Improving SDRAM Access Energy Efficiency for Low-Power Embedded Systems

JELENA TRAJKOVIC, ALEXANDER V. VEIDENBAUM, AND ARUN KEJARIWAL
University of California, Irvine

DRAM (dynamic random-access memory) energy consumption in low-power embedded systems can be very high, exceeding that of the data cache or even that of the processor. This paper presents and evaluates a scheme for reducing the energy consumption of SDRAM (synchronous DRAM) memory access by a combination of techniques that take advantage of SDRAM energy efficiencies in bank and row access. This is achieved by using small, cachelike structures in the memory controller to prefetch an additional cache block(s) on SDRAM reads and to combine block writes to the same SDRAM row. The results quantify the SDRAM energy consumption of MiBench applications and demonstrate significant savings in SDRAM energy consumption, 23%, on average, and reduction in the energy-delay product, 44%, on average. The approach also improves performance: the CPI is reduced by 26%, on average.

Categories and Subject Descriptors: C.3 [**Application-based Systems**]: Embedded Systems— *SDRAM design*; B.3.2 [**Memory Structures**]: Design Styles

General Terms: Design, Performance

Additional Key Words and Phrases: SDRAM, fetch buffer, write-combining buffer, embedded processors and low power

1. INTRODUCTION

Many embedded applications are memory intensive, especially multimedia applications. In such applications, main memoryaccess constitutes a significant

portion of the overall energy consumption [Barr and Asanovic 2003; Kim et al. 2003]. Thus in order to minimize the latter, it is critical to reduce the (DRAM) main memory energy consumption. Most of the research in the area of low-power systems has focused on reducing the energy consumption of the processor. On the contrary, this research studies the energy consumption of both the processor and the DRAM main memory. It investigates an architectural approach for reducing the main memory energy consumption without sacrificing overall performance.

Research in the area of low-power embedded processors, that is, processors that are optimized for low energy consumption and cost and do not aim for high performance, has identified the data and instruction caches [Wilkes 1965; Smith 1982; 1991] as the major consumers of power in the processor. These low-cost/low-power systems use only level-one (L1) caches and do not deploy a level-2 cache (which is standard in high-performance processors). Thus, much of the prior research to reduce the embedded processor energy consumption has focused on L1 caches.

However a main memory composed of commodity DRAM consumes orders of magnitude more energy per access than a cache access (see Tables II and III). Even though an L1 cache significantly reduces the total number of access to main memory, the total energy expended in accessing memory can be very large. For instance, Barr and Asanovic [2003] showed that DRAM consumes more energy than the processor in a PDA.

As will be shown below, in some embedded applications the total energy of main memory accesses is an order of magnitude higher than the total data cache energy consumption. Thus, it is very important to both study and optimize DRAM access for energy consumption. Synchronous DRAM (SDRAM) memory is one of the main types of DRAM used today. This research will, therefore, focus on a system consisting of a low-power processor and a single SDRAM memory IC (integrated circuit) to study main memory energy consumption. Some of the techniques previously proposed for cache optimization can be adapted and extended for this purpose.

The energy of an SDRAM access can be divided into two main components: the energy of a bank/row activation (activate-precharge pair) and the energy of a read or write access to an active bank/row. The activate-precharge pair consumes over 60% of the total memory access energy (as per the manufacturer's data [Micron]), as shown in Figure 1. The figure quantifies energy components of a 64 MB SDRAM memory [MicronDataSheet]. These were obtained using Micron's System Power Calculator [Micron]. Each bar on the graph shows the activate-precharge and write components of the total energy. The figure shows the total energy for writing 16 bytes of data (one cache line), two separate 16-byte writes, and two 16-byte writes "combined" in one activation-precharge. For separate writes, we assume that a bank/row is precharged after each access. This mode is chosen because of the following: the energy of an "idle" state after precharge is significantly smaller (6.6 mW) than the energy of an "idle" state after activate command (82.5 mW). The figure also presents data for three and four accesses, performed separately or combined. The read components of the total energy follow a similar pattern.
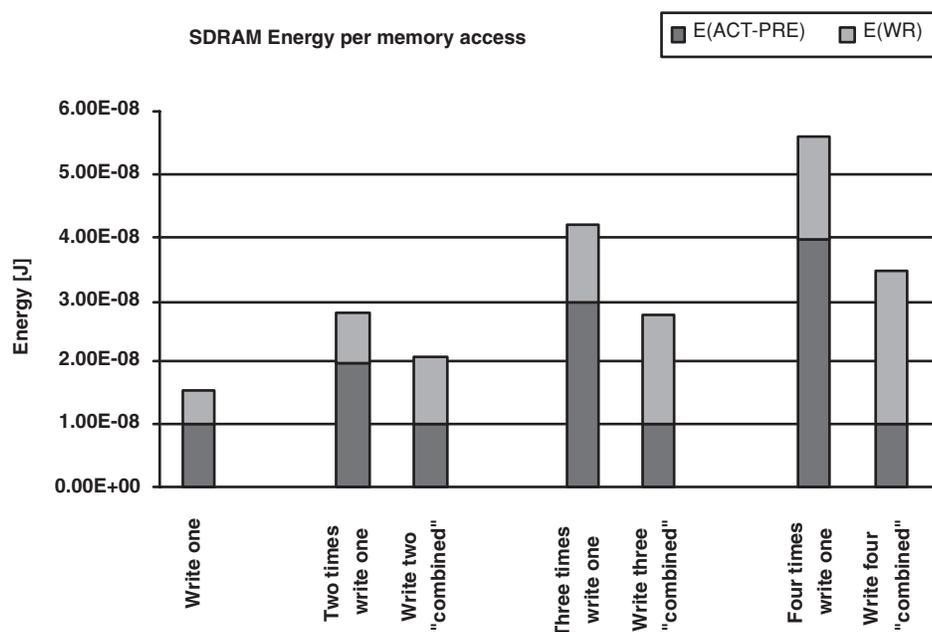
Fig. 1.   Energy per memory access.

The SDRAM organization allows the bank/row to be left "on" after an access, which permits additional read/write access to the corresponding bank/row without incurring the activate/precharge cost on each access. Reading or writing twice the amount of data within the same activate-precharge cycle does not double the energy consumption, but only increases it by approximately 32%. This property of SDRAM is the key to the optimization techniques proposed in this paper. However, if a bank/row is left in the "on" state, but not accessed, extra energy is consumed as compared to the "idle" SDRAM state when little energy is consumed. The SDRAM can even be put into a "sleep mode" (clock not running), where only static energy, defined as the energy consumed when no signal activity occurs (i.e., no switching), is consumed. Initiating a new access to a "sleeping" SDRAM incurs additional delays. Thus any access energy optimization needs to take these different SDRAM characteristics into account. The details of SDRAM energy consumption will be further discussed in Section 3. In this work, we take into account static and dynamic energy[1] consumption of SDRAM.

