Reducing SDRAM Energy Consumption in Embedded Systems[*]

Jelena Trajkovic, Alexander Veidenbaum
University of California, Irvine
{jelenat,alexv}@ics.uci.edu

Abstract

DRAM energy consumption in embedded systems can be very high, exceeding that of the data cache or even of the entire processor. This paper presents a scheme for reducing the energy consumption of SDRAM 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, cache-like structures in the memory controller to prefetch an additional cache block(s) on reads and to combine block writes to the same DRAM row. The results quantify the DRAM energy consumption of MiBench applications and demonstrate significant savings in both the DRAM energy consumption, an average of 23%, and the energy-delay product, an average of 44%. The approach also improves performance: the CPI is reduced by 26% on an average.

1. Introduction

Many embedded applications are memory intensive, especially multimedia applications. Memory access constitutes a significant portion of overall energy consumption in such applications [2, 10]. This research investigatges an architectural approach to reducing the memory system energy without any performance loss. In fact, it will be shown to result in a performance gain.

Much of prior research to reduce energy consumption has focused on caches. But main memory consumes orders of magnitude more energy per access than cache. As will be shown below, in some applications the total energy of main memory accesses can be an order of magnitude higher than the total data cache energy consumption. Thus it is very important to optimize DRAM access for energy consumption. Some of the techniques proposed for cache optimization can be extended for this purpose.

SDRAM memory is one of the major types of DRAM used in embedded systems. The energy of 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 65% of the total access energy (per manufacturer's data). The SDRAM organization allows the bank/row to be left "on" after an access, which permits additional read/write access to such 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 35%.

Embedded systems use a data cache and only access memory on cache misses or write-backs. Thus the memory is read/written in (cache) blocks (lines). Therefore this paper proposes reading/writing multiple lines at a time, i.e. within a single activate-precharge pair. This will be shown to lead to significant energy savings. Multiple block reads are accomplished via hardware prefetching and writes via write combining [16] at the memory interface.

Accessing multiple lines requires intermediate storage. This research proposes to use a small amount of such storage in the memory controller. Additional lines will be prefetched 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 contributions of this research are adapting both prefetching and write combining to SDRAM energy reduction, in particular combining writes to the same SDRAM row.

[*]This research was supported in part by the National Science Foundation under Grant No. NSF CCR-0311738.

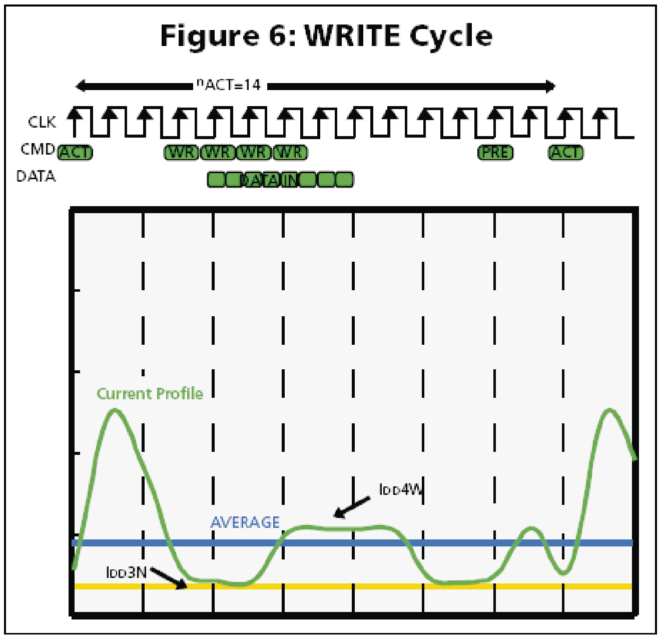

Figure 1. Write current profile

A second goal of this research is to avoid effecting the execution time or even to improve it while saving energy. The small buffers used for prefetching/combining act as a memory cache and can significantly improve read performance. Prefetching can sometimes degrade execution time by interfering with ”regular” memory access without supplying useful data. Write buffering enables the processor to continue execution and not wait for the write access to finish.[1]

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 describes 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

Many architectural approaches for reducing energy consumption in embedded processors have been proposed. We are unaware of architectural solutions for SDRAM energy reduction. For a software-based solution see [10], for instance. There is a large body of prior work on prefetching, write buffers, and write combining buffers that is briefly summarized below. Techniques for reducing energy consumption in main memory are also described.

Numerous prefetching algorithms have been proposed ([16, 8, 4, 20, 17]) based on history based prediction of future memory addresses. They differ in their way of predicting which block to fetch, how many blocks to fetch and what triggers prefetch. Many of these schemes require a complex prediction mechanism, although the so-called ”one-block-lookahead” schemes does not. For instance, a stream buffer [8] prefetches N consecutive lines triggered by a cache miss. None of the prefetching proposals targets energy reduction.

Write buffers [16, 9] have been used in many processors to avoid waiting for data to be written to memory and to avoid delaying cache miss fetch. Merging was introduced to improve performance of write buffers for write-through caches. This approach combines an incoming write request within a cache line with requests already residing in the write buffer, resulting in a more efficient memory usage. The technique has been implemented in many architectures: Digital’s VAX 8800 [6], StrongARM [14], MIPS R4300i [13], and Intel XScale [3].

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 [12].

Efforts have been made in reducing memory energy consumption based on different compression techniques. For instance,

[1]It is assumed that there is no post-L1 cache write buffer in a CPU with a write-back cache.

