBENCH. 2002. http://www.specbench.org/osg/cpu2000.

TADAS, S. AND CHAKRABARTI, C. 2002. Architectural approaches to reduce leakage energy in caches. In *International Symposium on Circuits and System*.

VEIDENBAUM, A., TANG, W., GUPTA, R., NICOLAU, A., AND JI, X. 1999. Adapting cache line size to application behavior. In *International Conference on Supercomputing.*

WITCHEL, E. AND ASANNOVIC, K. 2001. The span cache: Software controlled tag checks and cache cine Size. In the *28th Annual International Symposium on Computer Architecture*.

YANG, S., POWELL, M. D., FALSAFI, B., ROY, K., AND VIJAYKUMAR, T. N. 2001. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches. In the *7th International Symposium on High-Performance Computer Architecture*.

YE, Y. BORKER, S., ET AL. 1998. A new technique for standby leakage reduction in high-performance circuits. In *International Symposium on VLSI circuits*.

ZHANG, C., VAHID, F., AND NAJJAR, W. 2003a. A highly configurable cache architecture for embedded systems. In the *30th ACM/IEEE International Symposium on Computer Architecture*.

ZHANG, C., VAHID, F., AND NAJJAR, W. 2003b. Energy benefits of a configurable line size cache for embedded systems. In *International Symposium on VLSI Design*.

ZHANG, C., VAHID, F., AND LYSECKY, R. 2004. A self-tuning cache architecture for embedded systems. In Special issue on Dynamically Adaptable Embedded System. *ACM Transactions on Embedded Computing Systems 3*, 2 (May), 1–19.

ZHOU, H., TOBUREN, M. C., ROTENBERG, E., AND CONT, T. M. 2001. Adaptive mode-control: A static-power-efficient cache design. In the *10th International Conference on Parallel Architectures and Compilation Techniques*.

Received March 2003; revised March 2004; accepted January 2005