

Energy Benefits of a Configurable Line Size Cache 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
{[@cs.ucr.edu](mailto: vahid/najjar),
(F. Vahid is also with the Center for Embedded
Computer Systems at UC Irvine)*

Abstract

Previous work has shown that cache line sizes impact performance differently for different desktop programs – some programs work better with small line sizes, others with larger line sizes. Typical processors come with a line size that is a compromise, working best on the average for a variety of programs. We analyze the energy impact of different line sizes, for 19 embedded system benchmarks, and we show that tuning the line size to a particular program can reduce memory access energy by 50% in some examples. Our data argues strongly for the need for embedded microprocessors to have configurable line size caches, and for embedded system designers to put effort into choosing the best line size for their programs.

1. Introduction

Reducing energy consumption is a key issue in battery powered embedded systems. Off-chip memory accesses consume much energy in a microprocessor-based system, because of the high capacitive load and hence power of off-chip buses and memory. One method to reduce such power is to reduce the number of wires that switch per access [1][13]. Another method is to reduce the number of accesses to the off-chip memory. On-chip cache memories help reduce such accesses, and so not only improve performance, but also power.

Caches typically move data to and from off-chip memory in chunks of several bytes, perhaps 16, 32 or 64 bytes, known as the *line size*. When a program exhibits much spatial locality, then a larger line size can reduce the number of microprocessor stalls caused by cache misses. But without spatial locality, a large line size fetches many unnecessary bytes, which not only lengthen cache fill time, but may also

evict needed bytes from the cache, thus increasing off-chip memory accesses and stalls.

If the designer of a mass-produced microprocessor chip does not know what particular program will run on the chip, the designer may choose a line size that works best on average across a wide variety of programs. Table 1 shows the cache line sizes of several popular embedded microprocessors. We can see that there is no agreement on the best line size, but that generally, a line size of 32 bytes seems to be preferred.

We performed experiments that show the benefits of a chip designer creating a cache with a configurable line size rather than picking a particular line size for all programs, and of the chip user tuning the line size to the particular program that will run on the chip. The existence of a fixed program running on an embedded microprocessor is perhaps one of the most key characteristics distinguishing embedded from desktop computing systems, and the tuning of architectures to that fixed program is an area with much potential research.

2. Related work

Experiments in [3] show the importance of a properly configured line size on a cache's miss rate. Five Spec92 benchmarks were found to have minimum miss rates occurring

Processor	Line	Processor	Line
AMD-K6-IIIIE	32	Motorola MPC8540	32/64
Alchemy AU1000	32	Motorola MPC7455	32
ARM7	16	NEC VR5500	32
Hitachi SH7750S (SH4)	32	NEC VR4131	16/32
Hitachi SH7727	16	NEC VR4181	16
IBM PPC 750CX	32	NEC VR4181A	32
IBM PPC 7603	32	FMC Sierra RM7000A	32
IBM750FX	32	SandCraft sr71000	32
IBM403GDX	16	SuperH	32
IBM Power PC 405CR	32	TriMedia TM32A	64
Motorola MPC8240	32	Xilinx Virtex IIPro	32
Motorola MPC823E	16	Triscand A7	16

Table 1: Line size of popular embedded microprocessors.

for line sizes of 32 or 64 bytes, with higher miss rates occurring for sizes 16 bytes, or with 128 bytes or more.

Line size configuration can be done statically or dynamically. Statically means configuring the size before the program executes in its real environment (perhaps based on profiles from sample executions or simulations), while dynamically means modifying the line size as the program executes.

Static cache configuration is supported by many microprocessors available as cores. A designer selects a cache's line size, associativity, and even total size, resulting in a customized cache being generated for mapping onto an eventual chip [4].

Some pre-fabricated microprocessor chips also support static line size configuration. For example, the MIPS R3000/R4000 [4] has a configurable cache line size. Actually, the hardware architecture uses a fixed physical line size [15], but the number of words replaced on a miss could be varied. The Motorola M*CORE supports static configuration of certain other cache parameters, such as the amount of instruction and data associativity [9].

Some recent work focuses on the advantages of dynamically sizing cache lines. Nicolaescu [10] uses profile information to guide a compiler in inserting line size configuration information into a program. Witchel [17] proposed a software-controlled cache line size. A compiler specifies how much data to fetch on a data cache miss. Two hardware implementations are given to support the compiler-controlled cache.

