

A Highly-Configurable Cache Architecture for Embedded Systems

Abstract

Energy consumption is a major concern in most forms of 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 using it. 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 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. A study of 23 programs drawn from Powerstone, MediaBench and Spec2000 show that the cache saves energy for every program compared to conventional four-way set-associative as well as direct mapped caches. We show that the proposed cache architecture improves the dynamic power consumption compared to the way-shutdown cache. Furthermore, we extend the cache architecture to also support a way shutdown method designed to reduce the static energy that is increasing in importance in newer CMOS technologies.

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 power [11][15]. A direct mapped cache is more power efficient, consuming only about 30% the power of a four-way set associative cache of the same size. It also has a shorter access time. While its hit rate can be acceptable for many applications, for some it exhibits a very poor hit rate and hence suffers from poor performance and power 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 power and longer access time.

Deciding on a cache's total size involves a similar dilemma. A small cache is more power 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 power 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 for solving this dilemma in embedded systems is to manufacture multiple versions of the same processor each with a cache architecture tuned to a specific set of applications. Another is to provide a synthesizable core rather than a physical chip and have the cache architecture modified for the intended application. Both options unduly increase the unit cost. The second option suffers from a longer time to market [23].

The variety of cache architectures found in modern embedded microprocessors, summarized in Table 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. The configurability is 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 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-concatenatable and shutdown cache architecture, discussing the performance and area overhead imposed by such configurability. We demonstrate significant energy savings compared to non-configurable caches, for applications drawn from the Powerstone [18], MediaBench [17] and the Spec2000 [12] benchmark suites.

2. Cache Energy vs. Performance

After examining typical cache configurations of several popular embedded microprocessors, summarized in Table 1, we chose to use a base cache of 8 Kbytes having four-way set-associativity and a line size of 32 bytes.

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.

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 8 Kbyte cache, 3.8% for a two-way 8 Kbyte cache and only 2.9% for four-way 8 Kbyte cache [9]. Though these differences may appear small, they in fact translate to big performance differences, due to the large cycle penalty of misses. Thus, although energy per cache access may be higher for a four-way cache than for a one-way cache (in the remainder of this paper, we will

Processor	Instruct. Cache			Data Cache			Processor	Instruct. Cache			Data Cache		
	Size	As.	Line	Size	As.	Line		Size	As.	Line	Size	As.	Line
AMD-K6-IIIIE	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 lie	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.

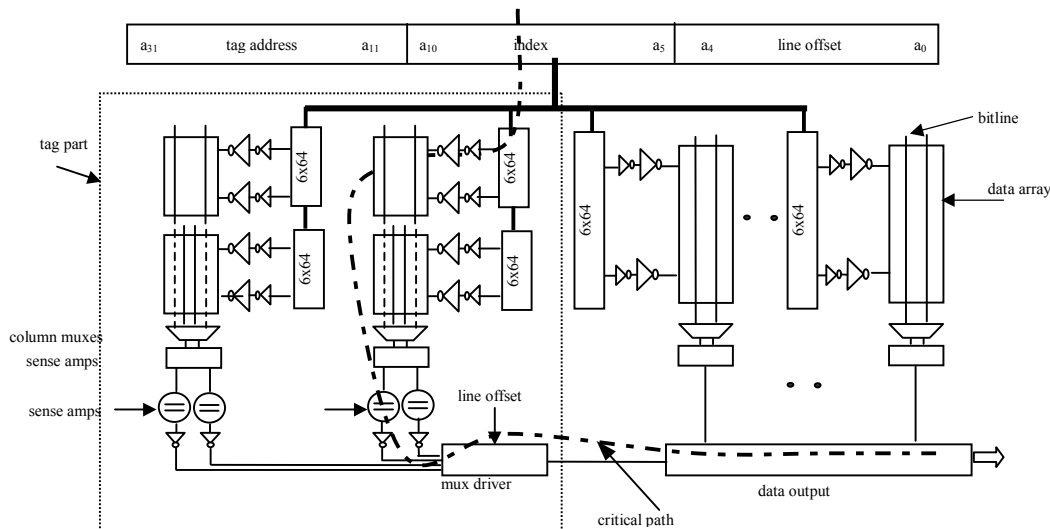


Figure 1: A four way set associative cache architecture with the critical path shown.

sometimes refer to a direct-mapped cache as having one way), 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 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. Notice that the hit rates for both are better for two ways than for one way, but the additional improvement using four ways is very small. Figure 2(b) shows overall energy for these two examples (considering cache energy, off-chip memory energy, and CPU stall energy), 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%.