Even simple embedded processors use a data cache and access memory only on cache misses or write-backs. A write-back data cache is typically used to reduce the write traffic and to aggregate writes into blocks. Thus, the memory is read/written in cache blocks (lines), one block at a time. However, reading/writing multiple memory blocks (from the DRAM) at a time would be more energy efficient within a single activate-precharge pair, but CPU cache controllers do not do that. In such a case, it may be very advantageous to change

---

[1]Dynamic energy is defined as the energy consumed when signals are active (i.e., switching).

the architecture to perform multiblock reads and writes from memory. We assume that we cannot change the processor architecture to accomplish this and, therefore, will add such a capability to the memory controller. Arguably, the architectural extension will work even better if integrated into the processor.

As will be shown in Section 5.2, the aforementioned new capability will lead to significant energy savings. Multiblock memory reads are similar to hardware prefetching, especially to stream buffers [Jouppi 1990], which have been shown to work very well. Multiblock writes are similar to write combining [Smith 1982], but, in our case, they combine multiple cache blocks (beyond individual words). These techniques in our case target energy reduction rather than performance improvement. They are performed in our proposal in the memory controller and do not involve the CPU cache. One can argue that an increase in the cache line size can achieve the same reduction in energy consumption. However, in Section 5.2.4, we show that doubling the line is not competitive with the techniques proposed in this work. In the rest of this paper, we will use the terms fetching/prefetching and write combining for describing DRAM access in our system.

Accessing multiple lines outside of the processor requires intermediate storage. This research proposes to use a small amount of such storage in the memory controller. Additional lines will be fetched into this storage on each read. Writes to the same SDRAM row will be buffered first and combined into a multiple-line SDRAM write whenever possible.

The main contribution of this article is adapting both prefetching and write combining for SDRAM energy reduction, in particular, combining multiple cache line writes to the same SDRAM row. A second goal of this research is to minimize performance degradation or even to improve execution time while saving energy. The small buffers (mentioned above) used for prefetching/combining act as a "memory cache" and can significantly improve read performance, something similar to store-to-load forwarding [Hennessy and Patterson 2006]. "Regular" prefetching can sometimes degrade execution time by interfering with "regular" memory access without supplying useful data. It can also (1) "pollute" the cache; (2) require additional storage and CPU complexity, if not prefetched in the cache. Prefetching at the memory controller avoids most of these problems by overlapping a (reduced) delay of reading additional lines with the transfer of the missed line to the CPU cache. Write buffering at the memory enables the processor to continue execution and not wait for the SDRAM write access to finish.

Numerous potential cache organizations, sometimes even configurable on a per-page basis (e.g., ARM processors), especially with respect to write policy and write buffering have been proposed in the past. For instance, a write-through cache (or a write-back cache) that does not perform write-allocate may employ a write-combining buffer that attempts to assemble a block from individual writes rather than writing single words to memory. The reader is referred to Zhang et al. [2005] for a list of popular embedded processors and their (different) cache configurations. To be practical we had to choose one cache organization to evaluate. Our choice, a write-back write-allocate L1 data cache [Smith 1982] typically has a better performance than most other alternative organizations.

This organization does not require post-L1 write buffering except perhaps to perform a read miss fetch before completing the write, i.e., for performance optimization. Such a "victim" buffer [Hennessy and Patterson 2006] does not perform multiple-line writes to memory and is thus orthogonal to our approach (although it will provide some performance improvement by reducing read miss latency). Therefore, the baseline processor organization assumed in this article has no post-L1 cache write buffer.

The rest of the article is organized as follows: Section 2 presents related work. Section 3 describes the SDRAM energy components for read and write access. Section 4 presents architectural modifications and describes the energy-saving techniques of our approach. Experimental results demonstrating the benefits of the approach are presented in Section 5. Conclusions are presented in Section 6.

2. RELATED WORK

There is a large body of prior work on prefetching, write buffers, and word-level write combining buffers that is briefly summarized below. Techniques for reducing energy consumption in main memory are also described. Many architectural approaches for reducing energy consumption in embedded processors have been proposed. To the best of our knowledge architectural solutions for SDRAM energy reduction have not been proposed so far. For a software-based solution see Kim et al. [2003], for instance.

Numerous prefetching algorithms have been proposed for fetching cache lines based on history driven prediction of future memory accesses. For instance, a stream buffer [Jouppi 1990] prefetches consecutive lines triggered by a miss. Prefetching schemes differ mainly in how they predict which block to fetch rather than in how many blocks to fetch. A variable-length prefetching mechanism proposed in Dahlgren et al. [1993] changes the number of blocks prefetched based on their utilization. Macroblocks were proposed as a fixed-size superline in Johnson et al. [1997]. A detection mechanism is used to identify the use of multiple lines within a macroblock and fetch the entire macroblock on future misses. However, superline detection requires a large hardware predictor to keep track of access history. An "adaptive line size" (ALS) cache was proposed for predicting the line size for the next fetch upon a line replacement in Veidenbaum et al. [1999]. Solihin et al. [2002] describe a method for prefetching based on correlation of memory addresses with an engine that resides in L2 cache that gives significant performance gain even with a low hit rate in the prefetch buffer. Since the engine is placed in L2 cache, it misses potential correlations that may lead to better prefetching. All of the prior schemes (except stream buffers) described above require a complex mechanism for detection and prediction of the next fetch size. In contrast, our work uses a simple mechanism and requires little hardware support. None of the prefetching proposals targets energy reduction.

Adding a write buffer is a common optimization that allows the processor to continue execution as soon as the data is written to the buffer. Thus, the major part of memory update is done independent of the processor execution. Smith [1979] gives a study of performance impact of write buffer and write merging.

Jouppi discusses write buffers and write caches and evaluates their performance for both write-back and write-through schemes [Jouppi 1993; Farkas et al. 1994]. Merging is introduced to improve performance of write buffers. This approach combines incoming write request with requests already residing in the write buffer, resulting in a more efficient bus and memory usage. The techniques has been applied to architectures like Alpha 21164, UltraSparc III, StrongARM, MIPS R4300i, and XScale. XScale allows combining requests for four word-aligned addresses. None of these attempt to combine multiple line write-backs to the memory.

Low-power mode is present in most state-of-the-art DRAMs. A significant amount of energy can be saved by setting as many memory chips as possible to sleep mode [Lebeck et al. 2000]. Likewise, efforts have been made in reducing memory energy consumption based on different compression techniques. For instance, Abali and Franke [2000] describe and evaluate a computer system that supports hardware main memory compression. Likewise, Ekman and Stenström [2005] proposed a main-memory compression scheme. As the compression ratio of applications dynamically changes, so does the real memory size that is managed by the operating system (OS). For such systems, OS changes are necessary to support main memory compression. These changes include the management of the free pages pool as a function of the physical memory utilization and the effective compression ratio, coupled with zeroing pages at free time rather than at allocation time.