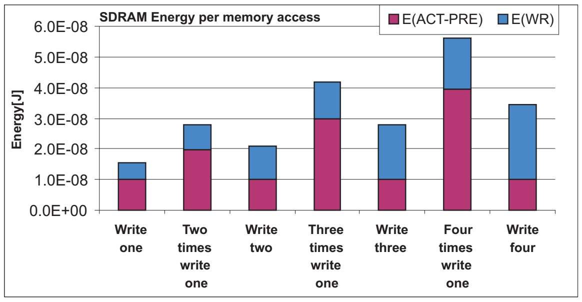

Figure 2. Energy per memory access

[1] describes and evaluates a computer system that supports hardware main memory compression. As the compression ratio of applications dynamically changes so does the real memory size that is managed by the operating system (OS). 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. [10] 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 tradition one and determine the best block size such that a number of requests for different row or bank is minimized.

3. SDRAM Energy Components

Let us identify potential sources of energy saving in SDRAM memories that are used in embedded devices.

In order to access data from an 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*.

Micron describes the current profile for the write (or read) operation in such memory [19] as shown in Figure 1[2] (reproduced from [19]) in Micron's Technical Note. The first large peak in 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 the precharge command. This shows that activation-precharge pair constitutes a significant portion of the overall current and thus the energy consumption.

Figure 2 quantifies energy components of a 64Mb SDRAM memory [18]. These were obtained using Micron's System Power Calculator [19]. 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 (B) of data (one cache line), two separate 16B writes, and two 16B writes "combined" in one activation-precharge. It also presents data for 3 and 4 accesses, performed separately or combined.

The figure shows the activate-precharge pair to be the dominant component. It contributes 65% to the total energy consumption for this SDRAM. The energy savings from combined read or write accesses are 24%, 34% and 38% for two, three and four combined accesses, respectively, comparing to same size performed in sequence.

4. Proposed Approach

The memory subsystem of a typical embedded system is shown in Figure 3a. The baseline configuration consists of a CPU with a single level of cache and a main memory. The cache is write-back. Memory latency is high in terms of processor cycles and the processor has to stall for many cycles waiting for the data to arrive from the memory.

As discussed above, it would be good to fetch more than one line on each cache miss and thus save energy. This has to be done under the usual constraints of embedded systems: reduced cost, energy, and complexity. Because of that the prefetching has to be "precise", cannot use very complex logic due to potential time and energy overheads , and cannot use a large buffer

[2]©2001 Micron Technology, Inc. All Rights Reserved. Used with permission.

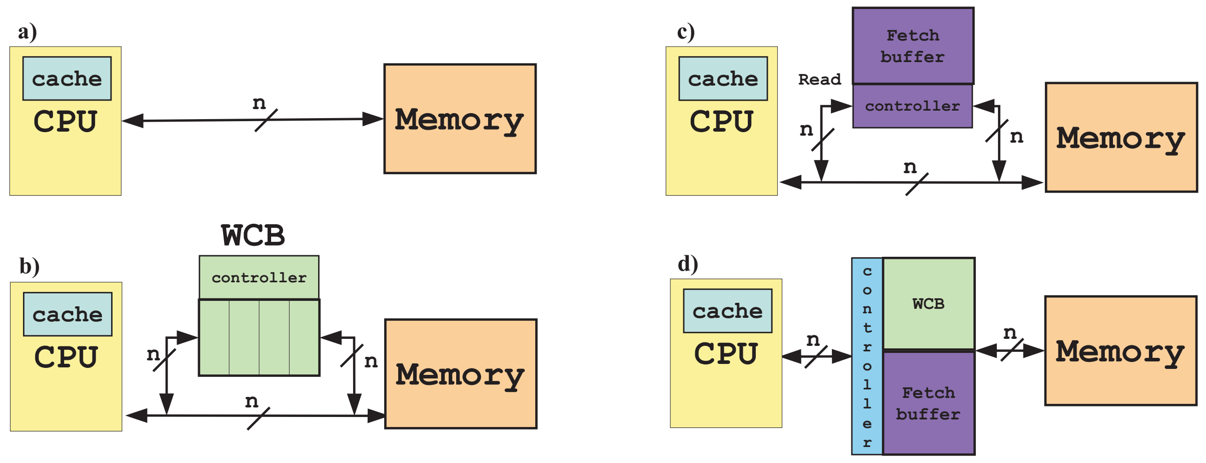

Figure 3. Memory subsystem architectures

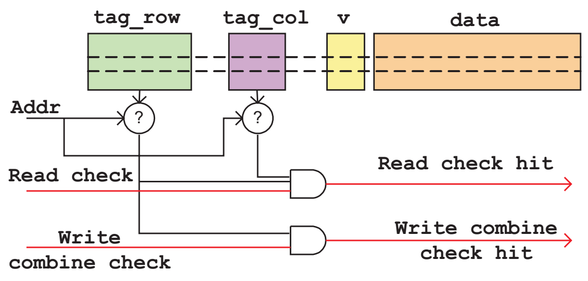

Figure 4. Architecture of write-combine buffer

memory, again due to overheads. Thus, the approach chosen for this work is to use a simple one-block lookahead [**?**] or stream buffer-like [**?**] prefetching.

Combining of multiple line writes is used in this paper with a goal of energy reduction. Thus a different type of combining or coalescing write buffer (WCB) is proposed for writes. The difference is that it should be able to combine *any* two writes to the same SDRAM row. It also needs to be small simple for the same reasons as discusses above for prefetching.

Let us initially discuss write and read combining separately to understand the benefits and requirements of each of them. Next, the ”combined” approach will be investigated and the best mechanism selected. In all cases, 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 Write Combining

The memory controller for write combining is shown by Figure 3b. Figure 4 shows the write-combining buffer (WCB) architecture for combining 2 write requests. Each entry consists of a split tag, a valid bit, and data storage for one cache line. Tag bits are divided into two groups: 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_row entries. LRU or pseudo-LRU replacement is used.

