REDUCING POWER IN SUPERSCALAR PROCESSOR CACHES USING SUBBANKING, MULTIPLE LINE BUFFERS AND BIT–LINE SEGMENTATION*

Kanad Ghose and Milind B. Kamble**
Department of Computer Science
State University of New York, Binghamton, NY 13902–6000
email: {ghose, kamble}@cs.binghamton.edu

Abstract

Modern microprocessors employ one or two levels of on–chip caches to bridge the burgeoning speed disparities between the processor and the RAM. These SRAM caches are a major source of power dissipation. We investigate architectural techniques, that do not compromise the processor cycle time, for reducing the power dissipation within the on–chip cache hierarchy in superscalar microprocessors. We use a detailed register–level simulator of a superscalar microprocessor that simulates the execution of the SPEC benchmarks and SPICE measurements for the actual layout of a 0.5 micron, 4metal layer cache, optimized for a 300 MHz. clock. We show that a combination of subbanking, multiple line buffers and bit–line segmentation can reduce the on–chip cache power dissipation by as much as 75% in a technology–independent manner.

Key words: Low power caches, power estimation.

1. INTRODUCTION

All modern microprocessors incorporate one or two levels of on chip caches to accommodate the large memory bandwidth requirements of the superscalar pipeline. To allow fast pipeline clock frequencies to be used, these on–chip caches are implemented using arrays of densely packed static RAM cells. The device count for these on–chip caches are often a significant fraction of the total transistor budget for the entire chip; indeed in some cases, the number of transistors devoted to the on–chip caches exceeds the transistor count for the processor's datapath and control logic. An extreme example of this is found in the HP PA 8500 microprocessor where 70% of the die area is occupied by large L1 caches to speed up OLTP applications. These on–chip caches are thus a significant source of power dissipation (e.g., 25% of the total chip power for the DEC 21164 [ERB+95], 43% of the total power for the SA–110 [Mon 96]).

* This work is supported in part by the National Science Foundation through award No. MIP–9504767.
** Currently with the Hewlett–Packard VLSI Technology Laboratory, Fort Collins, CO 80528. Work performed while at SUNY–Binghamton.

In the short term, as processor vendors put additional transistors made available through improvements in device technology into the on–chip caches, the fraction of the total chip power dissipated in the caches is likely to go up. There are several reasons why significant power dissipation occurs within the on–chip caches. First, the tag and data arrays of these caches are implemented as static RAMs, to allow the cache access rate to match the pipeline clock rate. Second, the cache area is more densely packed than other areas on the die, so that the number of transistors devoted to the cache are quite a significant percentage of the total number of transistors on the die. Third, these caches are frequently accessed; the I–cache is accessed on each clock cycle unless a cache miss occurred.

Several techniques have been proposed (and actually used in some cases) for reducing the power dissipated by caches in general. These include the use of: (a) alternative organizations [SuDe 95, KaGh 97, KBN 95], including similar techniques that apply to SRAMs [EvFr 95, Itoh 96]; (b) circuit design techniques that are generally applicable to SRAM components [HKY 95, Itoh 96], (c) Alternative realizations [FrPe+ 97] and (d) instruction scheduling techniques, such as the ones presented in [SuDe 95]. The techniques in these four categories are generally independent from each other and can thus be used in conjunction. The focus of this paper is on techniques that fall into the first category.

In this paper we propose and evaluate the use of *multiple* line buffers, subbanking and bit line segmentation for reducing the power dissipation within on–chip caches for superscalar CPUs. We show that a suitable combination of these techniques can be extremely effective in reducing the energy dissipation in all caches across the memory hierarchy.