Kim et al. [2003] identify successive memory accesses to different rows and banks as a source for increased latency and energy consumption in memory. They use block-based layouts instead of a traditional one and determine the best block size such that the number of requests to different rows or banks is minimized. Rixner et al. [2000] proposed a memory-scheduling technique for efficient use of the memory bandwidth. Similarly, Goossens et al. [2004] proposed a technique for efficient exploitation of the bandwidth between the SDRAM and the on-chip interconnect. However, the above do not address the issue of minimization of power consumption of the SDRAM.

3. SDRAM ENERGY COMPONENTS

Let us first identify potential sources of energy saving in SDRAM memories that are used in embedded devices. In order to access data from a SDRAM, a row in a particular bank has to be activated. After activation and a specified delay, a read or a write is performed. When the access is completed, a row precharge operation is performed. Also, if access to data in a different row has to be performed, the current row needs to be precharged before the new row is activated. The total energy of an access consists of two main components: *energy consumed by activate-precharge pair* and *read or write access energy*.

Figure 2 shows the current profile for the write (or read) operation in such a memory (reproduced from [Micron][2]). The first large peak on the left side of the graph corresponds to the activation command. The middle plateau corresponds to writing four words of data. Finally, a small peak can be noticed for

[2]Used with permission from ©2001 Micron Technology, Inc. All Rights Reserved.
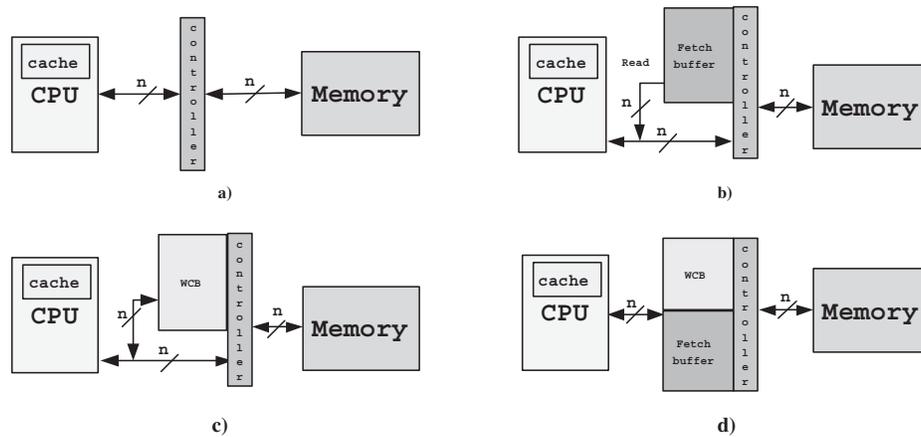
Fig. 3.   Memory subsystem architectures.

combine *any* two cache block/line writes to the same SDRAM row. To the best of our knowledge, all previous work considers only combining of writes with adjacent addresses within the same cache line, which is used primarily with write-through caches. The write buffer also needs to be small and simple for the same reasons, as discussed above for prefetching.

Let us first consider write and read combining separately so as to understand the benefits and requirements of each of them. Next, the "combined" approach will be investigated and the best mechanism is selected. In each case, the sizes of buffers are studied as part of this research. For reasons that will become more clear after the architecture of each separate buffer is studied, the combined approach will actually use separate buffers as opposed to a single "mini-L2 cache" in the memory interface. While conceptually the same, the implementations are quite different with separate buffers having an advantage.

4.1 Read Combining

The goal of read combining is to perform multiline SDRAM reads. Since there is only one read miss at any given time in an embedded in-order single-issue processor, there is nothing to combine it with. Thus, the only way to read-combine is to generate an additional address speculatively via prediction. This is what the aforementioned sequential prefetching mechanisms do. The difference is that our prefetching is aimed at main memory energy reduction.

It is possible to prefetch nonadjacent lines within a same row, in a way similar to write combining. This would, however, require a very sophisticated address predictor that would be both large and complex (see Kumar and Wilkerson [1998]). This is why only simple, sequential prefetching is considered here. It is also referred to as *read combining*.

The memory subsystem architecture for read combining is shown in Figure 3b. It fetches N additional cache lines on a read miss. The lines are stored in a fetch buffer (FB): a small, cachelike structure with a tag on each entry. Each cache read miss is checked against the FB. On a hit, the line from FB is sent to the CPU cache. On FB miss, the line is read from the SDRAM
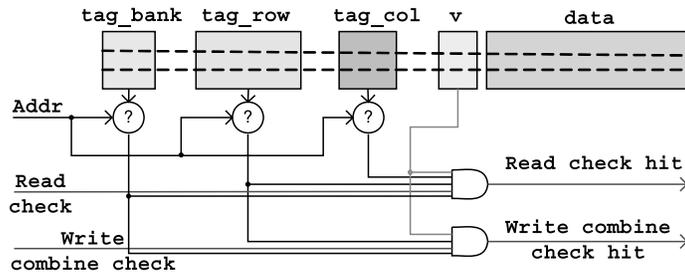
Fig. 4. Architecture of write-combine buffer for N = 1.

together with N additional lines which are all stored in the FB. The missed line is read first and sent to the CPU cache. All $N + 1$ read accesses are performed in the same activate-precharge cycle. The rest of this paper will primarily deal with $N = 1$. Results for $N = 2$ and 3 can be found in Trajkovic and Veidenbaum [2004], where it is shown that a larger N usually increases energy consumption because many prefetched lines are not used.

As will be shown, a small, fully associative FB is sufficient to achieve significant energy reduction. In addition, the performance is improved because of FB's prefetching effect and its lower access latency compared to the SDRAM.

4.2 Write Combining

The memory controller for write combining is shown in Figure 3c. Figure 4 shows the write-combining buffer (WCB) architecture for combining two write requests. Each entry consists of a split tag, a valid bit, and data storage for one cache line. Tag bits are divided into three groups: bits that determine the bank address in the memory (*tag_bank*), bits that determine the row address in the memory (*tag_row*), and the remaining tag bits that are part of the column address (*tag_col*).

The buffer is expected to be very small and thus full associativity is easily implementable. An address of an incoming cache line write request is checked against all tag_bank/tag_row entries. LRU (least recently used) or pseudo-LRU replacement is used.