A hit in the WCB means that the incoming write request can be combined and performed together with a valid WCB entry. Once the memory write is performed the WCB enrty is freed. Notice that the incoming write is not written into the WCB in this case.

A WCB write miss causes the incoming request to be stored in the WCB and to be potentially combined with a future write. A write to the WCB may cause a replacement and write-back from the WCB of a single-block entry.

This architecture can be extended to combine more than 2 accesses. To combine N+1 writes, N tag_col sub-tags, N valid bits, and N data store blocks are stored with each 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 memory on a hit if the entry counter has a value of N. On replacement less than N entries may be written to memory, as specified by the counter value.

To summarize, the WCB differs from the traditional write buffer or even a coalescing write buffer because it can merge data 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 thus to write N+1 entries as often as possible. A traditional write buffer, on the other hand, can only coalesce individual words (or sub-words) *within a cache line* and writes data to memory when the memory is not busy.

A major advantage of this new form of write combining is that it can not incur any energy losses. 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. It is solved as follows: every read address is checked against the full line address (i.e. both tag_row and tag_col bits) 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 miss latency reduction.

4.2 Read Prefetching

The goal of read combining is to perform multi-line DRAM reads. However, since there is only one read miss at any given time in an embedded system (with an in-order CPU), 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 other sequential prefetching mechanisms mentioned above do. The difference is that our ”prefetching” is aimed at main memory energy reduction.

It is possible to prefetch non-adjacent 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 [11]). This is why only simple, sequential prefetching is considered here.

The memory controller for read combining is shown in the Figure 3c. It fetches N additional cache lines on a read miss. The lines are stored in a fetch buffer (FB): a small, cache-like 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 an FB miss the line is read from the DRAM together with N additional lines which are 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 deal with N = 1, 2, and 3.

As will be shown, a small, fully associative FB is sufficient to achieve significant energy reduction. In addition, the performance is also improved due to FB's caching effect.

4.3 Read and Write Combining

Each of the write and the read combining has its own individual advantages. They are largely independent of each other and thus can be deployed together for an additive energy reduction as well as performance improvement. The question is what is the best architecture to perform the read and the write combining at the same time.

The architecture advocated by this paper is shown in Fig. 3d. This solution basically integrates the separate *fetch buffer (FB)* and *write-combine buffer (WCB)*. While a single, cache-like structure can be designed, it will have two major disadvantages. First, it will likely require that N, the number of cache lines to combine, be the same for reads and writes. As will be shown in this paper this is not desirable. And second, more importantly, it will make it more difficult to perform sequential read combining and ”within the SDRAM row” write combining which is very important. In addition, there will be interference and replacements of write lines by read prefetches and vice versa in this case. Also, the split tag is not required for read combining.

Merging the WCB and FB designs is not very 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. 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.

Benchmark	Emem relative to Ecache [%]	Benchmark	Emem relative to Ecache [%]
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

Table 1. Memory energy relative to the energy of data cache

Briefly, the solution is two-fold. 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 algorithm that the controller implements to keep coherency between buffers is:

```
On outgoing cache request
if replacement then
  check write in WCB
  invalidate FB entry if exists
else
  check read in WCB and FB in parallel (hit is possible in only one of buffers)
  if hit in WCB then
    supply data to the CPU
  end if
  if hit in FB then
    supply data to the CPU
  else
    generate N prefetch addresses
    check for the same row/bank (drop ones that exceed row/bank boundary) $N_1 \leq N$
    check for existence in WCB (drop matched) $N_1 \leq N_2$
    fetch $N_2 + 1$ addresses
  end if
end if
```

The only potential drawback of the combined operation is over-utilization of 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 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, 32b embedded processor resembling Intel's Xscale. It has a 8KB, 4-way set associative instruction and data caches with a 16Byte line and a 2 cycle latency. Data cache implements a *write-allocate, write-back* policy. The CPU operating frequency is 400MHz. The CPU memory bus is a 100 MHz, 32b bus. The baseline cache miss latency is 36 processor cycles for the first word to arrive, and an additional 4

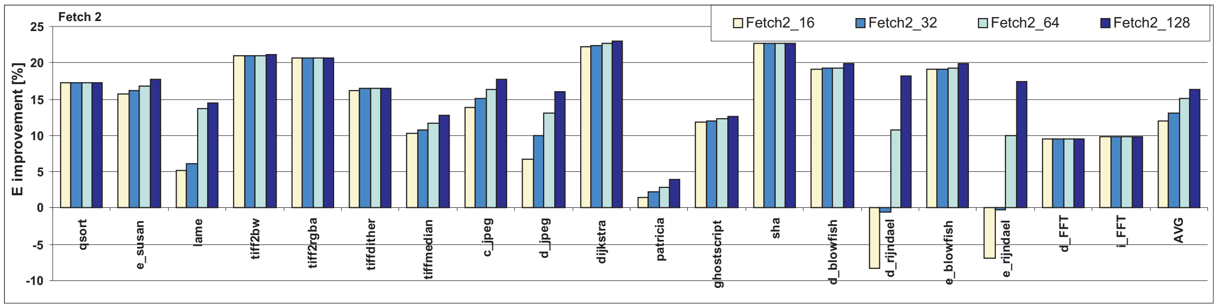

Figure 5. Memory energy reduction for read combining for different buffer sizes

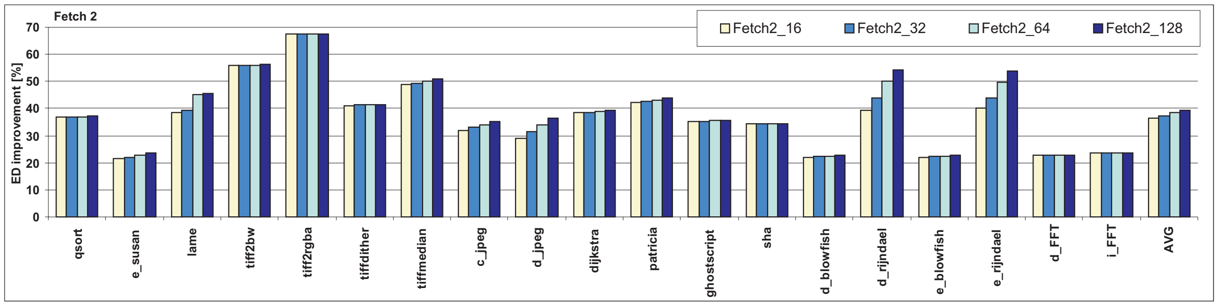

Figure 6. Memory ED product reduction for read combining for different buffer sizes

processor cycles for each consecutive word.

