A Highly Configurable Cache Architecture for Embedded Systems

Chuanjun Zhang
Department of Electrical Engineering
University of California, Riverside
czhang@ee.ucr.edu

Frank Vahid* and Walid Najjar
Department of Computer Science and Engineering
University of California, Riverside
{vahid/najjar}@cs.ucr.edu
http://www.cs.ucr.edu/{~vahid/~najjar}
*Also with the Center for Embedded Computer Systems, UC Irvine

Abstract
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 largely determined by the behavior of the application using that cache. Desktop systems have to accommodate a very wide range of applications and therefore the manufacturer usually sets the cache architecture as a 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 can be configured by software to be direct-mapped, two-way, or four-way set associative, using a technique we call way concatenation, having very little size or performance overhead. We show that the proposed cache architecture reduces energy caused by dynamic power compared to a way-shutdown cache. Furthermore, we extend the cache architecture to also support a way shutdown method designed to reduce the energy from static power that is increasing in importance in newer CMOS technologies. Our study of 23 programs drawn from Powerstone, MediaBench and Spec2000 show that tuning the cache's configuration saves energy for every program compared to conventional four-way set-associative as well as direct mapped caches, with average savings of 40% compared to a four-way conventional cache.

Keywords
Cache, configurable, architecture tuning, low power, low energy, embedded systems, microprocessor.

1. Introduction
Designers of embedded microprocessor platforms have to compromise between performance, cost, and energy usage. Caches may consume up to 50% of a microprocessor's energy [18][25]. A direct mapped cache is more energy efficient per access, consuming only about 30% the energy of a same-sized four-way set associative cache [23]. This reduction occurs because a direct mapped cache accesses only one tag and data array per access, while a four-way cache accesses four tag and data arrays per access. A direct mapped cache can also have a shorter access time in part because multiple data arrays need not be multiplexed. While a direct-mapped cache's hit rate may be acceptable for many applications, for some applications a direct-mapped cache exhibits a very poor hit rate and hence suffers from poor performance and energy consumption. Adding set associativity increases the range of applications for which a cache has a good hit rate, but for many applications, the additional associativity is unnecessary and thus results in wasted energy and longer access time.

Deciding on a cache's total size involves a similar dilemma. A small cache is more energy efficient and has a good hit rate for a majority of applications, but a larger cache increases the range of applications displaying a good hit rate, at the expense of wasted energy for many applications.

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 applications.

One option in embedded systems for solving the cache design dilemma and tuning the cache architecture to a small set of applications is for a manufacturer to produce multiple versions of the same processor, each with a cache architecture tuned to a specific set of applications. Another is to provide a synthesizable processor core rather than a physical chip, so that the cache architecture can be modified for the intended application. Both options unduly increase the unit cost. The second option also suffers from a longer time to market [24].

The diversity among cache architectures found in modern embedded microprocessors, summarized in Table

	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	16K	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

Table 1**:** Instruction and data cache sizes, associativities, and line sizes of popular embedded microprocessors. *As* means associativity. *DM* means direct-mapped. *Size* is total cache size in bytes. *U* means instruction and data caches are unified. *Line* is line size in bytes. Sources: Microprocessor Report, and data sheets of various microprocessors.

1, illustrates that the dilemma of deciding on the best cache architecture for mass production has yet to be solved.

We introduce a novel configurable cache architecture that to a large extent reduces the dilemma. By setting a few bits in a configuration register, the cache can be configured in software as either direct mapped or set associative, while still utilizing the full cache capacity. We achieve such configurability using a new technique we call *way concatenation*. Furthermore, using previously known techniques, the cache can be configured to shutdown certain regions in order to effectively reduce the cache's size. Way concatenation has no performance overhead. Way shutdown only has a small amount of performance overhead. Both have very small size overhead, compared to a regular four-way set-associative cache, as verified 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 way concatenation and shutdown cache architecture. Section 2 describes our base cache architecture and illustrates the impact of cache associativity on performance and energy, motivating the need for a cache with a configurable number of ways. Section 3 summarizes previous work. Section 4 introduces our way concatenation cache architecture and provides experimental results showing reductions in dynamic power compared to non-configurable caches, using applications drawn from the Powerstone [18], MediaBench [16] and the Spec2000 [12] benchmark suites. Section 5 extends the architecture to include way shutdown and provides experimental results showing reductions in static power. Section 6 describes methods for using a configurable cache. Section 7 provides conclusions and future work.

2. Cache Associativity's Influence on Energy

2.1 Energy

Energy is the product of power and time. There are two main components that result in power dissipation in CMOS circuits, namely static power dissipation due to leakage current and dynamic power dissipation due to logic switching current and the charging and discharging of the load capacitance.

Dynamic energy contributes to most of the total energy dissipation in micrometer-scale technologies, but static energy dissipation will contribute an increasing portion of energy in nanometer-scale technologies [1] [20][24]. We consider both types of energy in our 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 off-chip memory, energy is consumed when the microprocessor stalls while waiting for the instruction and/or data. Off-chip and stall energies have often been overlooked in previous works. Thus, our equation for computing the total energy due to memory accesses is as follows:

Equation 1: $energy\_mem = energy\_dynamic + energy\_static$

where:

$$energy\_dynamic = \underline{cache\_hits} * \underline{energy\_hit} + \underline{cache\_misses} * energy\_miss$$

$$energy\_miss = \underline{energy\_offchip\_access} + \underline{energy\_uP\_stall} + \underline{energy\_cache\_block\_fill}$$

$$energy\_static = \underline{cycles} * \underline{energy\_static\_per\_cycle}$$

The underlined terms are those we obtain through measurements or simulations. We compute *cache_hits* and *cache_misses* by running SimpleScalar [5] simulations for each cache configuration. We compute *energy_hit* of each cache configuration through simulation of circuits extracted from our layout using Cadence [6] (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 choose 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, and found that *energy_miss* ranged from 50 to 200 times bigger than *energy_hit*. Thus, we redefined *energy_miss* as:

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

and we considered the 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:

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

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 [1] 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.