2. ORGANIZING CACHES FOR LOW POWER

The dominant cache organization employed in modern microprocessors today is the set–associative cache [Smith 82], [Handy 93]; the direct–mapped cache and fully associative cache organizations are two extremes of this organization. In a normal *m*–way set associative cache, there are *m* tag and data array pairs, each consisting of S rows, where $S = 2^s$. Each data array location is known as a cache line, which is a group of W consecutive words ($W = 2^w$). A cache line is capable of holding the contents of what is called a memory block, consisting of W consecutive memory words. Tag and data array locations at the same offset within the tag and data arrays make up what is called a *set*. The placement rules for such a cache dictates that a word at an address A in the

memory (RAM), if present in the cache, can be found in any cache line within the set at offset (A div W) mod S. To uniquely identify the memory block that resides within a cache line, the tag part of the address (obtained by stripping the lower order (s+w) bits of A) are kept in the associated tag array location. The access steps for the m–way set–associative cache are as follows:

Step 1: Use the middle order s bits in the address of the word to be accessed to read out the set that potentially contains these words into output latches of the tag and data arrays. These latches together make up what is called a **line buffer.**

Step 2: Compare the tag part of the address being accessed in parallel with the m outputs from the tag arrays (using m independent comparators). A match with the output of the tag array indicates that the required data is within the output latch of the corresponding data array. If this is the case, a situation called a cache hit, the lower order w bits of the address are used to multiplex out the desired word from the data array output latch. If no match occurs, we have a cache miss, implying that the desired data is not within the cache.

In the pipelined caches that we have implemented in our simulator, the two steps mentioned above are naturally divided into 2 stages, thereby providing a peak throughput of one cache request per cache cycle (per port). The tag and data arrays need to be updated when a missing line is fetched or when a STORE request updates the contents of an existing line. This update of the arrays is performed by a third stage of the cache pipeline. On a cache miss, some replacement algorithm – implemented in hardware – is used to select a victim line from the set read out in step 1 and appropriate steps are followed to install the memory block into the victim's line frame when it is delivered from the lower levels of the memory hierarchy.

For a superscalar CPU, to maximize the number of instructions to be examined for dispatch in each cycle, a facility is needed to transparently step across the cache line boundary to the physically next line (if that line is cached). One way of doing this is to use a deck buffer mechanism (as used in the MIPS 10K) or to use odd and even cache banks (with an automatic line address incrementing facility, as originally used in the IBM RISC/6000 [Gro 90]). Modern superscalar CPUs also tend to support multiple pipelined load/store units that can request access to the D–cache simultaneously. This requirement is met by using a multiported cache or an interleaved cache.

2.1 Caches with multiple line buffers

We propose the extension of the set–associative cache with multiple output latches i.e., line buffers. The key idea is to exploit the temporal (and spatial) locality of reference in address streams. There is a very good chance that the cache line being currently accessed was accessed in the recent past, and that this cache line is still resident in one of the output latches. If the cache line being currently accessed is indeed present in one of the line buffers (a situation called a line buffer hit), Step 1 above of the normal cache access can be aborted and the desired line can be accessed directly from the line buffer. This not only saves accesses of the data arrays but, at the same time, also saves the access to the tag arrays.

Caches with a single line buffer were introduced by Su and Despain in a slightly different form in [SuDe 95]. In their design, the line address of the current access is first compared against the line address of the set that is currently resident in the line buffer. The normal access of the cache – including bit line precharging and row address decoding – is started only when a mismatch occurs. Consequently, this arrangement prolongs the cache access latency, a solution unattractive in practice. Second, if the cache access is pipelined in two stages, a completely new set of tag comparators are needed (along with a comparator for the set number) to allow a line buffer hit to be determined in parallel with the tag comparison step of a normal cache access for the *previous* access. Both of these problems are not present in our proposed line buffering scheme, as described below.