The main memory with the modified controller has a latency of 40 processor cycles for the delivery of the first word, and 4 cycles for each additional consecutive word. Both baseline and modified architectures use the same SDRAM (see data sheet [18]). The extra 4 cycles (10ns) in the access time to modified memory are due to FB and WCB access delays. The SDRAM clock rate is is 100MHz (speed grade -6).

The evaluation is performed using the SimpleScalar 3.0 simulator [5] executing PISA binaries. SimpleScalar's bus and memory model are modified to match this architecture.

Both FB and WCB are fully associative, with 16-byte lines and a latency of 12 processor cycles. The WCB can be configured to store N lines per entry.

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 and assuming a 0.18 micron process technology is used.

Dynamic energy consumption of the cache, WCB, and FB is modeled using modified CACTI 3.2 [15] for 0.18 micron technology. One of the main changes are in the sense amplifier energy model, which was overestimated in the original model. Main memory energy is modeled using Micron's System Power Calculator [19]. Benchmarks from a MiBench [7] are used in this study. All benchmarks are simulated using "large" input sets.

5.1 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 2 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;

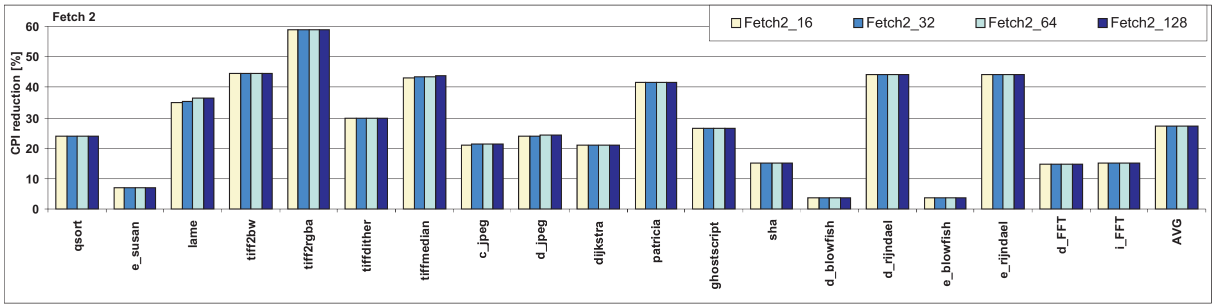

Figure 7. CPI improvement for read combining for different buffer sizes

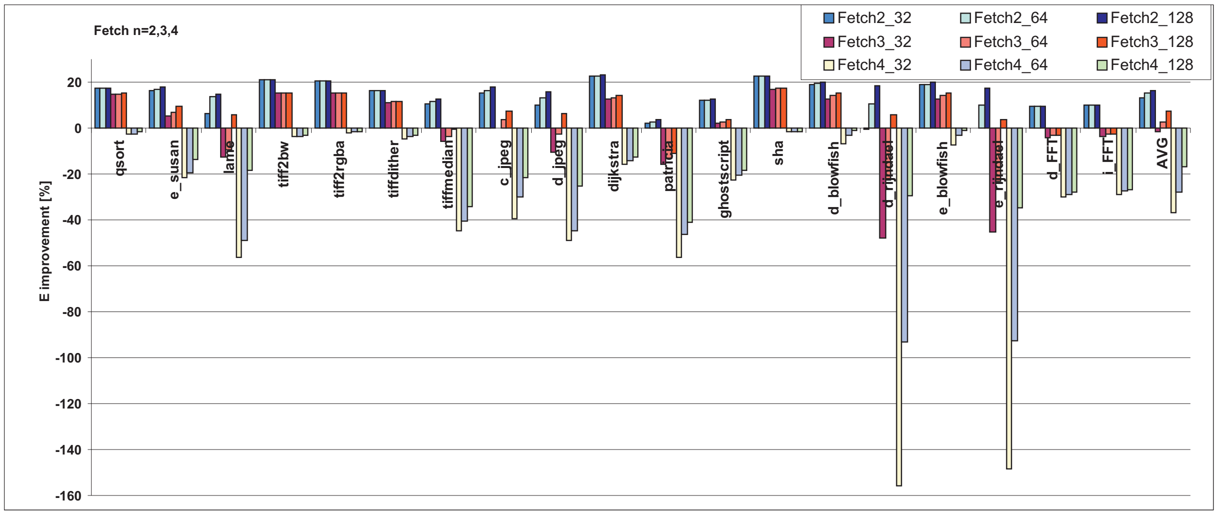

Figure 8. Memory energy reduction for read combining for different fetch and buffer sizes

- "FetchN_M+WCB_PxQ" for a hybrid configuration.

Table 1 shows memory energy per benchmark relative to the data cache memory consumption for the baseline model. This is with an average cache miss rate of 3.5%. [14] showed the data cache consuming 16% of overall the processor energy. For MiBench benchmarks the main memory consumes, on an average, 2.6 times the energy of the data cache. The worst case difference is 15x.