Figure 5 describes write request handling for the buffer that combines two writes. If an incoming write hits into the WCB, and the matching entry is valid, both new and the matched data are written to the memory. Once the writing is finished, the "matched" entry is freed (valid bit = 0), so that it can be used for the new incoming write request. Note that in case of a hit ("Write combine check hit" in the figure) the incoming write is not stored into the WCB. If an incoming write misses in the WCB, the data and the address corresponding to that write request are stored into WCB to be potentially combined in the future. The write miss can cause a replacement: a single line from the entry selected for the replacement is written back in the SDRAM. This architecture can be extended to combine more than two accesses. To combine $N + 1$ writes, N tag_col sub-tags, N valid bits, and N data store blocks are stored with each tag_bank/tag_row entry. The entry contains a counter to show how many writes to this SDRAM row are already present. An incoming write causes a write to
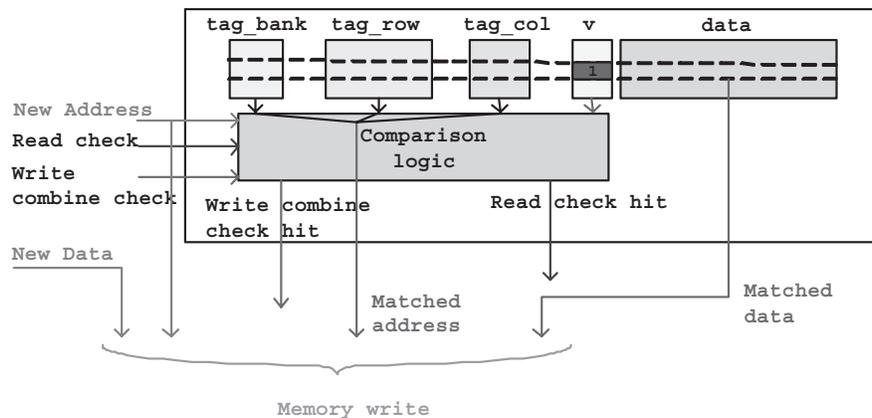
Fig. 5.   Write request handling (write-combine buffer for N = 1).

the memory on a hit if the entry counter has a value of N. On replacement, less than N entries may be written to the memory, as specified by the counter value and the valid bits.

To summarize, the WCB differs from the traditional write buffer or even a coalescing write buffer, because it can merge cache block writes that is anywhere in a given SDRAM row. As a result it writes data to the memory in units of $N+1$ cache blocks or less. The goal is to write $N+1$ entries as often as possible. A traditional write buffer, on the other hand, can only coalesce individual words (or subwords) *within a cache line* and writes data to memory when the memory is not busy servicing any other read or write request. A major advantage of this new form of write combining is that it cannot incur any energy losses, as will be shown later in Figure 9 in Section 5. The total number of writes is the same as in the baseline case but those writes are potentially grouped in a different way. In addition, the WCB reduces the processor CPI by allowing the CPU to continue execution as soon as data are written to the WCB (as opposed to waiting for the SDRAM write to complete).

The presence of the WCB creates a coherence problem on reads just like in case of store-to-load forwarding [Hennessy and Patterson 2006]. It is solved as follows: every read address is checked against the full line address (i.e., tag_bank, tag_row and tag_col bits in Figure 4) of every line in the WCB. A read hit implies the needed data is in the WCB and the matched line is sent to the CPU cache. This results in read miss latency reduction. In the case when a read access misses in the WCB, the miss latency overhead corresponds to the time required to perform address comparison. Intuitively, one can expect that this overhead would result in increased CPI. As our results show (see Figure 11 in Section 5) the increase in miss latency does not cause performance degradation.

4.3 Read- and Write-Combining

Both write and read combining have their own advantages. They are largely independent of each other and thus can be deployed together for an additive energy reduction as well as performance improvement. However, the question

is what is the best architecture to perform the read and the write combining at the same time.

For this, we propose a novel architecture as shown in Figure 3d. It integrates the separate *fetch buffer (FB)* and *write-combine buffer (WCB)*. While a single, cachelike structure can be designed, it will have the following two major disadvantages:

1. It will likely require that N, the number of cache lines to combine, be the same for reads and writes. As will be shown later in Section 5, this is not desirable from the perspective of minimizing the energy consumption (see Figure 12).
2. More importantly, it may significantly increase hardware complexity. The merged structure would need to have the same architecture as the WCB architecture. The split-tag organization is not required for read combining as it would increase required storage and per access energy. Searching the coalesced structure would be different for read and write requests, which would require additional hardware complexity.

Moreover, the replacement decisions would require more complex hardware, which could potentially increase access time. In addition, there will be interference and replacements of write lines by read prefetches and vice versa in the case a single, cachelike structure.

Merging the WCB and FB designs is less difficult, since each will continue to operate independently and has its own control. Thus, the write-combining operation in the WCB remains the same and the read-combining (prefetching) operation in the FB remains the same. Recall that WCB was already checked on each read miss and could supply data to the CPU cache. However, there is one change that is required for the merged organization. Additional coherency checks have to be performed between writes and prefetches. First, prefetched data can be invalidated by an incoming write from the CPU cache. Second, there is no point in prefetching lines already in the write-combining buffer.

Briefly, the solution is twofold. First, any incoming data cache write request is checked against both the FB and the WCB (in parallel). A matching FB entry is invalidated. Second, every prefetch address is checked against the WCB first, then sent to the DRAM only if there was no match. The small size of the WCB guarantees that this additional fully associative search has low energy overhead and does not cause slowdown. The coherency algorithm that the controller implements can be found in Trajkovic and Veidenbaum [2004].

The only potential drawback of the combined approach is overutilizing the limited memory bandwidth. A combination of write combining and read prefetching can use up all of the available bandwidth. A read miss may thus be delayed and cause a slowdown. The evaluation of access combining is presented in the next section. It will show that the energy and/or performance loss can be avoided in almost all cases. When it does happen, it can be minimized by a proper choice of architectural parameters.

5. EVALUATION METHODOLOGY

The system modeled in this paper consists of an embedded, in-order processor and a single, large SDRAM memory chip. One can think of a mobile phone as an example of such a system. The processor is a single-issue, 32-bit embedded processor resembling the MIPS R3000 processor [Kane 1988]. It has a 8 KB, four-way set associative instruction and data caches with a 16-byte line and a two cycle latency. Data cache implements a *write-allocate, write-back* policy. The CPU operating frequency is 400 MHz. The CPU memory bus is a 100 MHz, 32-bit bus. The baseline cache miss latency is 36 processor cycles for the first word to arrive and an additional four processor cycles for each consecutive word.

The main memory with a modified controller has a latency of 40 processor cycles for the delivery of the first word, and four cycles for each additional consecutive word. Both baseline and modified architectures use the same SDRAM (see data sheet [MicronDataSheet]). The extra four cycles (10 ns) in the access time to modified memory are because of FB and WCB access delays. The SDRAM clock rate is 100 MHz (speed grade -6).

The timing evaluation is performed using the SimpleScalar 3.0/Wattch simulator [Burger and Austin 1997; Wattch] executing PISA binaries. SimpleScalar's bus and memory model are modified to model the bus delays and to match this architecture. Both FB and WCB are fully associative, with 16-byte lines, and a latency of twelve processor cycles (from CACTI calculations). The WCB can be configured to store N lines per entry. The performance is evaluated using benchmarks from the MiBench suite [Guthaus et al. 2001], which models applications from a large variety of embedded domains. A brief description of the benchmarks is given in Table I. All benchmarks are simulated using "large" input sets.

5.1 Energy Calculation

Dynamic energy consumption of the cache, WCB, and FB is modeled using modified CACTI 3.2 [Shivakumar and Jouppi 1990] for 0.18-μm technology. One of the main changes are in the sense amplifier energy model, which was overestimated in the original model. Main memory energy is computed using Micron's system power calculator [Micron]. The energy per access for the memory (E_{mem}), L1 data cache (E_{dl1}), fetch buffer (E_{FB}), and the write-combining buffer (E_{WCB}) is given in Table II. Note that the memory is accessed in a single 16- or 32-byte transaction, in "burst" mode.