Figure 1 depicts a set–associative cache with multiple (in this case, 4) line buffers. The hardware augmentations to the basic set–associative cache are as follows. First, we need four latches to hold the number of the four most recently accessed sets. Second, four more comparators are needed to compare the set number field of the current access against the set numbers of the lines sitting in the four line buffers. Third, if no line buffer hit occur, the data from the set selected by the current address applied to the cache must be retrieved into one of the four line buffers. Consequently, on a line buffer miss, a victim must be selected from one of the four line buffers. The need to write back an updated line buffer's content into the tag and data arrays, when the line buffer gets selected as a victim, is avoided by writing through updates to the line buffer into the corresponding data array on write accesses that result in a line buffer hit. Along with these, a multiplexing facility is needed to direct the outputs of the tag and data arrays into the victim line buffer. In this case, the four line buffers effectively make up a 4–entry fully–associative write–through cache, which is accessed in parallel with the normal cache without any prolongation in the cycle time or without any impact on the normal cache access pipeline.

The access steps for a pipelined cache with multiple line buffers are:

Cycle 1:

ϕ1: – Precharge the tag and data arrays for a read access
– Start the decoding of the set address applied to the arrays (This is identical to what happens in normal set–associative cache).
– Simultaneously, compare the set selector field in the address being accessed with the set numbers of the four sets stored in the four line buffers.
– Concurrently, identify a victim line buffer in advance to handle misses on the line buffers and start the setup of the multiplexor at the output of the tag and data arrays.

ϕ2: – If the set number of the current access matches the set number associated with any of the line buffers (line buffer hit), abort the readout of the tag and data arrays and steer the tag and data values from the matching buffer into the tag comparators and the word multiplexer, respectively.
Otherwise (on a line buffer miss), latch in the contents of the set read out from the arrays into the victim line buffer and move the set

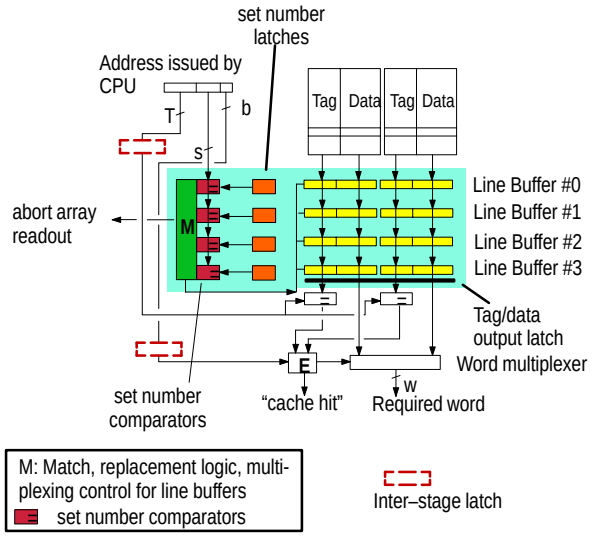


Figure 1. A 2–way Set–Associative Cache with Four Line Buffers

selector bits in the currently accessed address to the set number latch associated with this victim line buffer.

Cycle 2:

ϕ1: – Perform the tag comparison, as in a normal set–associative cache.

ϕ2: – Perform the word multiplexing on a cache hit, as in a normal cache.

In [KGM 97], Kin et al describe the use of a small "filter cache" that sits in front of a conventional L1 cache for reducing the power dissipation of the cache memory system. If a hit occurs in the smaller filter cache, data is accessed from the filter cache. The normal L1 cache access is started *only after a miss has been detected in this filter cache*. Consequently, the cache access latency is increased on a miss in the filter cache – this can have adverse impact on performance. Notice that although the multiple line buffers we have proposed behave as a fully associative cache and serves the same purpose as a filter cache, it does not impact the cache latency at all. This is because we probe the line buffers in parallel with the normal cache access. In the filter cache mechanism, the access time penalties can be avoided only when the filter cache has a sufficiently high hit rate – this requires the filter cache to be reasonably big (256 entries or more). Making the filter cache bigger increases its power dissipation and also takes a toll on performance, since the bigger size translates to a higher access time. Also, unlike a fully associative filter cache, we need only one set of tag comparators (which are the same ones that are needed in a conventional set–associative cache anyway) and four set number comparators to detect a line buffer hit. In contrast, in a four entry fully associative cache like the filter cache, four tag comparators would be needed. Thus, compared to the filter cache, the cache with multiple line buffers makes more effective use of the hardware without increasing the normal cycle time of the cache.