Veidenbaum et al [15] 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 [6] proposed a dynamic variable line size cache. Exploiting the high on-chip 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. This high bandwidth on-chip memory is not available in typical embedded systems.

Our work focuses on static configuration of line size. Though we emphasize the need for a configurable cache on a pre-fabricated microprocessor chip, the conclusions also apply to core-based systems of course. Static configuration has the advantages of less hardware and energy overhead, but the disadvantage of requiring a designer to select the appropriate configuration.

Our work differs from much previous work in that we consider not just performance, but also energy. Furthermore, our energy calculations are rather thorough, considering not just miss rates, but also the energy for off-chip accesses and for microprocessor stalls; factors often overlooked.

3. A Configurable Line Size Cache

3.1 Basic architecture

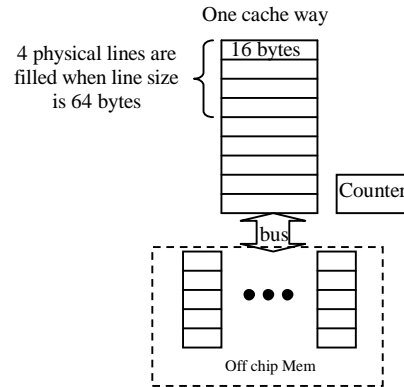


Figure 1: Architecture of a line size configurable cache.

Creating a cache with a configurable line size is relatively straightforward. One approach is shown in Figure 1. 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 4 for a 16-byte line size cache, assuming one word is read from off chip memory at a time.

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.

3.2 Using a configurable line size cache

A configurable cache might be used as follows in an embedded system design flow. 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. Thus, the cache configuration would only occur once, during system initialization.

3.3 Energy computation

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 caused by the cache miss. We assume a write back policy is used. We computed the energy due to dynamic power consumption as follows:

$$energy_dynamic = cache_hits * energy_hit + cache_misses * energy_miss$$

$$energy_miss = energy_offchip_access + energy_uP_stall + energy_cache_line_fill$$

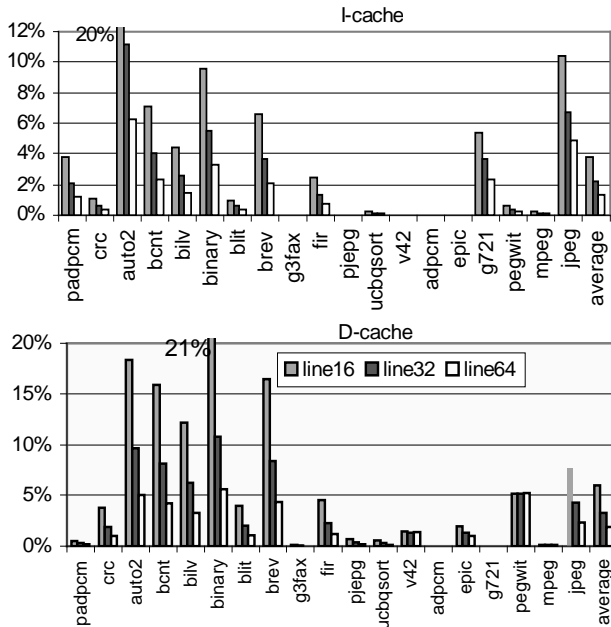


Figure 2: Miss rates of four way instruction (top) and data (bottom) caches for 16, 32 and 64 byte line sizes.

We obtained values for *cache_hits* and *cache_misses* by executing our benchmarks on SimpleScalar [2]. *Energy_hit* is the energy per access to the cache, which we used CACTI to compute using a 0.18 micron technology (we are currently creating an actual layout in a 0.18 micron technology). We estimate *energy_uP_stall* as 20% of the energy of an active microprocessor, a number we determined after looking at the stall power of several microprocessors. The *energy_cache_line_fill* is the energy to write an entire line to the cache. For a four-way set associative cache, only one way is accessed during such a write. The *energy_offchip_access* is the energy to access off-chip memory. We used the low-power 64-Mbit SDRAM manufactured by Samsung (model K4S643233E) as a reference, working at 2.5 V and 55 mA.