3. Previous Work

Several cache lookup variations have been proposed by researchers to reduce set-associative cache access energy. Phased-lookup cache [8] use 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,

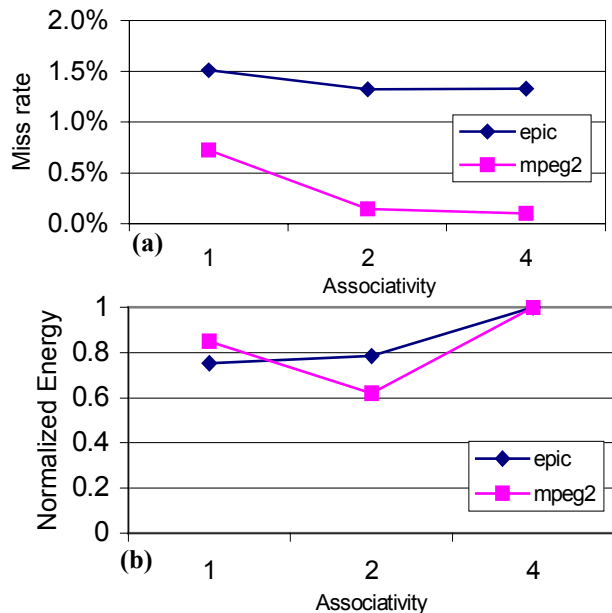


Figure 2: Miss rate (a) and normalized energy (b) of *epic* and *mpeg2* on 8K data caches of different associativities.

resulting in less data way access energy at the expense of longer access time. Way predictive set-associative caches [13][20] access one tag and data array initially, and only

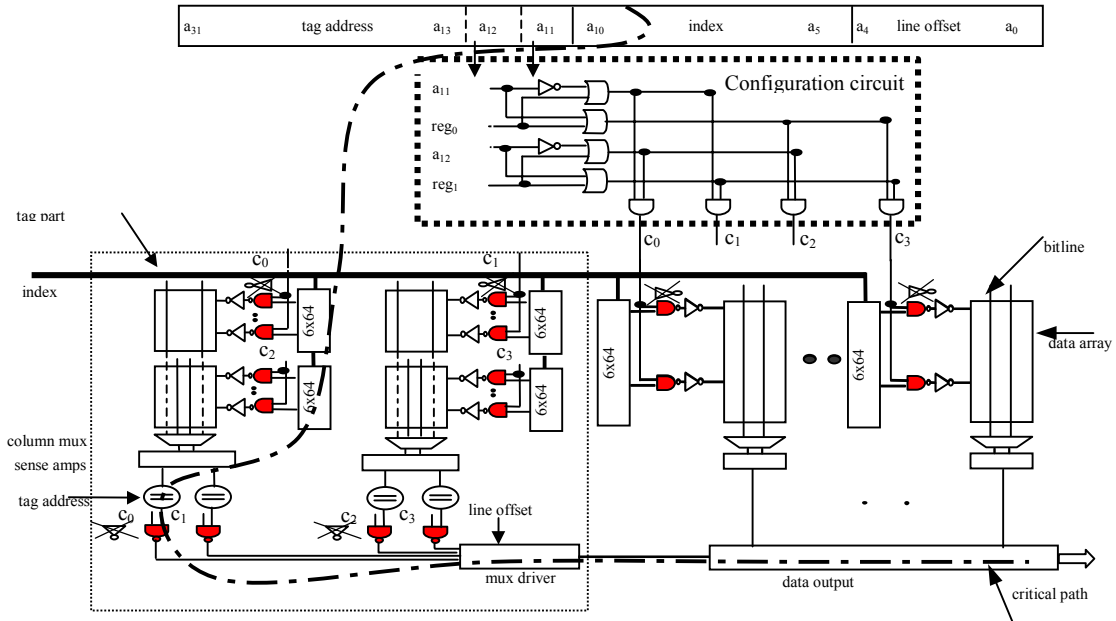
access the other arrays if that initial array did not result in a match, again resulting in less energy at the expense of longer average access time. Reactive-associative cache (RAC) proposed by Batson [4] also uses way prediction and checks the tags as a conventional set associative cache, but the data array is arranged in the way as a direct mapped cache. Since data from RAC proceeds without any way-select multiplexor, it achieves the speed of a direct mapped cache but consumes less energy than that of a conventional set associative cache. Pseudo Set-Associative Cache (PSAC) [6] is a set associative cache that has multiple hit times. The ways of a PSAC can be probed independently and consume less energy than that of a conventional set associative cache. Dropsho [7] 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 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 [16] 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.