The FB and WCB sizes were limited to avoid overhead and reduce cost. As a result, the FB consumes 0.75% and WCB consumes 4% of the data cache energy when both buffers are at full capacity (worst case scenario) and assuming a 0.18-μm process technology. The energy overhead of FB and WCB is *included* in memory energy.

Total memory energy is computed using the given per-access value and the number of accesses counted during the simulation. Total L1 data cache energy is obtained by simulation via Wattch. It is a function of per-access energy of tag and data access, as well as the number of accesses because of cache hits, misses

Table I. Description of Benchmarks in MiBench Suite.

Class	Benchmark	Lines of code	Language	Description
Automotive	qsort	100	C	Qsort algorithm
	susan	2122	C	Smallest univalue segment assimilating nucleus (image processing)
Consumer	jpeg	33717	C	Jpeg compression
	lame	18612	C	LAME Ain't an MP3 encoder
	tiff-v3.5.4	45066	C	Tag(ged) image file format
Networking	dijkstra	351	C	Dijkstra's algorithm
	patricia	599	C	Trie implementation
Office	ghostscript	197528	C	Ghostscript 5.0
Security	blowfish	2302	C	A keyed, symmetric block cipher
	rijandel	1788	C	Advanced encryption standard
	sha	269	C	Secure hash algorithm
Telecom	FFT	469	C	Fast fourier transform
	gsm	5473	C	GSM 06.10 13 kbit/s RPE/LTP speech compression

Table II. Energy per Access for the Memory, L1 Data
Cache, Fetch Buffer, and Write-Combining Buffer

E_{mem} (16 B)	15.3 nJ
E_{dl1}	307.4 pJ
E_{FB}	152.4 pJ (tag: 98.6 pJ, data: 53.8 pJ)
E_{WCB}	111.5 pJ (tag: 63.6 pJ, data: 47.9 pJ)

and writebacks. Similarly, the total energy of fetch and write combining buffer is obtained by simulations, where total number of tag and data accesses have been counted.

5.2 Results

The impact of the proposed architecture is evaluated by comparing the memory energy consumption, energy-delay product, and CPI relative to the baseline configuration. The following legend is used:

— "FetchN_M" for read fetch of N lines with a buffer of M lines ($N - 1$ lines are prefetched);
— "WCB_P" for write combining of two accesses with a buffer of P lines;
— "WCB_PxQ" for write-combining of $(Q + 1)$ accesses with a buffer of P entries x Q lines;
— "FetchN_M+WCB_PxQ" for a hybrid configuration.

Table III shows memory energy per benchmark relative to the data cache memory consumption for the baseline model. These large ratios were observed inspite of the fact that the average cache miss rate for the entire suite was 3.5%. Montagnaro et al. [1996] showed that the data cache consumes 16% of the overall processor energy. For MiBench benchmarks the main memory consumes an average of 2.6 times the energy of the data cache that is quantified. This demonstrates that the DRAM energy optimization is more important than that

Table III. Memory Energy Relative to the Energy of Data
Cache

Benchmark	E_{mem} relative to E_{cache} [%]	Benchmark	E_{mem} relative to E_{cache} [%]
tiff2rgba	1487.34	dijkstra	120.21
tiff2bw	867.42	d_FFT	118.39
tiffmedian	614.28	d_rijndael	72.83
lame	416.22	e_rijndael	70.39
tiffdither	234.03	e_susan	60.50
d_jpeg	219.58	ghostscript	46.90
qsort	187.47	patricia	31.99
c_jpeg	167.82	d_blowfish	30.11
sha	154.80	e_blowfish	30.10
i_FFT	123.39	AVG	265.99
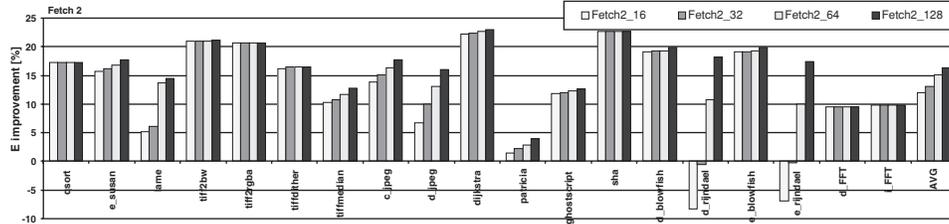
Fig. 6.   Memory energy reduction for read combining for different buffer sizes.

of the data cache. The worst-case difference is $15\times$, which means that the DRAM consumes approximately twice the energy used by the processor.

5.2.1 *Read Prefetch: The Effect of Fetch and Buffer Size.* First, let us evaluate read combining and its effect on memory energy consumption. Figure 6 shows energy reduction for different fetch buffer sizes relative to the baseline configuration. Buffer sizes of 16, 32, 64, and 128 entries are used, fetching two 16-byte blocks. The average memory energy savings are 12–17%. The smallest buffer already obtains a significant reduction, with each doubling of the size producing a small (1–2%) increase. Two benchmarks *d_rijndael* and *e_rijndael* have a noticeable increase in memory energy consumption for a 16-entry FB. For these two benchmarks, accesses to the data in the consecutive cache lines are so far apart in time, that for FB with 16 entries over 50% of prefetched data get replaced before being used. With 32 entries, there are basically no energy increases, making it a good choice. Note that the energy penalty for unnecessary/unused prefetches is 35%, while latency reduction for hit in the FB is over 65%, resulting in significant CPI improvement even for *d_rijndael* and *e_rijndael*.

The energy-delay (ED) product is shown in Figure 7. ED product is reduced by 68%, in the best case (*tiff2rgba*), and by 38%, on average, with buffer size having almost no impact in majority of the benchmarks (i.e., smallest size is good enough). It can be seen, that even the two benchmarks that have energy increase, viz., (*d_rijndael* and *e_rijndael*), obtain significant ED product savings. The energy delay product reduction largely results from an improved
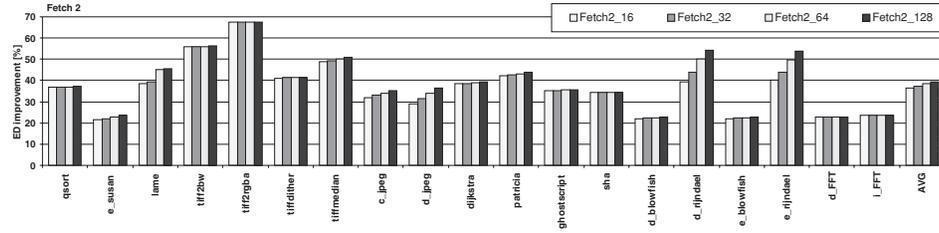
Fig. 7.   Memory ED product reduction for read combining for different buffer sizes.
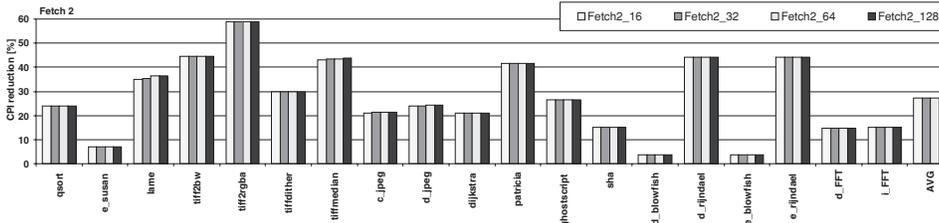