4. Experiments

We simulated several benchmarks with various line sizes, including embedded systems programs from Motorola’s Powerstone suite [9] (*padpcm*, *crc*, *auto2*, *bcnt*, *bilv*, *binary*, *blit*, *brev*, *g3fax*, *fir*, *pjpeg*, *ucbsort*, *v42*) and MediaBench [7] (*adpcm*, *epic*, *jpeg*, *mpeg2*, *pegwit*, *g721*, *art*), and some programs from Spec 2000 [5] (*mcf*, *parser*, *vpr*). We used the sample test vectors that came with each benchmark as program stimuli. We consider both a four-way set associative cache, and a direct mapped cache, and considered line sizes of 16, 32 and 64 bytes. All caches considered had a total size of 8 Kbytes.

4.1 Four-way set-associative cache

Figure 2 shows the miss rates for the benchmarks using a four-way set-associative instruction and data caches. We see in some programs that a small line size yields a much higher miss

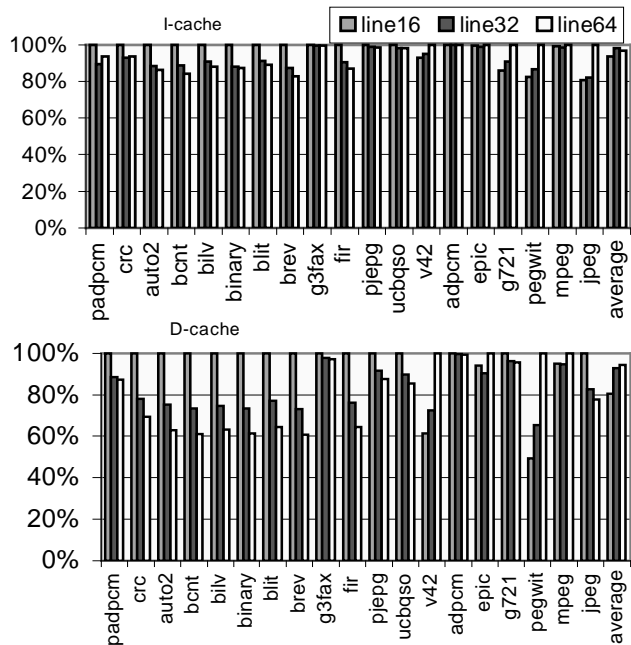


Figure 3: Normalized four way set associative instruction (top) and data (bottom) cache energy savings for 16, 32 and 64 byte line sizes (normalized by setting the configuration with highest energy consumption to 100%).

rate than a larger line size, in which case smaller line size will likely result in higher energy. In other programs, 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 many cases. In terms of miss rates, 15% is extremely high.

Figure 3 shows the energy results for a four-way set-associative instruction cache, for configurations of 16, 32 and 64-byte line size. The energy values are normalized, setting whatever configuration gave the highest energy for each example as 100%, so that the energies of the other two configurations show the savings compared to that highest configuration. We see that for most benchmarks, a line size of 64 bytes yields the least energy. However, several benchmarks, like *v42*, *g721*, *pegwit* and *jpeg*, yield the least energy at a line size of 16 bytes. The energy differences are 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.

Figure 3 shows the results for the data cache. We see some differences in best line size between instruction and data cache. We notice 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 likely because there is spatial and temporal locality varies more greatly for data access than instruction access – our analyses of the benchmarks show that about 70% of the execution time is spent in about 5% of the instruction code, resulting in high spatial and temporal instruction locality.

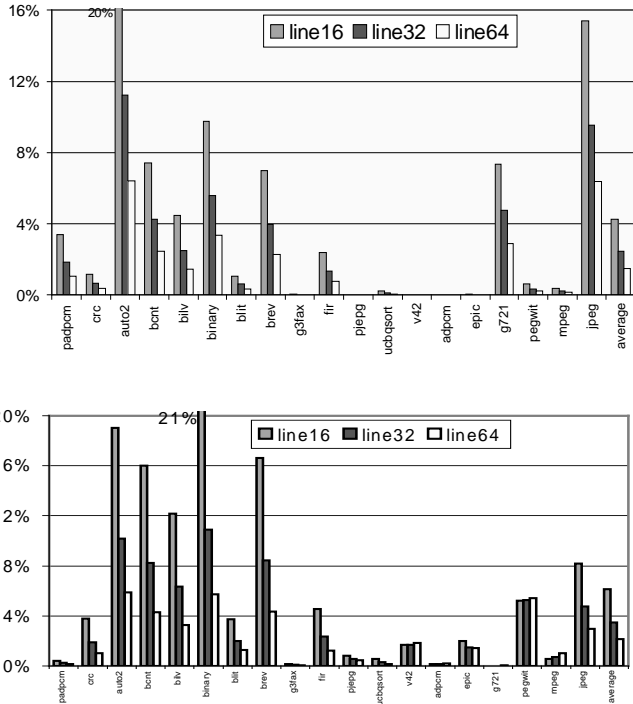


Figure 4: Miss rate of direct mapped instruction cache (top) and data cache (bottom) for 16, 32 and 64 byte line size.