What is the number of line buffers that one should employ? Clearly, for reducing the number of accesses to the tag and data array, the hit ratio on the line buffers should be fairly high. This requires the use of a large number of line buffers, whose effective delays are likely to be higher. The delay within the match, replacement and multiplexing control logic are also likely to be higher, resulting in a critical path that prolongs the cache cycle time – an undesirable consequence. Furthermore, the power dissipations in the additional components will also increase, offsetting the power savings. In practice, as our subsequent results show, a small number of line buffers result in significant power savings, without any undue increase in the overall area of the cache and without any increase in the cache cycle time.

2.2 Subbanking

One of the major source of power dissipation in a conventionally organized cache can be attributed to transitions in the bit lines of the data and tag arrays. Bit line dissipations occur when the bit lines are precharged or discharged.

To achieve further savings on the bit line energy, the data arrays can be subdivided into subbanks, so that only those subbanks that contain the desired data can be read out [SuDe 95]. (To the traditional RAM designers, subbanking is known as column multiplexing, a technique that is used to reduce the number of sense amps.) A subbank consists of a number of consecutive bit columns of the data array. A data line is thus spread across a number of subbanks. Since data is read out from one subbank at a time, a common set of sense amps can be shared across the subbanks, cutting down on the cache area. In effect, columns are multiplexed within a subbank. Column multiplexing in this manner is routinely used within static RAMs. The size of a subbank refers to the physical word width of each subbank. Each subbank within a data array can be activated independently. By using an array of bit flags to indicate the presence/absence of subbanks in the line buffer, the array access stage can determine if a subbank needs to be read out for the current request. This again does not affect the cycle time nor the pipeline of the cache. At the same time, the readout of line data from subbanks which might never be needed is avoided. Figure 2 depicts a two–way set associative cache with two subbanks per data array.

2.3 Bit–line Segmentation

Bit line segmentation offers a solution for further power savings. The internal organization of each column in the data or tag array gets modified as shown in Figure 3. Here every column of bitcells, sharing one (or more) pair of bitlines are split into independent segments as shown. An additional pair of lines are run across the segments. (These lines are shown as a single line in Figure 3 (b).) The bit lines within each segment can be connected or isolated from these common lines as shown. The metal layer used for clock distribution can implement this line, since the clock does not need to be routed across the bit cell array. Before a readout, all segments are connected to the common lines, which are precharged as usual. In the meantime, the address decoder identifies the segment targeted by the row address issued to the array and isolates all but the targeted segment from the common bit line. This reduces the effective capacitive loading (due to the

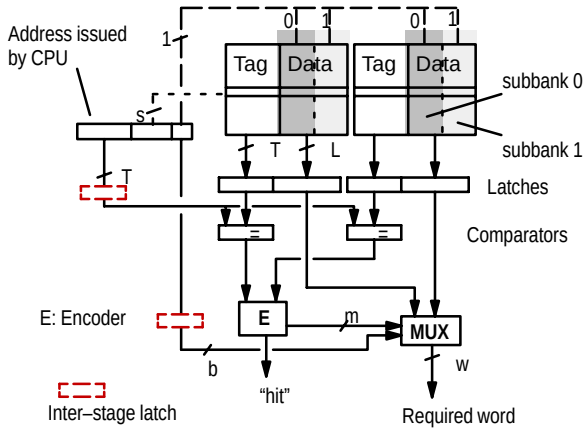
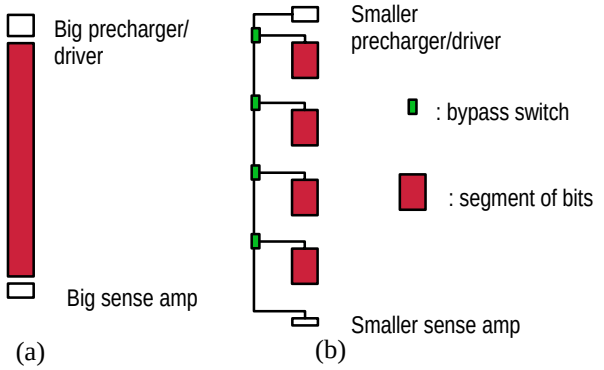