2.2 Base Cache

After examining typical cache configurations of several popular embedded microprocessors, summarized in Table 1, we choose 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 that we will extend to be configurable, and to which we will compare our results.

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 cache decodes the address' index field to simultaneously read out the appropriate tag from each of the four tag arrays, while decoding the index field to simultaneously read out the appropriate data from the four data arrays. The cache feeds the decoded lines through two inverters to strengthen their signals. The read tags and data items pass through sense amplifiers. The cache simultaneously compares the four tags with the address' tag field. If one tag matches, a multiplexor routes the corresponding data to the cache output.

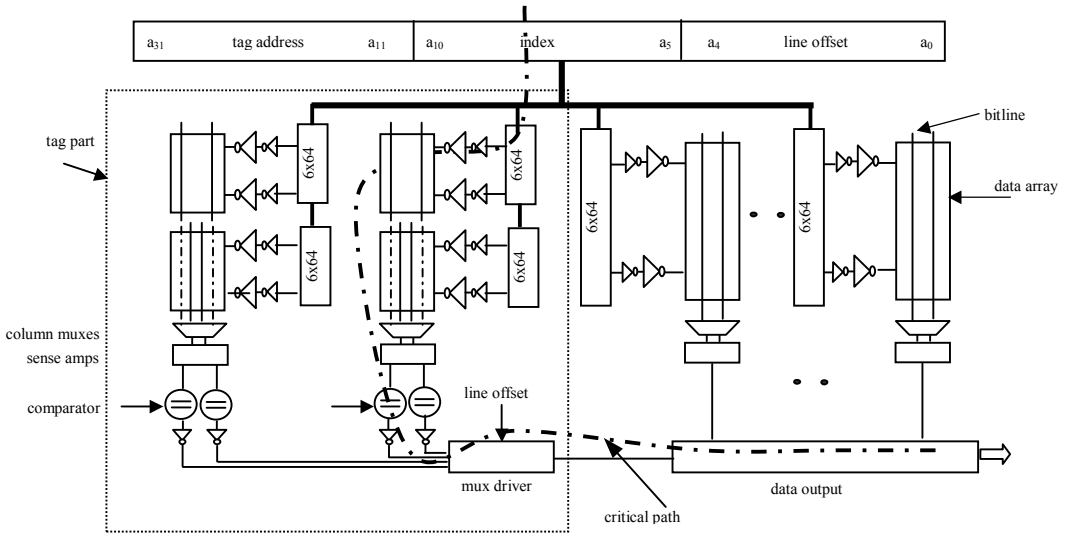

Figure 1: A four way set associative cache architecture with the critical path shown.

2.3 The Impact of Cache Associativity

The associativity greatly impacts system energy. A direct mapped cache uses less power per access than a four way set associative cache, since only one tag and one data array are read during an access, rather than four tag and four data arrays. The power dissipation of a direct mapped cache was shown to be about 30% of a same size four way set associative cache [23]. If a direct-mapped cache has a low miss rate, such a cache can result in low overall energy.

However, sometimes a direct mapped cache has a high miss rate, resulting in higher energy due to longer time as well as high power for accessing the next level memory. Increasing cache associativity can decrease the cache miss rate and hence reduce energy. For example, the average miss rate for the SPEC92 benchmarks is 4.6% for a one-way 8 Kbyte cache, 3.8% for a two-way 8 Kbyte cache and only 2.9% for four-way 8 Kbyte cache [10] (in the remainder of this paper, we will sometimes refer to a direct-mapped cache as having one way). 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 or even hundreds of cycles).

Thus, although accessing a four-way set associative cache requires more power per access, that extra power may be compensated for by the reduction in time and power that would have been caused by misses. Figure 2(a) shows the miss rate for two MediaBench benchmarks, *epic* and *mpeg2*, measured using SimpleScalar [5] and configured with an 8 Kbyte data cache having one, two or four-way set-associativity, with a line size of 32 bytes. Notice that the hit rates for both are better with two ways than with one way, but the additional improvement using four ways is very small.

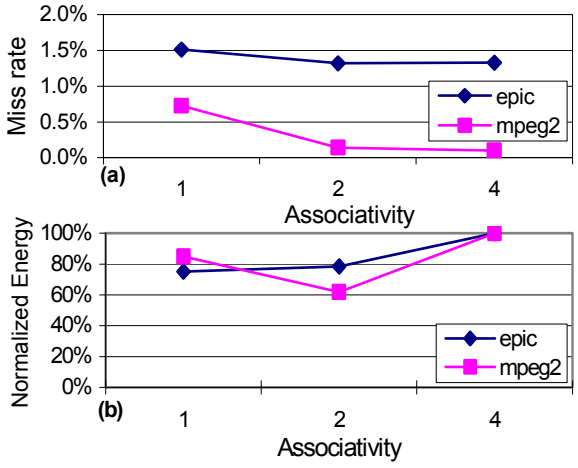

Figure 2: Miss rate (a) and normalized energy (b) of *epic* and *mpeg2* on 8 Kbyte data caches of different associativities.