5.1.1 Read Prefetch: the Effect of Fetch and Buffer Size

First, let us evaluate read combining and its effect on memory energy consumption. Figure 5 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 16B blocks. The average memory energy savings are 12% to 17%. The smallest buffer already obtains a significant reduction, with each doubling of the size producing a small (1% to 2%) increase. Two benchmarks *d_rijndael* and *e_rijndael* have a noticeable increase in memory energy consumption for a 16-entry FB. With 32 entries there are basically no energy increases, making it a good choice.

The energy-delay (ED) product shown in Figure 6. It is reduced by as much as 68% and by 38% on average, with buffer size having almost no impact. It can be seen, that both benchmarks that have energy increase obtain significant ED product saving. The energy delay reduction is large due to an improved average memory latency. The effect of latency reduction can be seen in the CPI improvement shown in Figure 7. CPI is also insensitive to the buffer size change. Read combining technique reduces CPI by as much as 59% and by 27% on an average.

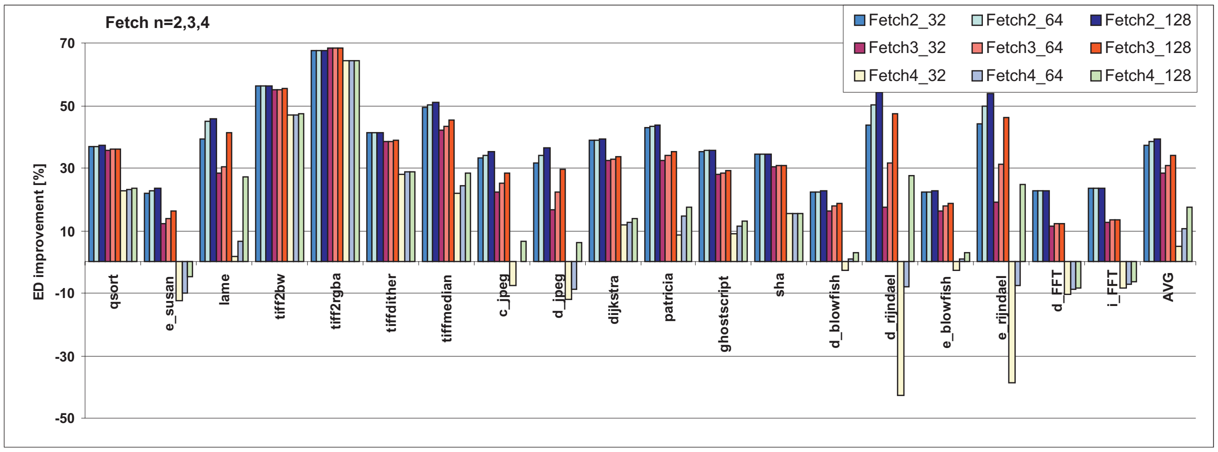

Figure 9. Memory energy-delay product for read combining for different fetch and buffer sizes

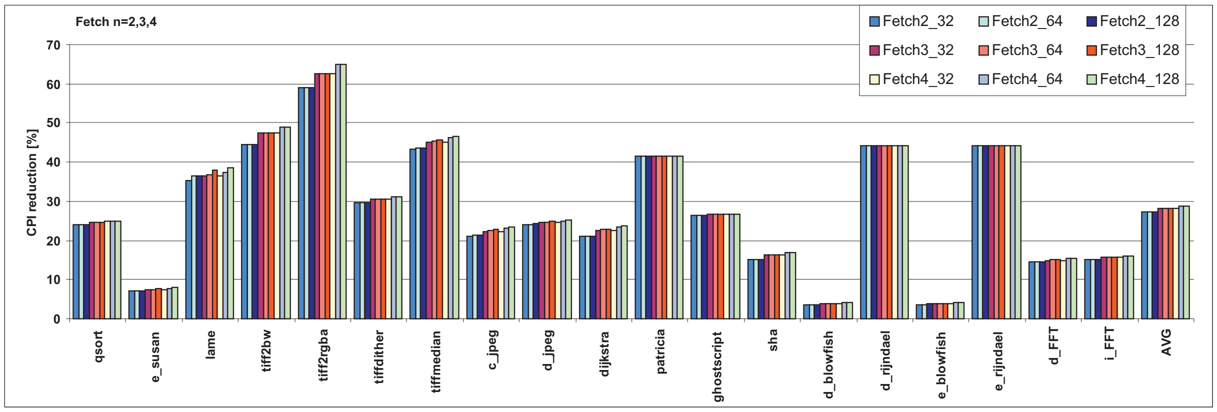

Figure 10. CPI reduction for read combining for different fetch and buffer sizes

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. The only two benchmarks that have significant difference in ED product saving (15%) are *d_rijndael* and *e_rijndael* . Therefore, for combined technique, we will explore read-combine buffer of 16 and 32 entries.

Let us now explore the use of different fetch size. Figure 8 shows the energy reduction relative to the baseline configuration. We have considered fetching 2, 3 and 4 lines and buffer sizes of 32, 64 and 128 entries. It can be seen that a fetch size larger than 2 is not beneficial. Generally, a larger fetch size increases energy consumption because many prefetched lines are not used. Fetch size 3 in some cases reduces energy consumption, but not as much as fetch size 2, while fetch size 4 always has too much overhead. On average, a fetch size of 2 saves 13% to 16% of energy, fetching 3 lines saves from -1% to 8% and fetching 4 increases energy consumption by 16% to 37%.