Figure 2. A 2–way Set–Associative Cache with 2 subbanks per data bank

diffusion capacitances of the pass transistors) on the common line. This reduction is somewhat offset by the additional capacitance of the common line that spans a single segment and the diffusion capacitances of the isolating switches. The common line is then sensed. Because of the reduced loading on the common line, the energy discharged due to a readout or spent in a write are small. Thus, smaller drivers, precharging transistors and sense amps can be used.



(a) original structure of column of bit cells

(b) segmented column of bitcells

Figure 3. Segmented data/tag array column

3. POWER ESTIMATION METHODOLOGY

We now describe the experimental setup used in our study of the energy efficiency achieved through the use of set associative caches with multiple line buffers, subbanking and bit line segmentation.

3.1 RTL Simulator for Processor & Caches

We use a detailed register–level simulator, SCAPE, which accurately simulates at the cycle level a superscalar pipelined processor, based on the MIPS instruction set. For the results presented here, we used a 3 level cache hierarchy, with split L1 caches (clocked at 300 MHz.), an unified L2 cache (running at half the CPU clock rate) and an unified L3 cache. The L1 and L2 caches were assumed to be on–chip and the L3 cache was assumed to be off–chip. Since superscalar CPUs need to fetch multiple instructions at a time to feed the instruction dispatch unit, the L1 I–cache was designed to supply multiple instructions per instruction fetch request. To improve the I–cache bandwidth, the I–cache was designed to have *even–odd* directories (and banks) so that requests that spanned consecutive rows could still be completed in the same cycle. The presence of multiple LOAD/STORE functional units in the superscalar pipeline demanded the use of multiple ports for the L1 D–cache.

Our simulations were for a 4–way superscalar CPU with 2 LOAD units, 1 STORE unit, 6 integer units, 2 Integer multiply/divide (pipelined) units and 2 Floating point (pipelined) units. We simulated the execution of the SPECInt95 and few SPECFp95 (su2cor, mgrid, applu) benchmarks to get a good mix of CPU–intensive and memory–intensive load.

3.2 Energy Dissipation Measures from VLSI Layout

For getting accurate measures of the dissipation in all major components of the cache organizations studied, we laid out the cache in a 4–metal layer, 0.5 micron technology and verified through SPICE simulations that it performed all major cache operations correctly. In particular, we sized critical components and made the bitcell arrays as compact as possible to make sure that a cycle time of 3.3 nsecs. (corresponding to a 300 MHz. clock rate) was sustainable [GhKa 99]. We also used SPICE to compute the energy dissipations in the cache components on major transition events, using a supply voltage of 3.3 volts. These measures included not only dissipations due to capacitive loading, but also leakage and short circuit currents (the later being a particularly dominating component in the sense amps).

The register level simulator generated transition counts within various cache components when the execution of the SPEC benchmarks were simulated. These were fed into a power estimation program that looked up appropriate energy dissipations for each event as obtained from the SPICE simulations. This power estimation program eventually produced a summary report of the power dissipations in various cache components as well as in the on–chip cache system as a whole.

3.3 Configurations Studied

For our base case, we assume a 32 Kbyte, direct–mapped L1 I–cache and a 32 Kbyte, 4–way set–associative L1 D–cache. The line sizes for both of these caches are set to 32 bytes, a typical number, with 16 byte subblock size. The L2–cache is assumed to be a 128 Kbyte, 4–way set–associative unified (i.e., shared by instructions and data) on–chip cache with a line size of 64 bytes and 32 byte subblock size. The 64 byte line size was chosen, since L2 caches must have a line size longer than that of the L1 caches to be effective. The off–chip L3–cache is assumed to be a 1Mbyte, 8–way set–associative unified off–chip cache with a line size of 128 bytes. The interconnection bus width was 32 bytes between L2 and L3, 16 bytes between L1 and L2 and 64 bytes between L3 and the

main memory. All the caches used write–back except L3 which was write–through with a 16 deep write back buffers. A buddy replacement algorithm, approximating LRU, was used for all the set–associative caches, as well as for choosing the victim line buffer on a line buffer miss.