Figure 2(b) shows overall energy for these two examples, computed using Equation 1, demonstrating that a two-way cache gives lowest energy for *mpeg2*, while a one-way cache is best for *epic*. Notice that the energy differences are quite significant – up to 40%.

Clearly, tuning the associativity to a particular application is extremely important to minimize energy, motivating the need for a cache with configurable associativity.

3. Previous Work

Several cache lookup variations have been proposed by researchers to reduce set-associative cache access energy. Phased-lookup cache [9] 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 [13][21] 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) [4] 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 RAC proceeds without any way-select multiplexor, the cache can achieve the speed of a direct mapped cache but consumes less energy than a conventional set-associative cache. Pseudo Set-Associative Caches [11], such as in [7], are set-associative caches with multiple hit times. The ways can be probed sequentially and consume less energy than that of a conventional set associative cache. Dropsho [8] discussed an accounting cache architecture that is based on the resizable selective ways cache proposed by Albonesi [2]. 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 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 [15] introduces an extremely small (and hence low power) direct mapped cache in front of the regular cache. 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.

Compression methods can reduce cache energy by reducing bit switching during accesses. For example, Yang proposed a scheme for compressing frequently seen values in the data cache [26].

Researchers have recently begun to suggest configurable cache architectures. Ranganathan [22] proposed a configurable cache architecture for a general-purpose microprocessor. When used in media

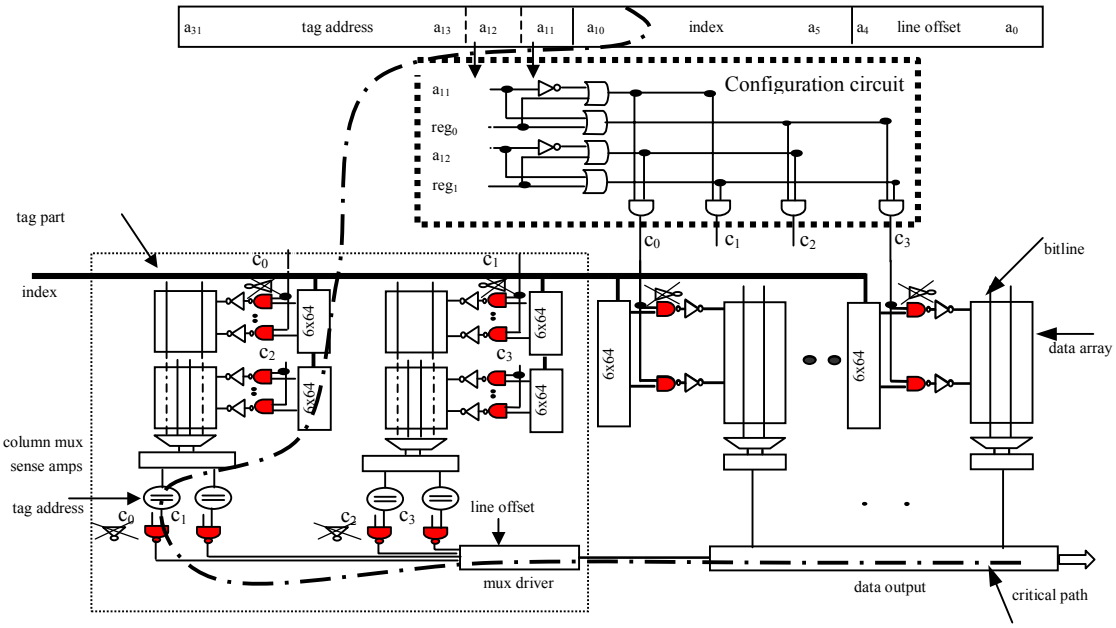

Figure 3: A way-concatenable four-way set associative cache architecture, with the critical path shown.

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 [14] 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 [17] 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.

A CAM (content addressable memory) based highly associative cache architecture [29] puts one set of a cache in a small sub-bank and uses a CAM for the tag lookup of that set, which may have 32 or even 64 ways, saving power at the expense of area.

One work closely related to ours is that on configurable caches whose memory hierarchy can be configured for energy and performance tradeoff, proposed by Balasubramonian [3]. The size of the L1 and L2 cache can be dynamically configured by allocating a part of a fixed size cache to L1 and L2 cache based on different applications or the same application at different phases. Their work targets general-purpose microprocessors that may require different cache hierarchy architectures. Other work closely related to ours are way-shutdown cache methods, proposed independently by both Albonesi [2] and by the designers of the Motorola M*CORE processor [18]. 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.

Our way-concatenate method is complementary to phased lookup, way predictive, pseudo-set associative, and filter caching methods. Unlike those other methods, our method does not result in multi-cycle cache accesses during a hit, but those methods could be combined with ours to reduce energy further. Our method's way shutdown also reduces static power, which those other methods don't. Our method does at this time require an initial profiling step, though in future work we plan to automate the tuning of the cache to a program.

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 a superior method for reducing dynamic power.

4. Way Concatenation for Dynamic Power Reduction

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 this turned out to be rather simple.

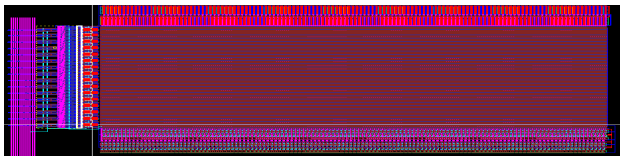

Figure 4: Layout of one way of our data cache.