Figures 9 and 10 present ED product and CPI savings. It can be seen that for ED product savings, fetch size 2 uniformly outperforms other fetch sizes. The only exception is *tiff2rgba* for fetch 3 where the difference is just 1%. The largest saving is 68% and the average improvement ranges form 5% to 39%. CPI saving is not significantly affected by any parameter change except for *lame*, *tiff2bw*, *tiff2rgba*, and *tiffmedian*, where the difference is less than 5%. In the best case, a 65% improvement is obtained, and on average the improvement ranges from 27% to 29%.

5.1.2 Write Combining: the Effect of Combining and Buffer Size

Figure 11 shows the relative memory energy for 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,

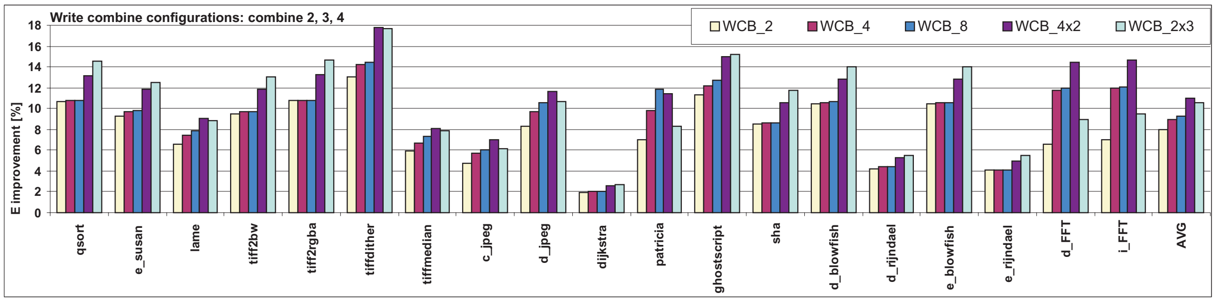

Figure 11. Memory energy reduction for different write-combining and buffer sizes

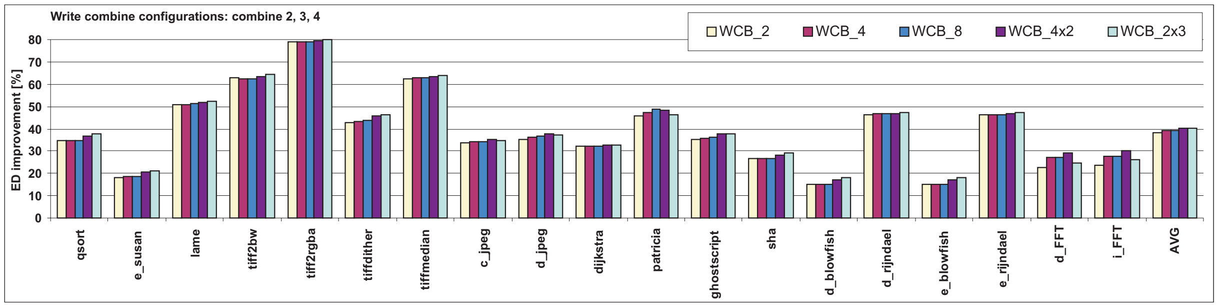

Figure 12. Memory ED product reduction for different write-combining and buffer sizes

the improvement ranges from 8% to 11%; that is smaller than for read combining. Buffer size has little impact (the left three bars), but additional energy savings are obtained when combining 3 or 4 lines. Write combining achieves up to an 80% reduction of ED product (see Figure 12) with a 40% average. CPI savings (Figure 13) are also not affected by size or configuration. Write combining achieves an up to 76% CPI improvement, with a 33% average.

5.1.3 Hybrid Configurations

Figure 14 shows the effect of both write and read combining. The fetch buffer with 16 entries is used with write combining buffers of size 8, configured to combine either 2, 3 or 4 writes. The results show that combining 3 lines is the best configuration. On average 21.5% to 23.5% energy savings are obtained. As seen in Figures 15 and 16, the difference in ED product and CPI savings for different configurations is not more than 2%. ED product is reduced by 71% in the best case and by 44% on average. CPI is improved by up to 56%, with a 26% on average.

Figure 17 shows the effect of both write and read combining when the fetch buffer with 32 entries is used with write combining buffers of size 8, configured to combine either 2, 3 or 4 writes. Still, combining 3 lines gives the best configuration. On average 22% to 24% energy savings are obtained. The difference in ED product and CPI saving is negligible (2%) for different configurations, as seen in Figures 18 and 19 respectively. ED product and CPI reduction are the same as for the configuration with 16-entry read-combining buffer.

6. Conclusions

This research developed a technique for reducing energy consumption for SDRAM memory access in embedded systems. 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 read prefetching and combined write access to the main memory. Since prefetched data resides in a fast and small cache-like memory, an access to it is significantly cheaper, both in terms of time and energy consumption. Combining write accesses leads to gains without any penalty. The technique was evaluated using the SimpleScalar simulator of an Xscale-like embedded processor.

The results demonstrate that a significant reduction in memory energy consumption and delays can be achieved by read prefetching and write combining. Even with small size buffers, 256B/512B for prefetching and 128B for write combing, an