4. RESULTS AND DISCUSSIONS

Figure 4 depicts the impact of subbanking on the base case and variations (with a smaller line size). In all cases depicted in this figure, the width of a subbank was maintained at 4 bytes. As expected, in the conventional organizations (i.e, organizations that do not use subbanks) power dissipation goes up with the line size. This is a direct consequence of activating the sense amps for all subbanks during a readout. The *relative* power savings are more substantial with subbanking for larger line sizes, since only the sensing of the enabled subbanks take place.

In Figure 5, we depict how power dissipations in the caches for the base configuration are reduced with the addition of line buffers. To avoid an explosion in the number of results to be depicted, if multiple line buffers were deployed, all caches (L1–I, L1–D and L2) used an identical number of line buffers. As the number of line buffers were varied from 1 to 8 (in powers of 2, although there is no reason not to have linear increases), the power requirements of each cache dropped, although non–linearly. Our power estimations took into account the dissipations due to the use of additional set number comparators, tri–stated line buffer latches and the longer interconnections. The decrease in power dissipation with the number of line buffers is as expected; the likelihood of accessing a recently accessed line held in a line buffer goes up as the number of line buffers increase. Beyond 8 to 10 line buffers, although the power dissipations continued to drop, the critical path in detecting a set number match followed by the time needed to steer the tag and data out of the matching line buffer increased beyond the targeted cycle time of 3.3 nsecs. Not much of an improvement resulted in the critical path despite sizing of the tri–state drivers in the line buffers or the pull–down transistors in the set number comparators. The key observation is that with the use of a small number of line buffers (8 in this case), a reduction of 40% to 50% in the dissipations of the individual caches and the total for all on–chip caches is easily achieved without compromising the cache cycle time.

The impact of segmenting the bit lines on the power dissipation of the caches is shown in Figure 6. (To avoid an explosion in the number of results, the number of segments were kept identical in all the caches.) An interesting variation in the degree of power savings is seen among the caches in this figure. Increasing the number of segments in the L1 I–cache or the L2 cache does not result in the kind of power savings that are realized for the L1 D–cache. For the L1 I–cache, which is directly mapped, this can be explained as follows. Because of the direct mapping (which is a one way set–associative cache), the length of the bit lines in the L1 I–cache is considerably longer than the length of the bit wires for the 4–way L1 D–cache. Consequently, the precharge transistors needed for the L1 I–cache are bigger than the ones

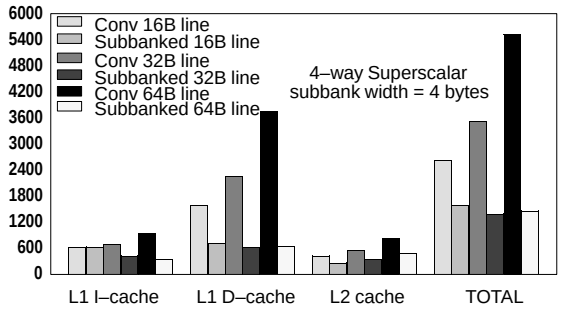


Figure 4. Power dissipation in caches with subbankng (in milliwatts). Cache sizes & associativities correspond to the base configuration (see text), but other parameters are varied as shown.