	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%

Figure 5: Dynamic access energy of a configurable cache compared with conventional four-way set-associative cache.

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 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.

Notice that we are essentially using 6 index bits for a four-way cache, 7 index bits for a two-way cache, and 8 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. Our approach could easily be extended for 8 (or more) ways.

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 [6] layout tools and we extracted the circuit from the layout. The technology we used was TSMC 0.18 CMOS, the most advanced CMOS technology available to universities through the MOSIS program [19]. The dimension of our SRAM cell was 2.4µm x 4.8µm, using conventional six-transistor SRAM cells. We used Cadence's Spectra to simulate the circuit. 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 Figure 5. These energies include dynamic power only, not static. *Cnv* stands for conventional cache, *Cfg* stands for configurable cache. The numbers below *Cnv* and *Cfg* are the size and associativity of the cache, e.g., 8K2W means an 8 Kbyte, two-way set associative cache. The energy savings of the configured two-way 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. 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. 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 also changed two inverters on the critical path into NAND gates: one inverter after the decoder, and one after the comparator. Although a NAND gate is typically slower than an inverter, we can increase the NAND gate transistor sizes in the configurable cache to achieve the speed of the original inverter of the base cache. We measured the original critical path delay of the cache to be 1.28 ns. We determined that changing two inverters along the critical path to NAND gates, having the same size transistors as the original inverters, increased the critical path length by 0.062 ns, or by 4.8%. By increasing the sizes of certain transistors in the NAND gates to three times their original size, we were able to reduce the critical path delay to the original 1.28 ns. Replacing the inverters by NAND gates with larger transistors resulted in a less than 1% area increase of the entire cache.

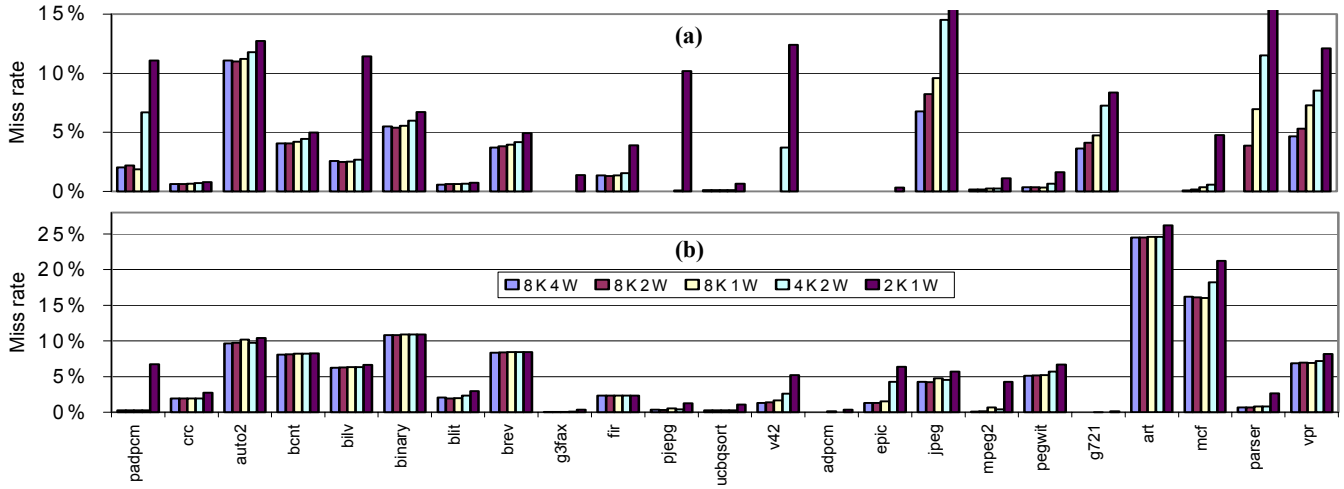

Figure 6: Instruction (a) and data (b) cache miss rates for way concatenate caches (the 8K caches) and way shut down caches (the 4K and 2K caches).

One might ask why the original inverter's transistors weren't sized bigger to achieve a faster cache access time. The reason is that the original inverters themselves only contributed a small amount to critical path delay – tripling the two inverters' transistor sizes would only decrease the critical path delay by 0.023 ns, representing a mere 1.8% improvement.

4.4 Experiments

To determine the benefits of our configurable cache with respect to reducing dynamic power consumption and hence energy, we simulated a variety of benchmarks for a variety of cache configurations using SimpleScalar. The benchmarks included programs from Motorola's Powerstone suite [18] (*padpcm, crc, auto2, bcnt, bilv, binary, blit, brev, g3fax, fir, pjpeg, ucbqsort, v42*), MediaBench [16] (*adpcm, epic, jpeg, mpeg2, pegwit, g721*) and several programs from Spec 2000 [12] (*mcf, parser, vpr, art)*. We used the sample test vectors that came with each benchmark as program stimuli.

All of the energy values we report in this section represent those from dynamic power consumption. The original way-shutdown method [2] was also intended to reduce dynamic power. Thus, we also generated data for way shutdown, and compare our data with that data.