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 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. 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 [15]. 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 will show that our way concatenation approach is superior to way shutdown in reducing dynamic power.

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 multi-cycle cache accesses during a hit, but those methods could be combined with ours to reduce

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



the number of misses and hence off-chip memory accesses further. 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. Our method's way shutdown also reduces static power, which the other methods above don't.

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 fix size of cache, which we show to be superior in 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.

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 $a11$ and $a12$ will essentially 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.

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 cache. We used Cadence as the layout tools and we extract the circuit from the layout. The technology we are using is TSMC 0.18, a modern CMOS technology. The dimension of the SRAM cell is $2.4\mu\text{m} \times 4.8\mu\text{m}$, using conventional six-transistor SRAM cells. We use Spectra to simulate the netlist of the extracted circuits. We measured the cache's access time and energy consumption from the outputs of the simulation. We measured the energy of the various

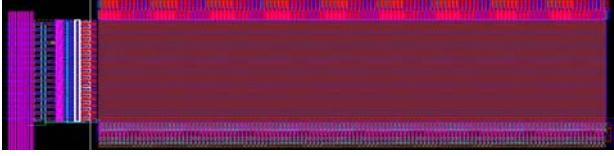


Figure 4: Layout of one way of cache data

Cnv 8W4	Cfg 8W4	Cfg 8W2	Cfg 8W1	Cfg 4W2	Cfg4W1	Cfg2W1
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%

Table 2: Energy savings of configurable cache compared with conventional cache

parts of a conventional four-way set-associative cache for accessing a cache, 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 2. These energies considered 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, e.g., *8w2* means an 8 Kbyte, 2-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, bitlines and wordlines 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 does such a cache slow down access time. This is especially important because the cache access time often represents the critical path for a microprocessor, and thus cache access slowdown may slowdown the system clock. 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 select the size of transistor in the configure circuit large enough that the speed of the configure circuit is faster than that of the decoder. This is reasonable because we only have four OR and four AND gate in the configure circuit. However, we have changed two inverters on the critical path into NAND gates. NAND gates are slightly slower than inverters. Based on our layout of the cache, we found out that tripling the size of the transistors of the original inverter will make the new NAND gate as fast as the original inverter. Thus, we do not have access time overhead, but instead have slight area overhead. Considering the area overhead of the configure circuit and the NAND gate along the wordline driver, the total area overhead due to way concatenation is less than 1%, as measured from our cache layout.

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 [14] (*padpcm*, *crc*, *auto2*, *bcnt*, *bilv*, *binary*, *blit*, *brev*, *g3fax*, *fir*, *pjpeg*, *ucbqsort*, *v42*), MediaBench [17] (*adpcm*, *epic*, *jpeg*, *mpeg2*, *pegwit*, *g721*,) and some programs from Spec 2000 (*mcf*, *parser*, *vpr*, *art*). We used the sample test vectors that came with each benchmark as program stimuli.

4.4.1 Energy Calculations

We considered not only the cache energy access, but also the energy for accessing the next level of memory, and the stall energy of the microprocessor. These later two components of energy have often been overlooked in previous works, yet they should be considered when computing the cost of a miss. We use the following equations to compute energy:

Equation 1:

$$\text{energy_dynamic} = \text{cache_hits} * \text{energy_hit} + \text{cache_miss} * \text{energy_miss}$$

Equation 2:

$$\text{energy_miss} = \text{energy_offchip_access} + \text{energy_uP_stall} + \text{energy_cache_block_fill}$$

We later evaluate the impact of different ratios of those terms, to account for different architectures and technologies.

The values for *cache hits* and *cache misses* were determined from the SimpleScalar simulations for each cache configuration. The *energy_hit* value of each cache configuration was determined by the simulation of circuits extracted from our layout. We assumed a *miss_cycles* of 20 cycles.