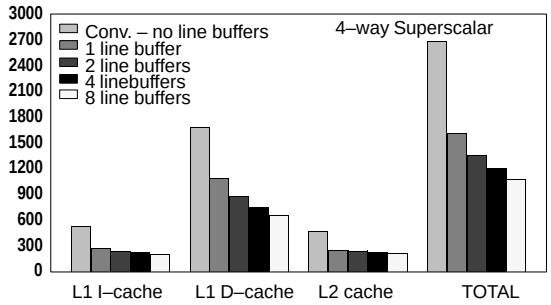


Figure 5. Power dissipation in caches with line buffering (in milliwatts). Cache parameters correspond to the base configuration (see text), but number of line buffers are varied as shown.

needed for the L1 D–cache. Consequently, the diffusion capacitance of the bigger precharge transistors dominates the bit line capacitance. The effective capacitance of the bit line during sensing and precharging depends less heavily on the length of the bit line segments. A similar explanation is valid for the L2 cache, which also has longer bit lines because of its larger capacity. In any case, the use of 8 to 16 segments result in a savings of 25% to 30%+ in the overall on–chip cache dissipations.

The power savings realized when subbanking, bit–line segmentation and multiple line buffers are all employed is shown in Figure 7. In all of these variations, 16 bit line segments are used. Clearly, the overall savings is not the sum of the savings achieved using each technique on its own. This is obvious from the manner in which the various techniques interact with each other. Line buffering using multiple buffers and subbanking are the two techniques that result in the most power savings. Bit line segmentation saves a smaller fraction of the resulting power when either or both of these two techniques are deployed. No matter what, with the use of all of these techniques (with 8 line buffers), the total dissipation of the on–chip caches come down from about 2690 mWatts (for the conventionally organized caches) to about 624 mWatts, representing a staggering power savings of about 75%.

Figure 8 shows a breakdown of the energy dissipations within major components of the L1 Icache and L1 Dcache and the L2 cache of the base case with subbanking, 8 line buffers and 16 bit line segments. These figures show that bitline and sense

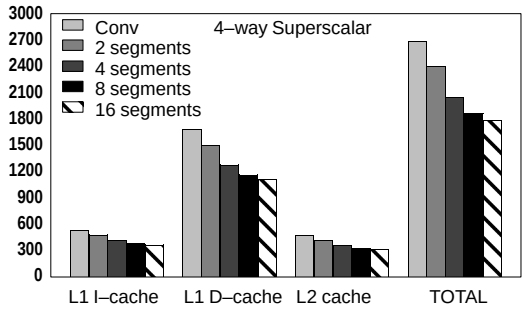


Figure 6. Power dissipation in caches with bit line segmentation (in milliwatts). Cache parameters correspond to the base configuration (see text), but number of bit line segments are varied as shown.

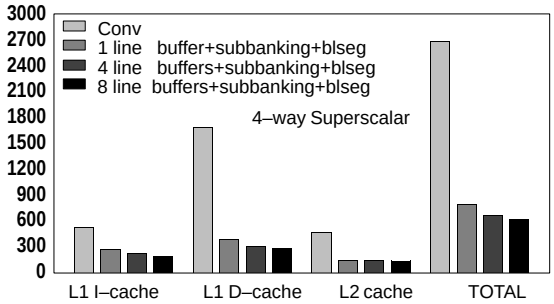


Figure 7. Power dissipation in caches with subbanking, multiple line buffers and bit line segmentation (in milliwatts). Cache sizes, associativities and line sizes correspond to the base case.

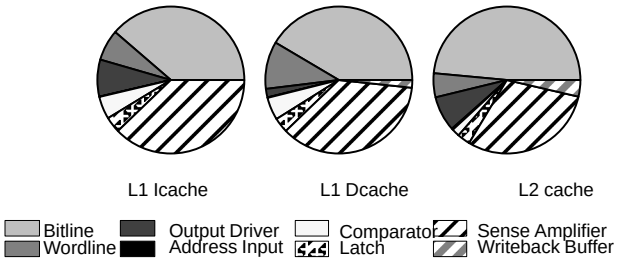


Figure 8. Components of the power dissipation in individual caches with subbanking, 8 line buffers and 16 bit line segments (in milliwatts). Cache sizes, associativities and line sizes correspond to the base case. Address decoder power is part of the "address input" component.

amp dissipations are the most dominant component of the total power dissipation. One should be careful not to compare the absolute "slice" represented by the bitline power dissipation in the pie charts for the conventional and the line buffered caches since the total power dissipations for these caches are widely different. Latches and comparators, as expected, also represent a non negligible component of the overall power dissipations in the caches with line buffers. Although not shown here, the bit line and sense amp dissipations form an even bigger fraction of the overall power dissipation for the conventional organizations, as one would expect.