4.4.1 Results

Figure 6(a) shows instruction cache miss rates for the benchmarks for three configurations of our way-concatenatable cache: 8 Kbytes with 4-way associativity (8K4W), 8 Kbytes with 2-way associativity (8K2W), and 8 Kbytes with 1-way (direct-mapped) associativity (8K1W). The figure also shows miss rates for two configurations of a way shutdown cache: 4 Kbytes with 2-way associativity (4K2W), meaning 2 of the 4 ways have been shut down, and 2 Kbytes with 1-way associativity (2K1W), meaning 3 of the 4 ways have been shut down. Figure 6(b) shows data cache miss rates for the same cache configurations.

Let us begin by looking at the first three configurations, representing the way concatenate configurations, which use all 8 Kbytes of the cache. 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 examples, like *jpeg* and *parser*, do much better with four-way caches. *jpeg*'s miss rate is only 6.5% with a four-way instruction cache, but 9.5% with a one-way 8 Kbyte cache. *parser*'s miss rate is nearly 0% with a four-way instruction cache, but is 7% with a one-way 8 Kbyte cache.

Looking now at the latter two configurations, representing way shutdown configurations, we see significant increases in the miss rate for many examples. For example, *v42* has a nearly 0% miss rate for all three way-concatenate instruction cache configurations, but has 4% and 12% miss rates for the way shutdown configurations. 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 concatenation does not.

We computed energy data for the benchmarks, for two different types of configurable instruction and data caches: caches supporting way concatenation only and supporting way shutdown only. For each benchmark, we simulated all possible cache configurations. We show the energy for those configurations for one of the benchmarks, *g3fax*, in Figure 7. The x-axis represents each configuration. The first configuration is I8KD8KI4D4, representing an instruction cache with 8 Kbytes active (I8K), a data cache with 8 Kbytes active (D8K), with the instruction cache configured to be four-

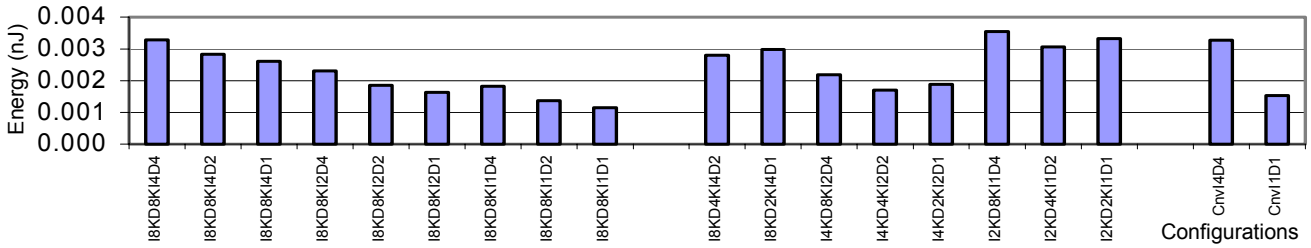

Figure 7: Energy of *g3fax* under way concatenation, and under way shutdown, considering dynamic power only.

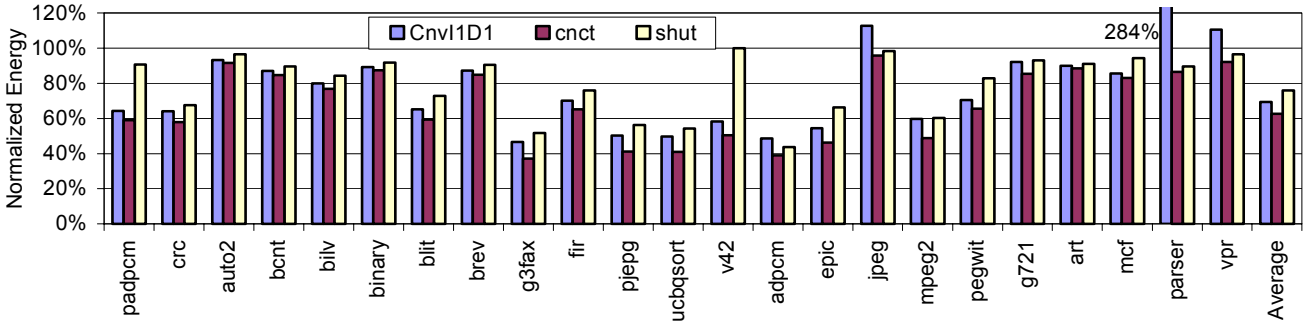

Figure 8: Energy consumption compared to an 8 Kbytes four-way cache, for way concatenation and shut down, considering dynamic power only, with *k_miss_energy*=50.

way set associative (I4), and the data cache configured to be four-way set associative (D4). The first group of nine configurations represents configurable caches using only way concatenation. The second group of eight configurations represents configurable caches using only way shutdown. The last group of two represents the conventional (non-configurable) four-way and direct mapped cache configurations.

We generated such data for every benchmark, found the lowest energy configuration for each group, and summarized those best configurations in Table 2. 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 *mpeg2, jpeg*, *g721*, *brev*, *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.

Example	Best	Example	Best
padpcm	I8KD8KI1D1	ucbqsort	I4KD4KI1D1
crc	I4KD4KI1D1	v42	I8KD8KI1D1
auto2	I8KD4KI1D1	adpcm	I2KD8KI1D1
bcnt	I8KD2KI1D1	epic	I8KD8KI1D1
bilv	I4KD4KI1D1	jpeg	I8KD8KI4D2
binary	I8KD2KI1D1	mpeg2	I8KD8KI1D2
blit	I2KD8KI1D1	g721	I8KD8KI2D1
brev	I8KD4KI1D2	art	I4KD8KI1D1
g3fax	I4KD4KI1D1	mcf	I8KD8KI1D1
fir	I8KD2KI1D1	parser	I8KD8KI4D1
pjepg	I4KD8KI1D1	vpr	I8KD8KI2D1
pegwit	I8KD8KI1D1		

Table 2**:** Cache configuration yield lowest system energy, considering dynamic power only.