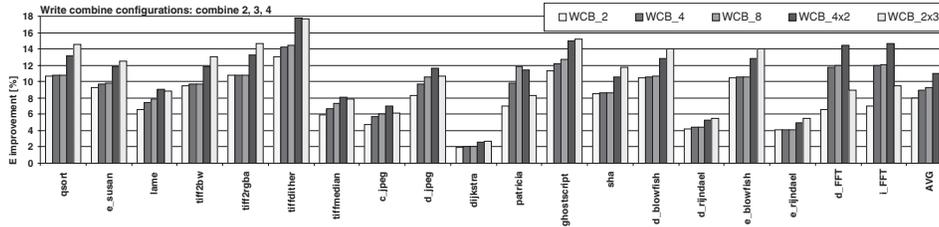Fig. 8.   CPI improvement for read combining for different buffer sizes.

Fig. 9.   Memory energy reduction for different write combining and buffer sizes.

average memory latency. The effect of latency reduction can be seen in the CPI improvement shown in Figure 8. CPI is also insensitive to the buffer size change. Read-combining technique reduces CPI by as much as 59 and by 27%, on average.

If we consider energy as the main factor, the smallest buffer that provides savings with no overhead is one with 32 entries. On the other hand, if we consider ED product, it is the buffer with 16 entries that brings the same savings as the largest buffer in the majority of cases. We also note that only two benchmarks, viz., *d_rijndael* and *e_rijndael*, have significant difference in ED product saving (15%) from a larger buffer size. Therefore, for the combined technique we use a read-combining buffer with 16 entries.

5.2.2   *Write Combining: the Effect of Combining and Buffer Size.*   Figure 9 shows the memory energy reduction achieved via write combining. Buffer sizes of 2, 4, and 8 entries are used, with 2-, 3-, and 4-line combining. The buffer configurations are chosen to have approximately the same size in all cases. On average, per configuration, the improvement ranges from 8 to 11%; it is smaller than for read combining. This is, in part, because of the fact that writes are less
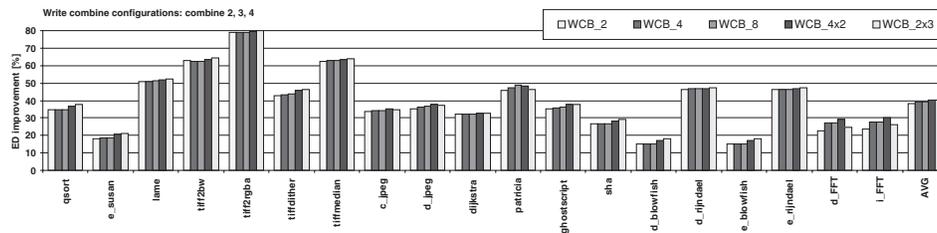
Fig. 10. Memory ED product reduction for different write combining and buffer sizes.
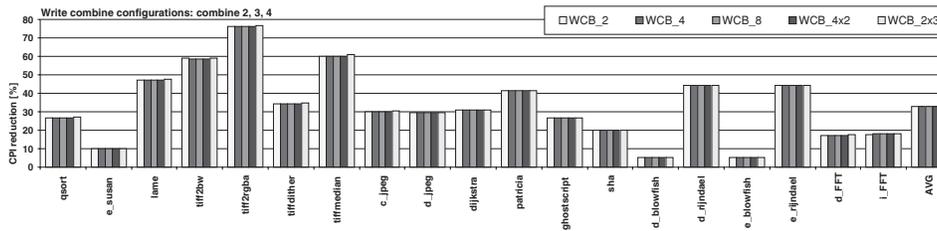

Fig. 11. CPI improvement for different write combining and buffer sizes.

frequent than reads and, in part, because of a smaller buffer size. Buffer size has little impact, (the left three bars), in most cases, but additional energy savings are obtained when combining three or four lines. Write combining achieves up to 80% reduction of ED product (see Figure 10), with a 40% average. CPI savings (see Figure 11) are not affected by size or by configuration. Write combining achieves CPI improvement up to 76%, with a 33% average. The main reason for CPI improvement is a reduction in read-miss penalty on replacement of "dirty" lines; also, reads can hit in the WCB, which helps in achieving higher performance. However, this improvement will disappear if the CPU implements a victim buffer [Hennessy and Patterson 2006].

5.2.3 *Hybrid Configurations.* Figure 12 shows the effect of both write and read combining. The fetch buffer with 16 entries is used together with a write-combining buffer with 8 entries, configured to combine either 2, 3, or 4 writes. The results show that combining three lines is the best configuration. On average 21.5–23.5% energy savings are obtained. From Figures 13 and 14, we see that the difference in ED product and CPI savings for different configurations is not more than 2%. ED product is reduced by a maximum of 71 and by 44%, on average. CPI is improved by up to 56%, with a 26% average.

5.2.4 *Comparison with a Double-Size Cache Line.* One can argue that an increase in cache line size can achieve the same in results/benefits as the techniques proposed in this paper. To address this, the baseline was compared with a cache 32-byte line. Figure 15 shows the data cache energy consumption when using 32-byte cache lines relative to the baseline configuration (16-byte cache line). The cache with doubled line size has the same capacity and, therefore, has one-half less the number of lines. The values for energy are as reported by
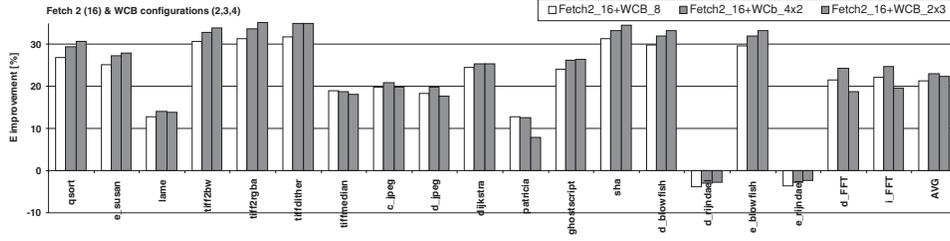
Fig. 12. Memory energy reduction for different combined configurations, for 16-entry FB.
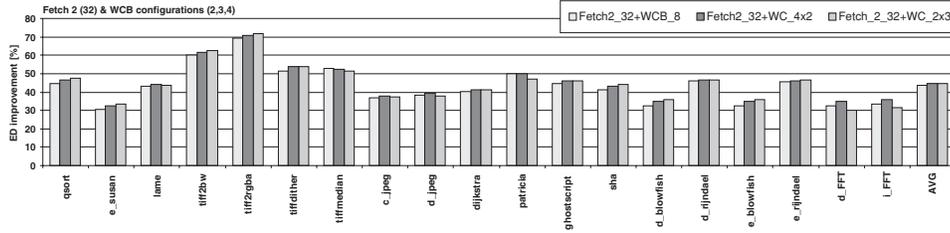