5. CONCLUSIONS

On–chip caches are a major source of power dissipation in contemporary superscalar microprocessors. The bulk of the energy dissipated in conventionally organized caches is in precharging, sensing and discharging the bit lines of the tag and data arrays. We demonstrated that an alternative cache organization with multiple line buffers, subbanking and bitline segmentation can be very effective in reducing the power dissipation in on–chip caches. This organization does not compromise the cache cycle time (and other aspects of performance, such as the cache hit ratio and cache access rate). The maximum power saving achieved using this organization was about 75%, suggesting that the proposed cache architecture to be a viable choice for modern CPUs.

References

[ERB+ 95] Edmondson, J. F. et al, "Internal Organization of the Alpha 21164, a 300–MHz 64–bit Quad–issue CMOS RISC Microprocessor", Digital Technical Journal, Vol. 7, No. 1, 1995, pp. 119–135.

[EvFr 95] Evans, R. J. and Franzon, P. D., "Energy Consumption Modeling and Optimization for SRAM's", IEEE Journal of Solid–State Circuits, Vol. 30, No. 5, May 1995, pp 571–579.

[FrPe+ 97] Fromm R., et al., "The Energy Efficiency of IRAM Architectures", in Proc. of the 24th International Symposium on Computer Architecture, June 1997.

[GhKa 99] Ghose, K. and Kamble, M. B., "A 0.5 micron Cache and Its Low Power Variants", Technical Report CS–TR–99–2, Dept of Comp. Sci., SUNY–Binghamton, January 1999.

[Gro 90] Grohoski, G.F., "Machine Organization of the IBM RISC System/6000 Processor", IBM Jrnl. of Resaerch and Development, Vol. 34, No. 1 (Jan. 1990), pp. 37–58.

[Handy 93] Handy, J., *The Cache Memory Book*, , Academic Press, 1993.

[HKY+ 95] Hasegawa, A. et al, "SH3: High Code Density, Low Power", IEEE Micro magazine, Dec. 1995, pp. 11–19.

[Itoh 96] Itoh, K., "Low Power Memory Design", in *Low Power Design Methodologies*, ed. by Rabaey, J. and Pedram, M., Kluwer Academic Pub., pp. 201–251.

[KaGh 97] Kamble, M. B. and Ghose, K., "Energy–Efficiency of VLSI Caches: A Comparative Study", in Proc. IEEE 10–th. Int'l. Conf. on VLSI Design, Jan. 1997, pp. 261–267.

[KBN 95] Ko, U., Balsara, P. T. and Nanda, A.K., "Energy Optimization of Multi–Level Processor Cache Architectures", in Proc. of the Int'l. Sym. on Low Power Design, 1995, pp. 45–49.

[KGM 97] Kin, J., Gupta, M. and Mangione–Smith, W.H., "The Filter Cache: An Energy–Efficient Memory Structure", in Proc. MICRO 30, 1997, pp. 184–193.

[Mon 96] Montanaro, J. et al., "A 160 MHz, 32b 0.5 W CMOS RISC Microprocessor", in IEEE ISSCC 1996 Digest of Papers, 1996.

[Smith 82] Smith, A. J., "Cache Memories", ACM Computing Surveys, Sept. 1982, pp. 473–530.

[SuDe 95] Su, C. and Despain, A., "Cache Design Tradeoffs for Power and Performance Optimization: A Case Study", in Proc. of the Int'l. Sym. on Low Power Design, 1995, pp. 63–68.