Figure 8 shows normalized energy, using the energy of a conventional four-way cache as 100%. *CnvI1D1* corresponds to using direct-mapped instruction and data caches, *cnct* to using way-concatenate caches, and *shut* to using way-shutdown caches.

4.4.2 Main Observations

The first observation we make from the data in Figure 8 is that a configurable cache results in significant energy savings for many examples, compared to either a non-configurable four-way set-associative cache or a non-configurable direct-mapped cache. A way-concatenatable cache results in an average energy savings of 37% compared to a conventional four-way cache, with savings over 60% for several examples. Compared to a conventional direct mapped cache, the average savings are more modest, but the direct mapped cache suffers large penalties for some examples – up to 284% for *parser,* with degraded performance in several examples.

The second observation we make is that way concatenation is clearly better than way shutdown for reducing dynamic power. Way concatenation saves more

energy than way shutdown for nearly every benchmark, sometimes saving 30-50% (e.g., for *padpcm* and *v42*).

4.4.3 Impact of k_miss_energy ratio

In our energy calculation equation, we include the memory access energy and the processor stall energy, using the ratio *k_miss_energy* to represent the ratio of the sum of the memory access and processor stall energy with the 8 Kbyte 4-way set-associative cache access energy. We showed the results for *k_miss_energy*=50 in Figure 8. We also generated results for a ratio of 200, but we omit the plots for space reasons. However, those results still showed significant energy savings for many examples, compared to either a non-configurable four-way set-associative cache or a non-configurable direct-mapped cache, resulting in an average savings across all the benchmarks of 26% and 10%, respectively. Perhaps more importantly, we saw big penalties associated with a direct mapped cache and a way shutdown cache for certain examples – over 800% for *parser*, for example.

5. Adding Way Shutdown for Static Energy Reduction

We have thus far focused on dynamic power consumption, traditionally the major consumer of power in CMOS technology. As CMOS technology continues to evolve, transistors with lower threshold voltages are becoming common. Low threshold 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 voltages also result in more sub-threshold leakage current through the transistor, resulting in increased static power consumption. Thus, static power is becoming a greater concern. Some researchers are thus working on leakage power reductions, such as DRG-Cache [1].

We observe in Figure 6(a) and (b) that although way shutdown increases the miss rate for some benchmarks, for other benchmarks, way shutdown has negligible impact. Such negligible impact means that the benchmark did not need the full capacity (8 Kbyte) 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 [2] 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 since we want to use way shutdown to reduce static power, we use the shutdown method by Powell [20], involving a circuit level technique called gated-Vdd, shown in Figure 9. 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 only about 5%. Powell showed 8% performance overhead from the extra transistor.

5.2 Energy Calculations

Recall in Equation 1 that we defined:

$$energy\_static = cycles * energy\_static\_per\_cycle.$$

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

We have defined a constant *k_static* which is the percentage of static energy to the total energy dissipation of per cycle. To consider future CMOS technology trends, we evaluate the situations where *k_static* is 30% and 50%.

5.3 Experiments

Figure 10 shows the energy consumption of *g3fax* for a configurable cache supporting both way concatenation and way shutdown, considering both dynamic and static

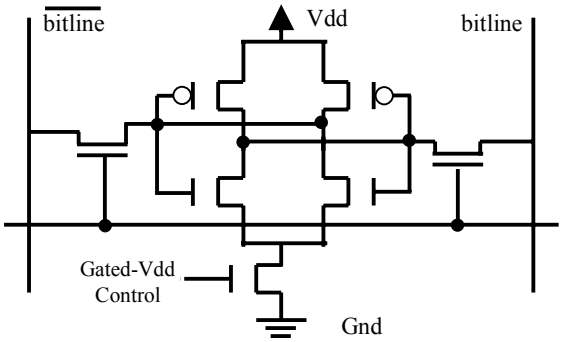

Figure 9: SRAM cell with a NMOS gated Vdd control.

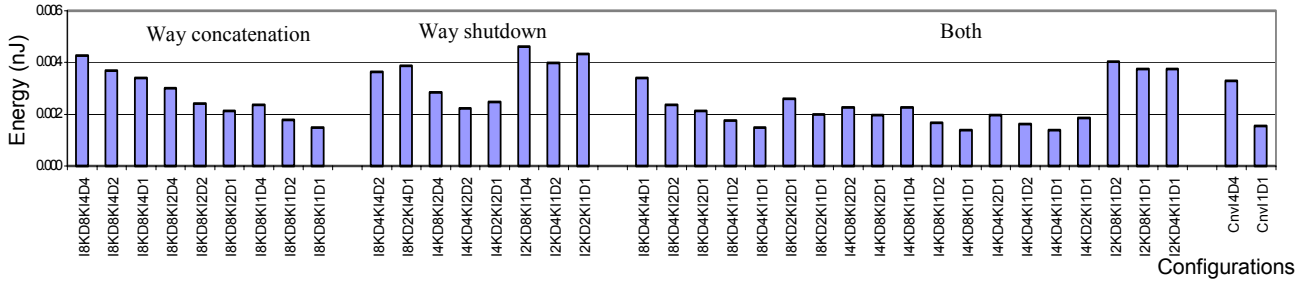

Figure 10: Energy of *g3fax* under way concatenation, way shutdown, and both, considering dynamic and static power.