The *energy_offchip_access* value is the energy consumed by off-chip memory, which is heavily dependent on the size and technology of the memory chip. The *energy_uP_stall* value was the energy consumed when microprocessor is stalled to wait instruction or/and data.. Typical power consumption for an embedded microprocessor is around hundreds of milliwatts.

Energy evaluation of a microprocessor system, for the purposes of evaluating an architectural feature like a configurable cache, is challenging. On the one hand, to be “accurate,” we could evaluate the architectural feature in a “real” microprocessor system. While accurate, those results may not apply to other systems, which may use different processors, memories, and caches. 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 thus created a ratio *k_miss_energy*, equal to *energy_miss* divided by *energy_hit*. After examining typical microprocessors, we selected *k_miss_energy* = 50 and 200 to represent a range of different systems. That means access to off-chip memory is 50 to 200 times more expensive than access to the on-chip cache.

Figure 6: Instruction (a) and data (b) cache miss rates for different cache sizes and associativities. The 8K caches correspond to way concatenation; the 4K and 2K caches to way shutdown.

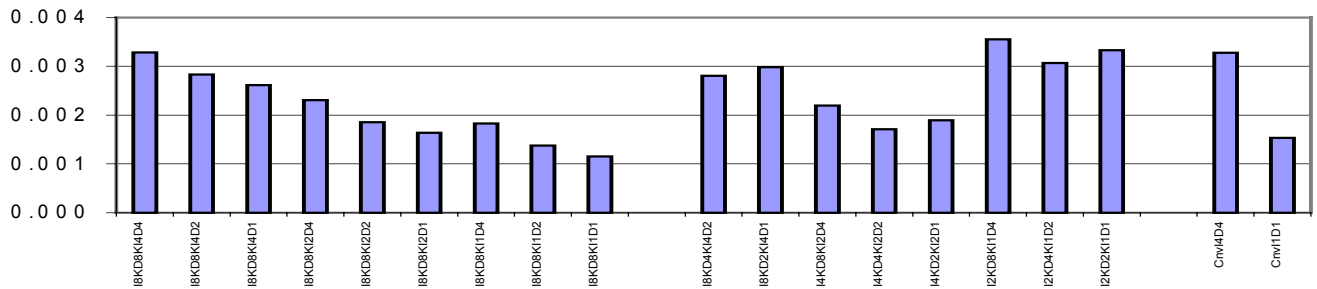
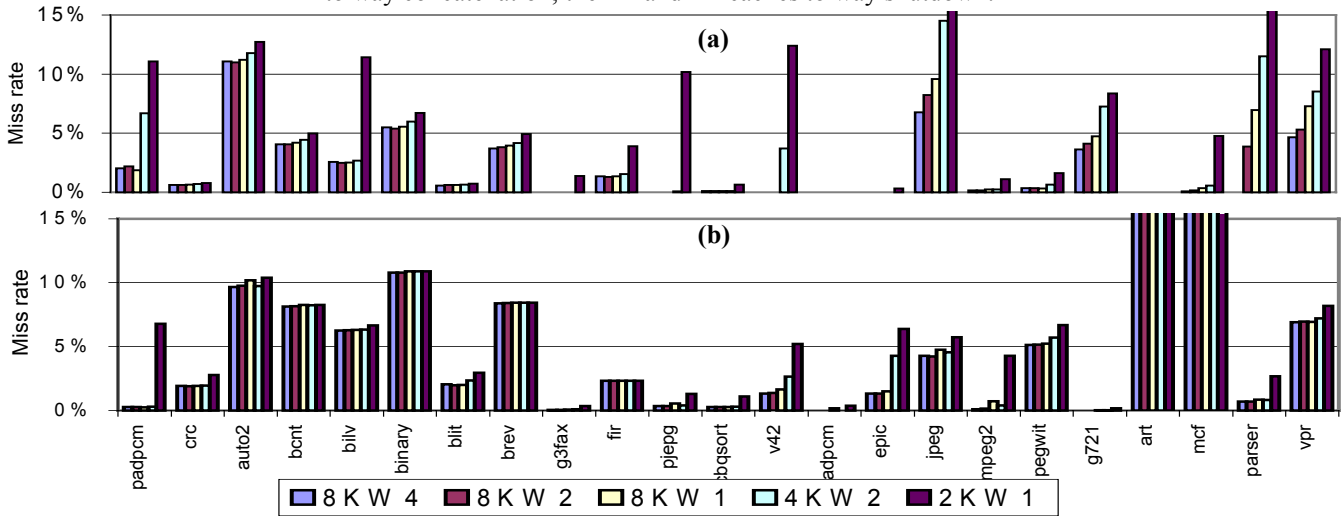


Figure 5: Energy of *g3fax* under way concatenation, way shutdown, considering dynamic power only.