We can see why a line size of 32 is so popular in typical processors that don't have a configurable line size (Table 1). Although 32 bytes is not the best line size on average or the best for any of the benchmarks, we see that a line size of 32 bytes does behave the least erratically. Sizes of 16 bytes and 64 bytes are sometimes the best, but sometimes much worse. 32 bytes is usually somewhere in between. 32 bytes is clearly a compromise. Being able to choose instead either 16 or 64 bytes is clearly superior on a case-by-case basis.

4.2 Direct mapped cache

Figure 4 shows the miss rates for direct mapped instruction and data caches. Figure 5 shows the normalized energy savings for those caches. We see that the line size becomes even more critical for direct mapped caches – caches that are extremely popular in embedded systems due in part to their low power per access. The differences in miss rates among line sizes are even more pronounced than before. We see a nearly 60% energy difference in some cases of the data cache.

4.3 Overhead of configurability

The overhead of cache line size configuration is negligible. From Figure 1, we can see 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.

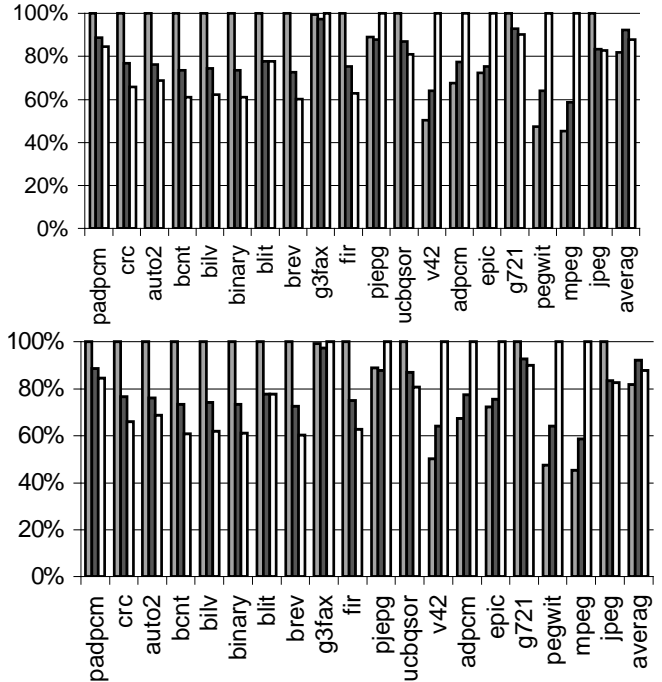


Figure 5: Direct mapped instruction (top) and data (bottom) cache energy savings for 16, 32 and 64 byte line sizes (normalized by setting the configuration with highest energy to 100%).

4.4 Average savings through tuning

We compared the average savings that tuning a cache's configurable line size to a program would yield compared to a fixed line size of 32 bytes. Figure 6 shows average energy savings of a configurable line size cache compared with a fixed 32-byte line size cache. Figure 7 shows the savings compared with a 16-byte line size. Compared to a fixed 32-byte line size, tuning yields reasonable average improvements, over 10% for a direct mapped data cache. Compared to a fixed 16-byte line size, improvements reach almost 20%. Caches may contribute to about 50% of an embedded processor's total power [8][12], and thus savings in memory access power can be very quite significant to overall power savings.

5. Conclusions

We have shown that tuning a cache's line size to a particular program is an extremely effective method of reducing memory access energy for embedded systems. Choosing among 16, 32 or 64 byte line sizes can by itself impact memory access energy by nearly 60%. Adding such configurability to a cache architecture is straightforward, and selecting the best configuration can be done fairly simply by embedded system designers.