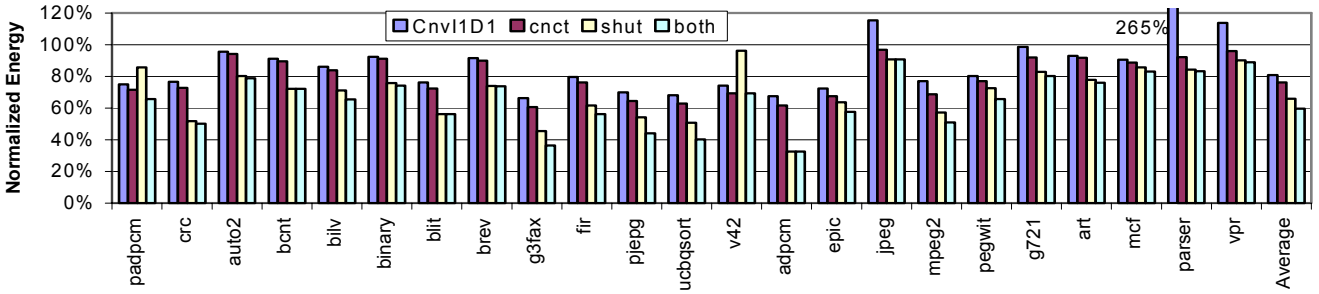

Figure 11: Energy consumption compared to an 8 Kbytes four-way cache, for way concatenation and shutdown, considering both dynamic and static power, *k_miss_energy*=50, *k_static*=30%.

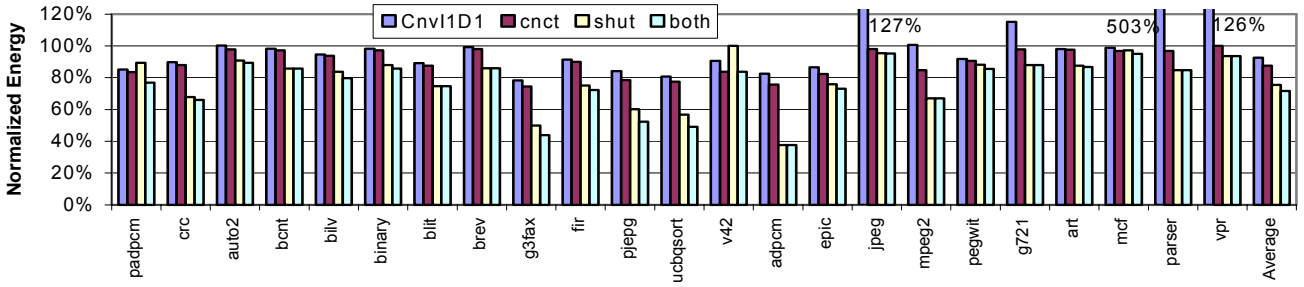

Figure 12: Energy consumption compared to an 8 Kbytes four-way cache, for way concatenation and shutdown, considering both dynamic and static power, *k_miss_energy*=200, *k_static*=50%.

power, assuming *k_static*=30%. For this example, we can see that the combination of way concatenation and way shutdown is better than either method alone.

Figure 11 is the normalized energy consumption of the benchmarks, using the conventional four-way cache's energy as 100%. As in Figure 8, we compare a conventional direct mapped cache (*CnvI1D1*), configurable caches supporting concatenate only (*cnct*), shutdown only (*shut*), and both (*both*). We see that the average energy savings (rightmost column) compared with a conventional four way cache is over 40%.

Figure 12 shows results for increasing *k_static* even further, to an extremely large 50%, and increasing *k_miss_energy* to 200 (representing possible values in the future). We found that, although way shutdown alone gains most of the savings in most examples, in some cases (like *v42* and *padpcm*) the combination of both concatenate and shutdown is necessary – way shutdown alone increases misses too much for some examples.

Table 3 shows the best cache configuration with *k_static*=30%, *k_miss_energy =50*. We see that several examples still need the full 8 Kbytes of cache, but may work best with ways concatenated.

6. Using a Configurable Cache

A configurable cache could be used as follows. 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.

Example	Best	Example	Best
padpcm	I8KD4KI1D2	ucbqsort	I4KD4KI1D1
crc	I2KD4KI1D1	v42	I8KD8KI1D1
auto2	I4KD2KI1D1	adpcm	I2KD2KI1D1
bcnt	I2KD2KI1D1	epic	I2KD8KI1D1
bilv	I4KD2KI1D1	jpeg	I8KD2KI4D1
binary	I4KD2KI1D1	mpeg2	I4KD4KI1D2
blit	I2KD2KI1D1	g721	I8KD2KI2D1
brev	I2KD2KI1D1	art	I4KD2KI1D1
g3fax	I4KD2KI1D1	mcf	I4KD4KI1D1
fir	I4KD2KI1D1	parser	I8KD4KI4D1
pjepg	I4KD2KI1D1	vpr	I8KD2KI2D1
pegwit	I4KD4KI1D1		

Table 3**:** Optimal cache configuration when both dynamic and static energy are considered, *k_static* =30%, *k_miss_energy* = 50.