Fig. 13. Memory ED product reduction for different combined configurations, for 16-entry FB.
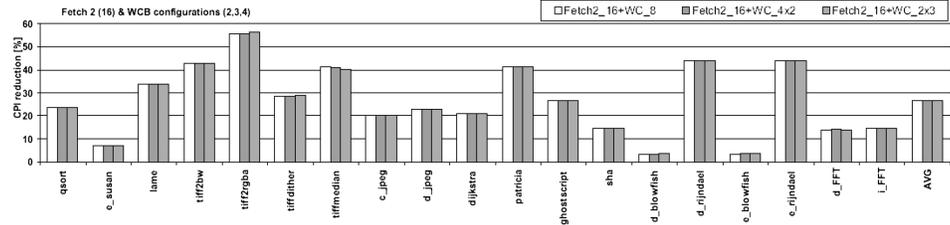

Fig. 14. CPI improvement for different combined configurations, for 16-entry FB.

SimpleScalar/Wattch. From the figure, we see that the data cache consumes more energy with a 32-byte cache line for all benchmarks, with an average of 13.6%. Even though the 32-byte cache line configuration has fewer cache misses, energy of each access is higher, thereby resulting in higher energy consumption. Further, the energy consumed for replacing a line is higher because of increased number of words per line. From above we conclude that it is not desirable to increase the cache line size beyond 16 bytes.

Figure 16 shows the memory energy consumption for 32-byte cache line size relative to the baseline configuration. From the figure we see that all the benchmarks, except *tiff2bw* and *dijkstra*, consume more memory energy in the 32-byte cache line configuration. Even though the 32-byte line configuration has fewer number of memory accesses, energy of each access is higher, thereby resulting in higher energy consumption. This results in an increase in the total memory energy consumption. The high increase in the memory energy consumption in *patricia* is because of a $1.5\times$ increase in the number of memory accesses with the 32-byte line configuration.The increase in the number of
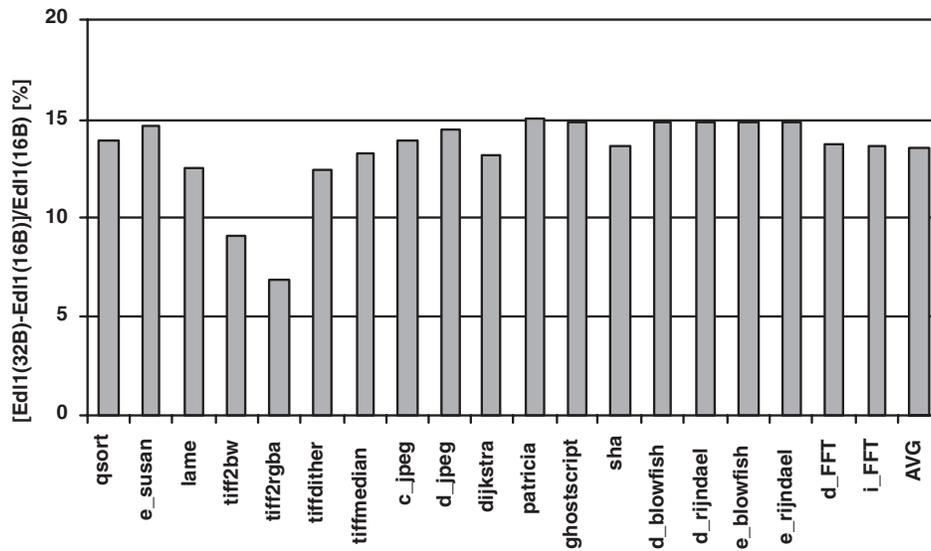
Fig. 15.   Data cache energy consumption with 32-byte relative to 16-byte cache line.
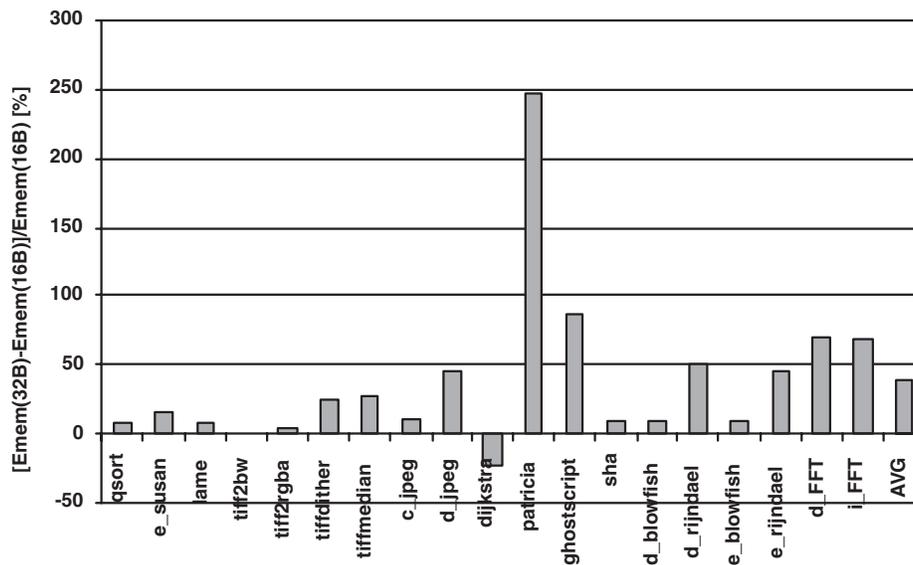

Fig. 16.   Memory energy consumption with 32-byte line relative to 16-byte line.

accesses can be attributed to the increase in the number of misses and replacements. On the other hand, both *tiff2bw* and *dijkstra* have significant decrease in number of memory accesses, thereby resulting in energy savings of 0.6 and 23.5%, respectively. On average, the memory energy consumption increased by 39.3%. Thus doubling the line is not competitive with the techniques proposed in this article.

6. CONCLUSIONS

In this article, a novel approach for reducing energy consumption for SDRAM memory access in embedded systems was proposed. It was shown that SDRAM energy consumption is, on average, $2.66\times$ higher than that of a data cache for a low-power low-cost embedded processor making DRAM energy consumption minimization very important. We introduced architectural additions to the memory controller of a fully parameterizable unit that consists of a small high-speed fetch buffer and a write-combine buffer. This allowed DRAM read prefetching and combined write access to the main memory. Since prefetched data resides in a fast and small cachelike fetch buffer, an access to it is significantly cheaper, both in terms of time and energy consumption. Combining multiple cache-line write accesses leads to gains without any penalty. Writes are buffered in a small (eight entries) fully associative buffer waiting to be combined with any subsequent access to the same SDRAM row. Separate buffers are advocated for simultaneous use of read and write combining.