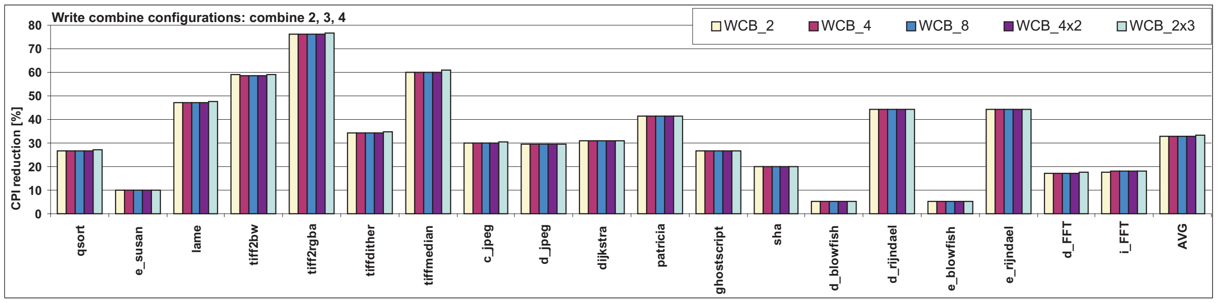

Figure 13. CPI improvement for different write-combining and buffer sizes

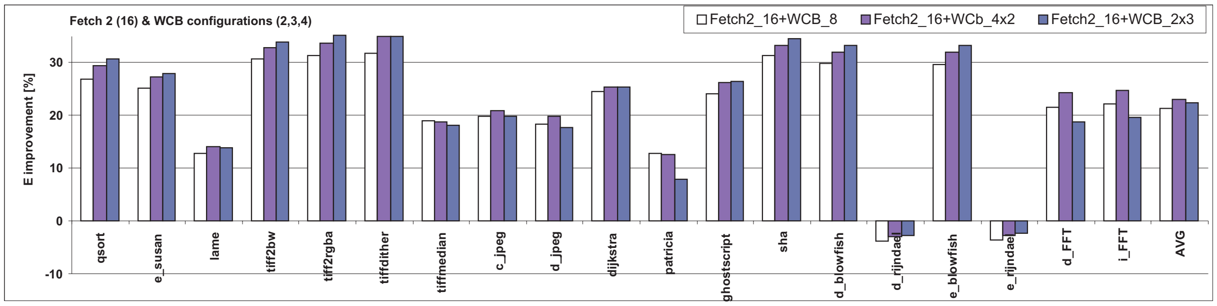

Figure 14. Memory energy reduction for different combined configurations, for 16 entry FB

average 23% energy reduction is achieved. The energy-delay product is improved, on average, by over 40%. The CPI is reduced by 26%, on average.

Prefetching or write combining can be powered down individually to better tune them to a given application. The proposed approach requires simple hardware suitable to embedded systems. In a resource constrained environment of embedded systems running multimedia applications these energy savings provide a significant benefit.

References

[1] B. Abali and H. Franke. Operating system support for fast hardware compression of main memory contents. In *Memory Wall Workshop, the 27th Ann. Int. Sym. On Computer Architecture*, 2000.

[2] K. Barr and K. Asanovic. Energy aware lossless data compression. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services (MobiSys 2003), San Francisco, CA*, 2003.

[3] L. T. Clark and et al. An embedded 32b microprocessor core for low-power and high-performance applications. *IEEE JSSC*, 36(11):1599–1608, Nov. 2001.

[4] F. Dahlgren, M. Dubois, and P. Stenstrom. Fixed and adaptive sequential prefetching in shared-memory multiprocessors. In *In Proceedings of the 1993 International Conference on Parallel Processing,*, pages 56–63, 1993.

[5] D.Burger and T. Austin. The simplescalar tool set, version 2.0. Technical report, Technical Report TR-97-1342, University of Wisconsin-Madison, 1997.

[6] J. Fu, J. Keller, and K. Haduch. Aspects of the vax 8800 c box design. *Digital Technical Journal, Number 4*, February 1987.

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, pages 83–94, 2001.

[8] N. P. Jouppi. 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*, pages 364–373. ACM Press, 1990.

[9] R. Kessler, E. McLellan, and D. Webb. The alpha 21264 microprocessor architecture. In *ACM SIGPLAN Notices*, 1998.

[10] H. S. Kim, N. Vijaykrishnan, M. Kandemir, E. Brockmeyer, F. Catthoor, and M. J. Irwin. Estimating influence of data layout optimizations on sdram energy consumption. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 40–43. ACM Press, 2003.

[11] S. Kumar and C. Wilkerson. Exploiting spatial locality in data cache using spatial footprint. In *International symposium on Computer Architecture*, 1998.

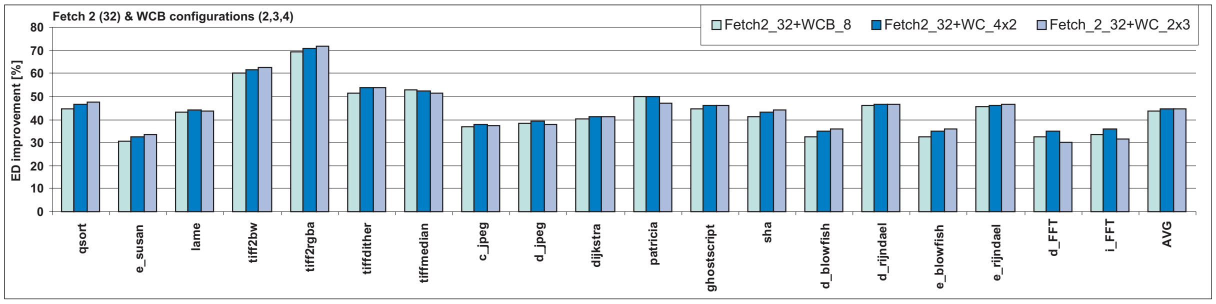

Figure 15. Memory ED product reduction for different combined configurations, for 16 entry FB

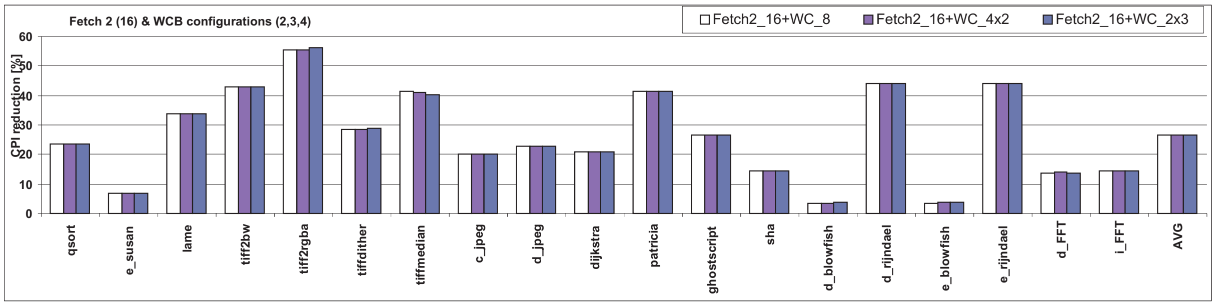

Figure 16. CPI improvement for different combined configurations, for 16 entry FB

[12] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power aware page allocation. In *In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX), November 2000*, 2000.