Much of our own ongoing related work has sought to automate the process of finding the best cache configurations through dynamic tuning, by implementing an automatic cache tuner that finds the best configuration. We have also developed a simulation-based method that finds the Pareto-optimal set of cache configurations for a given program [28].

7. Conclusions

We have introduced a novel configurable cache design method called way concatenation. When considering dynamic power, we showed average energy savings of 37% compared to a conventional four-way set-associative

cache, and we showed way concatenation to be superior to previously proposed way shutdown methods. To also save static power, we extended the configurable cache to include a way shutdown method, and we showed this configurable cache to be the most energy efficient for all benchmarks considered, across a wide range of technological assumptions, with average savings of 40%. The method is simple and imposes little area overhead.

In other work, we also explored the energy benefits of a cache line concatenation [27]. We will continue our work to combine with existing multi-cycle cache access methods like phased lookup, way-prediction, pseudo-associativity, and filter caching. We are also developing an on-chip component that can automatically and dynamically tune the cache's configuration to a particular executing program. We plan to also explore the dynamic tuning of multilevel cache architectures.

8. Acknowledgements

This work was supported by the National Science Foundation (grants CCR-9876006 and CCR-0203829) and by the Semiconductor Research Corporation.

References

[1] A. Agarwal, H. Li, and K. Roy, "DRG-Cache: A Data Retention Gated-Ground Cache for Low Power," Design Automation Conf., June 2002.

[2] D.H. Albonesi, "Selective Cache Ways: On-Demand Cache Resource Allocation," Journal of Instruction Level Parallelism, May 2000.

[3] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures," Int. Symp. on Microarchitecture, 2000.

[4] B. Batson and T.N. Vijaykumar, "Reactive-Associative Caches," Int. Conf. on Parallel Architectures and Compilation Techniques, Sep. 2001.

[5] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0," Univ. of Wisconsin-Madison Computer Sciences Dept. Technical Report #1342, June 1997.

[6] Cadence, http://www.cadence.com

[7] B.Calder, D. Grunwall, and J. Emer, "Predictive Sequential Associative Cache," Int. Symp. on High Performance Computer Architecture, Feb. 1996.

[8] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M.L. Scott, "Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power," Int. Conf. on Parallel Architectures and Compilation Techniques, 2002.

[9] A. Hasegawa, I.Kawasaki, K.Yamada, S.Yoshioka, S. Kawasaki, and P. Biswas, "SH3: High code density, low power," IEEE Micro, Dec. 1995.

[10] J. L. Hennessy and D .A. Patterson, "Computer architecture Quantitative Approach," 2nd Edition, Morgan-Kaufmann Publishing Co., 1996.

[11] M. Huang, J. Renau, S.M. Yoo, and J. Torrellas, "L1 Data Cache Decomposition for Energy Efficiency," Int. Symp. on Low Power Electronics and Design, 2001.

[12] http://www.specbench.org/osg/cpu2000/

[13] K. Inoue, T. Ishihara, and K. Murakami, "Way-Predictive Set-Associative Cache for High Performance and Low Energy Consumption," Int. Symp. On Low Power Electronics and Design, 1999.

[14] H. Kim, A.K. Somani, and A. Tyagi, "A Reconfigurable Multi-function Computing Cache Architecture," IEEE Transactions on VLSI, Vol. 9, No. 4, pp. 509-523, Aug. 2001.

[15] J. Kin, M. Gupta and W. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," Int. Symp. on Microarchitecture, pp. 184-193, Dec. 1997.

[16] C. Lee, M. Potkonjak and W. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," Int. Symp. on Microarchitecture, 1997.

[17] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart Memories: A Modular Reconfigurable Architecture," ACM SIGARCH Computer Architecture News , Volume 28, Issue 2, 2000.

[18] A. Malik, B. Moyer and D. Cermak, "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility," Int. Symp. on Low Power Electronics and Design, June 2000.

[19] The MOSIS Service, http://www.mosis.org

[20] M. Powell, S.H. Yang, B. Falsafi, K. Roy, and T.N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," Int. Symp. on Low Power Electronics and Design, 2000.

[21] M. Powell, A. Agarwal, T.N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping," Int. Symp. on Microarchitecture, 2001.

[22] P. Ranganathan, S. Adve, and N.P. Jouppi, "Reconfigurable Caches and their Application to Media Processing," Int. Symp. on Computer Architecture, 2000.

[23] Glen Reinman and N.P. Jouppi. CACTI2.0: An Integrated Cache Timing and Power Model, 1999. COMPAQ Western Research Lab.

[24] Semiconductor Industry Association. Int. Technology Roadmap for Semiconductors: 1999 edition. Austin, TX: Int. SEMATECH, 1999.

[25] S. Segars, "Low power design techniques for microprocessors," Int. Solid-State Circuits Conf. Tutorial, 2001.

[26] J. Yang and R. Gupta, "Energy Efficient Frequent Value Data Cache Design," Int. Symp. on Microarchitecture, Nov. 2002.

[27] C. Zhang, F. Vahid, and W. Najjar, "Energy Benefits of a Configurable Line Size Cache for Embedded Systems," Int. Symp. on VLSI Design, 2003.

[28] C. Zhang and F. Vahid, "Cache Configuration Exploration on Prototyping Platforms," IEEE Int. Workshop on Rapid System Prototyping, June 2003.

[29] M. Zhang and K.Asanović, "Highly-Associative Caches for Low-Power Prossrors," Kool Chips Workshop, in conjunction with Int. Symp. On Microarchitecture, Dec. 2000.