The results demonstrate that a significant reduction in memory energy consumption and delays can be achieved by read prefetching and write combining. The latter also reduces the overall execution time, which, in turn, minimizes the static energy consumption. Even with small size buffers, 256 byte/512 byte for prefetching and 128 byte for write combining, an average 23% DRAM energy reduction is achieved. The DRAM energy-delay (ED) product is improved, on average, by over 40%. The CPI is reduced by 26%, on average. Further reduction in ED and CPI can be obtained if the proposed buffers were integrated in the CPU.

Prefetching or write combining can be powered down individually to better tune them to a given application. In order to utilize power down, compiler analysis, profiling or user directives could be used. The proposed technique can be applied to programs regions, such as loops or functions, where it is profitable and turned off in other program regions. For instance, the technique would not be effective for accesses with long strides, such as reading an array A[33i+7] in a loop.

The proposed approach requires simple hardware suitable to embedded systems. In a resource constrained environment of embedded systems running multimedia or other type of applications, these energy savings provide a significant benefit. As future work, we plan to evaluate the impact of technology scaling on the efficacy of the proposed apporach.

REFERENCES

ABALI, B. AND FRANKE, H.   2000.   Operating system support for fast hardware compression of main memory contents. In *Memory Wall Workshop, the 27th Annual International Symposium on Computer Architecture*.

BARR, K. AND ASANOVIC, K.   2003.   Energy aware lossless data compression. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA.

BURGER, D. AND AUSTIN, T. 1997. The simplescalar tool set, version 2.0. Technical Rep. TR-97-1342, Computer Science Department, University of Wisconsin-Madison.

DAHLGREN, F., DUBOIS, M., AND STENSTROM, P. 1993. Fixed and adaptive sequential prefetching in shared-memory multiprocessors. In *Proceedings of the International Conference on Parallel Processing*. 56–63.

EKMAN, M. AND STENSTRÖM, P. 2005. A robust main-memory compression scheme. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, Madison, Wisconsin. 74–85.

FARKAS, K., JOUPPI, N. P., AND CHOW, P. 1994. How useful are non-blocking loads, stream buffers, and speculative execution in muliple issue processors. Technical Rep. 94/8, Digital Western Research Laboratory (Dec.)

GOOSSENS, K., GANGWAL, O. P., RÖVER, J., AND NIRANJAN, A. P. 2004. Interconnect and memory organization in SoCs for advanced set-top boxes and TV — Evolution, analysis and trends. http://www.homepages.inf.ed.ac.uk/kgoossen/2004-interconnectcentric-chap15.pdf.

GUTHAUS, M. R., RINGENBERG, J. S., ERNST, D., AUSTIN, T. M., MUDGE, T., AND BROWN, R. B. 2001. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*. 83–94.

HENNESSY, J. AND PATTERSON, D. 2006. *Computer Architecture A Quantitative Approach*.

JOHNSON, T. L., MERTEN, M. C., AND HWU, W. W. 1997. Run-time adaptive cache hierarchy management via reference analysis. In *Proceedings of the 24th International Symposium on Computer Architecture*, Denver, CO. 315–326.

JOUPPI, N. P. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, Seattle, WA. 364–373.

JOUPPI, N. P. 1993. Cache write policies and performance. In *Proceedings of the 20th International Symposium on Computer Architecture*, San Diego, CA.

KANE, G. 1988. *MIPS RISC Architecture*. Prentice-Hall, Englewood Cleffs, NJ.

KIM, H. S., VIJAYKRISHNAN, N., KANDEMIR, M., BROCKMEYER, E., CATTHOOR, F., AND IRWIN, M. J. 2003. Estimating influence of data layout optimizations on SDRAM energy consumption. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, Seoul, Korea. 40–43.

KUMAR, S. AND WILKERSON, C. 1998. Exploiting spatial locality in data cache using spatial footprint. In *Proceedings of the International Symposium on Computer Architecture*.

LEBECK, A. R., FAN, X., ZENG, H., AND ELLIS, C. 2000. Power aware page allocation. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA. 105–116.

Micron. The Micron System-Power Calculator http://www.micron.com/products/dram/syscalc.html.

MicronDataSheet. The Micron: Synchronous DRAM 64Mb x32 Part number: MT48LC2M32B2 http://download.micron.com/pdf/datasheets/dram/sdram/64MSDRAMx32.pdf.

MONTANARO, J., WITEK, R. T., ANNE, K., BLACK, A. J., COOPER, E. M., DOBBERPUHL, D. W., DONAHUE, P. M., ENO, J., HOEPPNER, G. W., KRUCKEMYER, D., LEE, T. H., LIN, P. C. M., MADDEN, L., MURRAY, D., PEARCE, M. H., SANTHANAM, S., SNYDER, K. J., STEPHANY, R., AND THIERAUF, S. C. 1996. A 160–MHz, 32–b, 0.5–W CMOS RISC microprocessor. *IEEE J. Solid-State Circuits 31,* 11, 1703–1714.

SHIVAKUMAR, P. AND JOUPPI, N. 1990. Cacti 3.0: An integrated cache timing, power, and area model. Tech. rep., Digital Equipment Corporation, COMPAQ Western Research Lab.

RIXNER, S., DALLY, W. J., KAPASI, U. J., MATTSON, P., AND OWENS, J. D. 2000. Memory access scheduling. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, Vancouver, British Columbia, Canada. 128–138.

SMITH, A. J. 1979. Characterizing the storage process and its effect on the update of main memory by write through. *J. ACM 26,* 1, 6–27.

SMITH, A. J. 1982. Cache memories. *ACM Comput. Surv. 14,* 3, 473–530.

SMITH, A. J. 1991. Second bibliography on cache memories. *SIGARCH Comput. Architecture News 19,* 4, 154–182.

SOLIHIN, Y., TORRELLAS, J., AND LEE, J. 2002. Using a user-level memory thread for correlation prefetching. In *Proceedings of 29th Annual International Symposium on Computer Architecture (May)*, Anchorage, Alaska. 171–182.

TRAJKOVIC, J. AND VEIDENBAUM, A. 2004. Intelligent memory controller for reducing power consumption in embedded systems. Technical Rep. ICS-TR-04-02, School of Information and Computer Science, University of California at Irvine.

VEIDENBAUM, A., TANG, W., GUPTA, R., NICOLAU, A., AND JI, X. 1999. Adapting cache line size to application behavior. In *International Conference on Supercomputing*, Rhodes, Greece. 145–154.

Wattch. Wattch Version 1.02 `http://www.eecs.harvard.edu/~dbrooks/sim-wattch-1.02.tar.gz`.

WILKES, M. V. 1965. Slave memories and dynamic storage allocation. *IEEE Trans. Electronic Comput. EC-14,* 4, 270–271.

ZHANG, C., VAHID, F., AND NAJJAR, W. 2005. A highly configurable cache for low energy embedded systems. *Trans. Embedded Comput. Sys. 4,* 2, 363–387.