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.

4.4.2 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, 8 Kbytes with 2-way associativity, and 8 Kbytes with 1-way associativity (direct mapped). The table also shows miss rates for two configurations of a way shutdown cache: 4 Kbytes with 2-way associativity (meaning 2 of the 4 ways have been shut down), and 2 Kbytes with 1-way associativity (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 cache. *parser*'s miss rate is nearly 0% with a four-way instruction cache, but is 7% with a one-way 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 concatenate does not.

We computed energy data for the benchmarks, for three different types of configurable instruction and data caches: caches supporting way concatenation only, supporting way shutdown only, and caches supporting both techniques. For each benchmark, all possible cache configurations were simulated. We show the energy for those configurations for one of the benchmarks, *g3fax*, in Figure 5. The x-axis represents each configuration. The

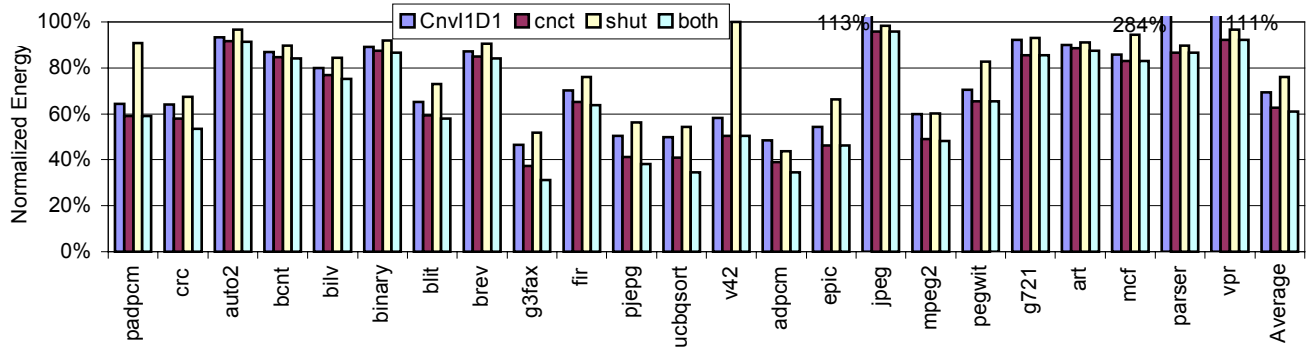


Figure 7: Energy consumption compared to an 8 Kbytes four-way cache, for way concatenation and shut down, considering dynamic power only, $k_miss_energy=50$.

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-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 3. Figure 7 shows normalized energy, using the energy of a conventional four-way cache as 100%. 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 *mpeg*, *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.

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	I8KD8KI14D1
pjpeg	I4KD8KI1D1	vpr	I8KD8KI2D1
pegwit	I8KD8KI1D1		

Table 3: Cache configuration yield lowest system energy, considering dynamic power only.

mpeg does best with a one-way instruction cache but a two-way data cache. Our configurable cache saves an average of 44% energy compared to a conventional four-way set associative cache, ranging from 16% savings for *parser* to 67% for *adpcm*.

4.4.3 Main Observations

The first observation we make from this data 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 42% compared to a conventional four-way cache. Compared to a conventional direct mapped cache, the savings are more modest, but at the cost of large penalties for some examples – up to 284% for *parser*, and degraded performance in several examples. Savings on particular examples are quite large compared to both conventional caches. A configurable cache not only does better than conventional caches on average, but is in fact the lowest energy solution for every example.

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%.