[13] MIPS Technologies, Inc.: R Series Documents http://www.mips.com/content/PressRoom/TechLibrary/RSeriasDocs.

[14] J. Montagnaro and et al. A 160–mhz, 32–b, 0.5–w cmos risc microprocessor. *IEEE JSSC*, 31(11):1703–1714, Nov. 1996.

[15] P.Shivakumar and N. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical report, Digital Equipment Corporation, COMPAQ Western Research Lab, 1990.

[16] A. J. Smith. Cache memories. *ACM Comput. Surv.*, 14(3):473–530, 1982.

[17] Y. Solihin, J. Torrellas, and J. Lee. Using a user-level memory thread for correlation prefetching. In *In Proceedings of 29th Annual International Symposium on Computer Architecture, May 2002.*, 2002.

[18] The Micron: Synchronous DRAM 64Mb x32 Part number: MT48LC2M32B2.

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

[20] A. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji. Adapting cache line size to application behavior. In *Int'l Conf. Supercomputing*, 1999.

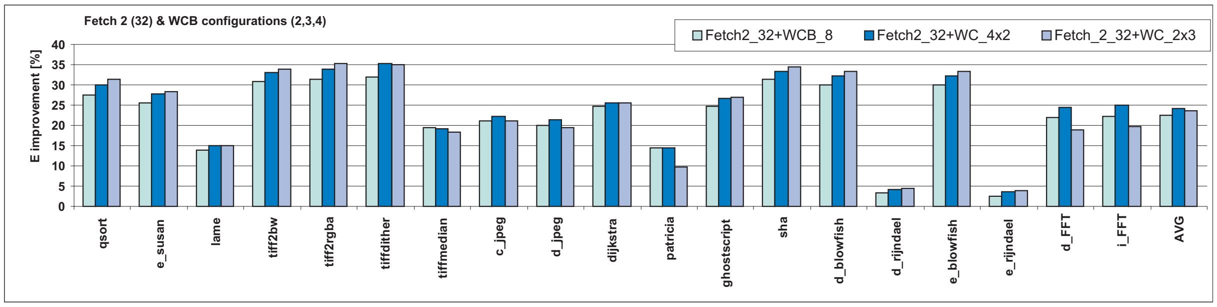

Figure 17. Memory energy reduction for different combined configurations, for 32 entry FB

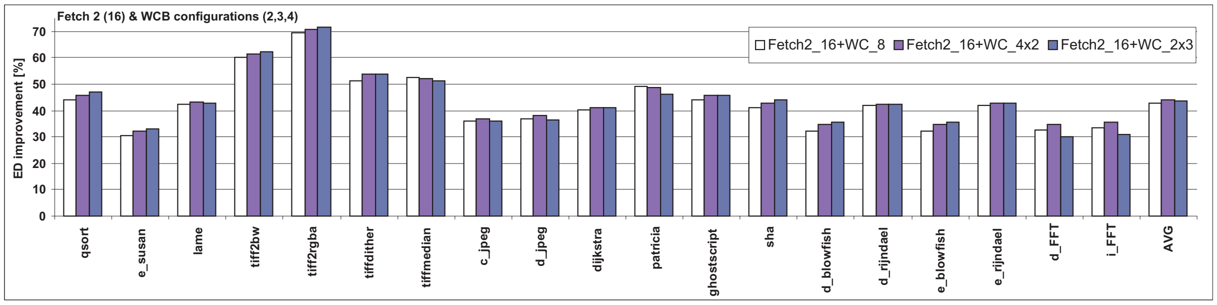

Figure 18. Memory ED product reduction for different combined configurations, for 32 entry FB

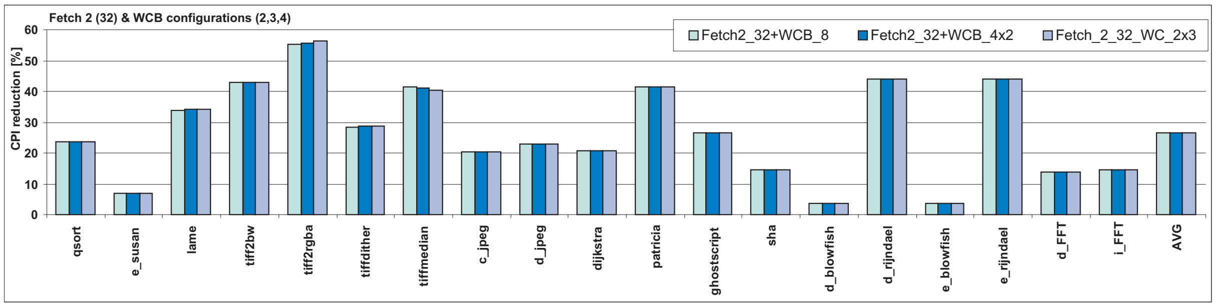

Figure 19. CPI improvement for different combined configurations, for 32 entry FB