If one compares the miss rates across four-way and direct-mapped caches, one sees that choosing the best associativity for a given program is also critical. For this reason, we are currently working on a configurable cache architecture that not only has a configurable line size, but also a configurable number of ways (while maintaining the same total size). We

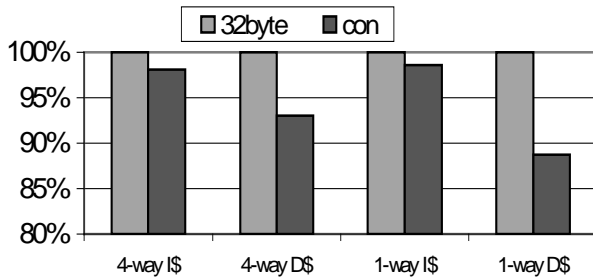


Figure 6: Average energy savings of a configurable cache compared with a fixed 32-byte line size cache.

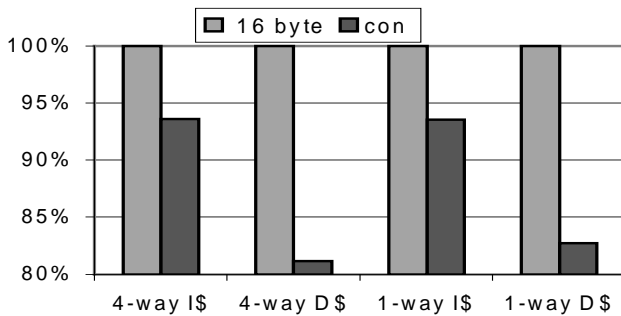


Figure 7: Average energy savings of a configurable cache compared with a fixed 32-byte line size cache.

are also nearly finished with a layout of the configurable cache in a 0.18 micron CMOS technology, from which we will be deriving actual delay and power values.

6. Acknowledgments

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

References

- [1] L. Benini, G. De Micheli, E. Macii, D. Scjuto, C. Silvano. "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems," GLS-VLSI-97:IEEE 7th Great Lake Symposium on VLSI, pp. 77-82: Urbana-Champaign: IL: March 1997.
- [2] D. Burger and T.M. Austin. The SimpleScalar Tool Set, Version 2.0. University of Wisconsin-Madison Computer Sciences Department Technical Report #1342. June, 1997
- [3] J. L. Hennessy and D. A. Patterson: Computer architecture Quantitative Approach, 2nd Edition, Morgan-Kaufmann Publishing Co., Menlo Park, CA. 1996
- [4] <http://www.mips.com>
- [5] <http://www.specbench.org/osg/cpu2000/>
- [6] K. Inoue, K. Kai, A High-Performance/Low-Power On-Chip Memory-Path Architecture with Variable Cache-Line Size, IEICE Trans. Electron. Vol. E83-CV No. 11, Nov. 2000
- [7] 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, MICRO 1997
- [8] L. Lee., B. Moyer, J. Arends. Low-Cost Embedded Program Loop Caching – Revisited. University of Michigan Technical Report CSE-TR-411-99, 1999. Applications. International Symposium on System-Level Synthesis, pp. 74-81, 1997.
- [9] A. Malik, B. Moyer and D. Cermak. "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility." International Symposium on Low Power Electronics and Design. June. 2000.
- [10] D. Nicolaescu, X. Ji, A. Veidenbaum, A. Nicolau and R. Gupta, "Compiler-directed Cache Line Size Adaptivity", IMS, 2000.
- [11] Glen Reinman and Norman P. Jouppi, CACTI 2.0: An Integrated Cache Timing and Power Model. WRL Research Report 2000/7
- [12] Simon Segars, Low Power Design Techniques for Microprocessors. ISSCC2001
- [13] M. R. Stan, W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 3, No. 1, pp. 49-58, March 1995
- [14] F. Vahid and T. Givargis. Platform Tuning for Embedded Systems Design. IEEE Computer, Vol. 34, No. 3, pp. 112-114, March 2001.
- [15] A. Veidenbaum, W. Tang, R. Gupta, "Adapting Cache Line Size to Application Behavior." ICS'99 Rhodes Greece
- [16] N.H.E. Weste and K. Eshraghian, "Principles of CMOS VLSI Design, A System Perspective" Second Edition. Addison-Wesley. 1993 pp. 233-235.
- [17] E. Witchel and K. Asannovic, "The Span Cache: Software Controlled Tag Checks and Cache Line Size", 28th ISCA, Sweden, 2001
- [18] Semiconductor Industry Association. International Technology Roadmap for Semiconductors: 1999 edition. Austin, TX: International SEMATECH, 1999.