4.4.4 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 energy and processor stall with the 8 Kbyte 4-way set-associative cache access energy. We showed the result of $k_miss_energy=50$ in Figure 7. We also created results for a of 200, but omit the plots for space reasons. However, those results still show 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

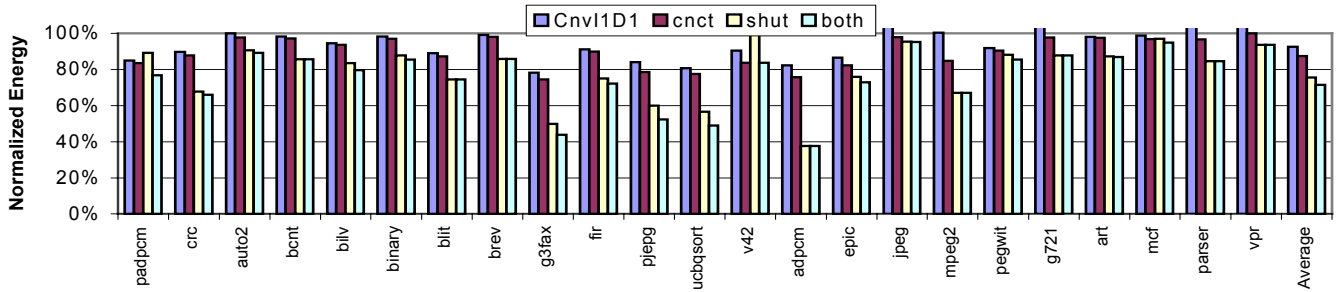


Figure 10: Energy consumption compared to an 8 Kbytes four-way cache, for way concatenation and shutdown, considering both dynamic and static power, $k_miss_energy=50$, $k_static=30\%$

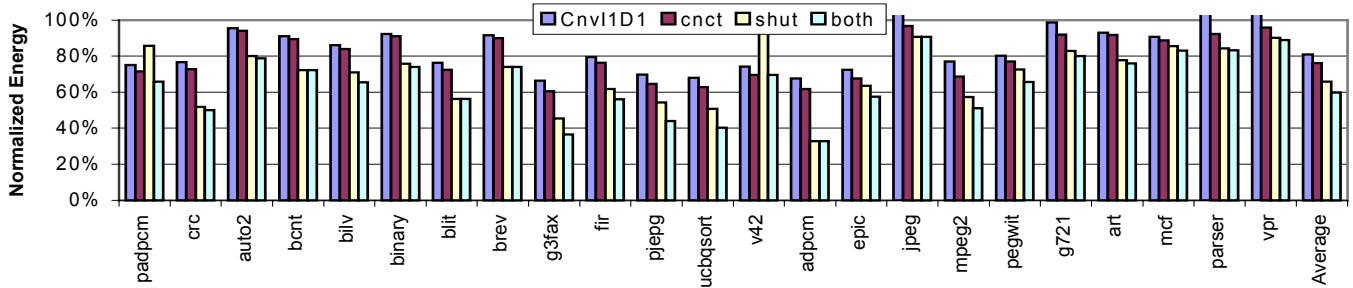


Figure 11: Energy consumption compared to an 8 Kbytes four-way cache, for way concatenation and shutdown, considering both dynamic and static power, $k_miss_energy=200$, $k_static=50\%$.

Figure 10 is the normalized energy consumption of the benchmarks, using the conventional four-way cache’s energy as 100%. As in Figure 7, we compare a conventional direct mapped cache (*Cnv11D1*), and configurable caches supporting concatenate only, shutdown only, and both. We see that the average energy savings (rightmost column) compared with a conventional four way cache is over 40%.

We also obtained results for increasing k_static even further, to a very large 50%, but don’t show the plots due to space limitations. 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 the miss rate too much for some examples. Table 4 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.

Figure 11 illustrates the impact of a higher value of $k_miss_energy=200$. While total savings are reduced, the best cache is still the configurable cache supporting both way concatenation and way shutdown.

6. Using a Configurable Cache

A configurable cache would likely 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. Because our configurable cache supports only static (re)configuration, we can safely assume that the cache will be flushed after each configuration change.

Much of our own ongoing related work has sought to automate the process of finding the best cache

Example	Best	Example	Best
padpcm	I8KD4K11D2	ucbqsort	I4KD4K11D1
crc	I2KD4K11D1	v42	I8KD8K11D1
auto2	I4KD2K11D1	adpcm	I2KD2K11D1
bcnt	I2KD2K11D1	epic	I2KD8K11D1
bilv	I4KD2K11D1	jpeg	I8KD2K14D1
binary	I4KD2K11D1	mpeg2	I4KD4K11D2
blit	I2KD2K11D1	g721	I8KD2K12D1
brev	I2KD2K11D1	art	I4KD2K11D1
g3fax	I4KD2K11D1	mcf	I4KD4K11D1
fir	I4KD2K11D1	parser	I8KD4K14D1
pjpeg	I4KD2K11D1	vpr	I8KD2K12D1
pegwit	I4KD4K11D1		

Table 4: Optimal cache configuration when both dynamic and static energy are considered, $k_static=30\%$, $k_miss_energy=50$.

configuration (reference omitted for blind review).

7. Conclusions

We have introduced a novel configurable cache design method called way concatenation. When considering dynamic power, we showed average energy savings of 40% 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 all benchmarks considered, across a large range of technological assumptions. The method is very simple and imposes very little area and performance overhead.

Our future work plans include extending the cache to support line concatenation and to be combined 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 tune the cache's configuration to a particular executing program.

References

- [1] A. Agarwal, H. Li, and K. Roy. DRG-Cache: A Data Retention Gated-Ground Cache for Low Power. Design Automation Conf., New Orleans, June 2002.
- [2] D.H. Albonesi. Selective Cache Ways: On-Demand Cache Resource Allocation. J. 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 In 33rd Int. Symp. on Microarchitecture, 2000
- [4] B. Batson, T.N.Vijaykumar. Reactive-Associative Caches. International Conference on Parallel Architectures and Compilation Techniques. Barcelona, Spain, Sept. 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] B.Calder, D.Grunwall, and J. Emer. Predictive Sequential Associative Cache. In Int'l Symp. on High Performance Computer Architecture. February 1996.
- [7] 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. The 11th Int. Conf. on Parallel Architectures and Compilation Techniques, 2002.
- [8] A. Hasegawa, I.Kawasaki, K.Yamada, S.Yoshioka, S. Kawasaki, and P. Biswas, SH3: high code density, low power. IEEE Micro, December 1995
- [9] J. L. Hennessy and D .A. Patterson: Computer architecture Quantitative Approach, 2nd Edition, Morgan-Kaufmann Publishing Co., Menlo Park, CA.1996
- [10] <http://www.cs.ucr.edu/~dalton/platune>
- [11] <http://developer.intel.com/design/strong>.
- [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 Electronic Design 1999
- [14] A. Malik, B.Moyer, and D. Cermak, "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility". Int. Symp. On Low Power Electronic Design of ISLPED,Rapallo, Italy, 26-27 July 2000.
- [15] L. Hwang Lee, B. Moyer, and J. Arends. low-cost Embedded Program Loop Caching- Revisited. Univ. of Michigan Technical Report CSE-TR-411-99, December 1999.
- [16] J. Kin, M. Gupta and W. Mangione-Smith. The Filter Cache: An Energy Efficient Memory Structure. Int. Symp. on Microarchitecture, pp. 184-193, December 1997
- [17] C. Lee, M. Potkonjak and W. Mangione-Smith. MediaBench C. Lee, M. Potkonjak and W. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems, Int. Symp. On Microarchitecture 1997
- [18] A. Malik, B. Moyer, D. Cermak. A Low Power Unified Cache Architecture Providing Power and Performance Flexibility. Int. Symp. on Low Power Electronics and Design. June 2000
- [19] 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. New York, NY, USA: ACM, 2000
- [20] M. D. Powell, A. Agarwal, T.N. Vijaykumar, Babak Falsafi, and Kaushik Roy. Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping. In 34th Int. Symp. on Microarchitecture, 2001
- [21] S.J.E. Wilton, N.P. Jouppi, CACTI: An Enhanced Cache Access and Cycle Time Model, IEEE Journal of Solid-State Circuits, Vol. 31, No. 5, pp. 677-688,1996
- [22] A. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji, "Adapting cache line size to application behavior," in Proceedings of Int. Conf. on Supercomputing 1999.
- [23] Semiconductor Industry Association. International Technology Roadmap for Semiconductors: 1999 edition. Austin, TX: International SEMATECH, 1999.