Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Systems Architecture 51 (2005) 653–664
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

A low energy cache design for multimedia applications exploiting set access locality

Jun Yang [a,*], Jia Yu [a], Youtao Zhang [b]

[a] Computer Science and Engineering Department, University of California, Riverside, CA 92521, United States
[b] Computer Science Department, University of Texas at Dallas, Richardson, TX 75083, United States

Received 7 February 2004; received in revised form 24 August 2004; accepted 17 February 2005
Available online 20 April 2005

Abstract

An architectural technique is proposed to reduce power dissipation in conventional caches. Our technique is based on the observation of cache access locality: current access is likely to touch the same cache set including the tags as the last access. We show that considerable amount of power driving the cache tag and data banks can be saved if this cache access locality is fully exploited. This is achieved through buffering and accessing the last accessed cache set instead of driving the tag and data banks. Unlikely previous designs, our technique does not incur performance degradation. Experimental results carried out on 8 KB/16 KB/32 KB data and instruction caches have respectively shown 31%/35%/36% and 51%/58%/66% power savings.

Keywords: Low power design; Cache; Set buffer; Multimedia

1. Introduction

As the speed gap between the memory and CPU continues to increase, modern processors tend to enlarge their on-chip caches in order to reduce the number of accesses to long latency memories. For this reason, the on-chip caches have been a major consumer of total chip power. As an example, the Intel Pentium Pro dissipates 33% and the StrongARM 110 dissipates 42% [15] of its total power in caches. Consequently, there have been increasing interests in designing low-power on chip caches.

A variety of cache designs are being developed to conserve power [1,18,5,11,6,7]. The first category proposed to reduce dynamic switching energy. Such techniques include filtering mechanisms [11], reconfiguring the cache—including its size and associativity—according to the needs of

* Corresponding author. Tel.: +1 951 827 2558; fax: +1 951 827 4643.
E-mail address: junyang@cs.ucr.edu (J. Yang).

1383-7621/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2005.02.003

an application program [1,18], compression techniques [5], and way prediction [4,17]. Another category saves static leakage energy. Such techniques include lower Vdd [22], higher threshold voltages [2], leakage-biased bitlines [7]. Our technique belongs to the first category—saving dynamic power.

The multimedia applications have become an important workload for general purpose processors nowadays. Previous research have shown the importance of designing low-power caches and memories running multimedia applications [20,9,14]. In the Cool-Cache architecture developed by Unsal et al., the compiler helps remap scalars to smaller scratchpad area and direct accesses to a tagless cache to achieve energy savings. Moshnyaga et al. proposed to dynamically adjust memory bit-width per pixel to reduce video memory power consumption. Finally, Inoue et al. developed a history-based tag comparison elimination technique that reduces the power expenditure in the tag array.

In [21], we proposed a low complexity design that can save considerable amount of power by exploiting the abundant cache access locality for multimedia applications. We found that multimedia applications present high *set-wise access locality* (SAL)—a set is continuously accessed for a period of time. Take a direct-mapped cache as an example, the SAL is reduced to line access locality where a line is accessed by successive cache accesses. For set-associative caches, a set contains multiple lines so the SAL is slightly higher than that of a direct-mapped cache. Fig. 1 shows the set-wise access locality for a 32 KB data cache with different set-associativities running MediaBench benchmarks [12]. On average, we observed around 37% of total cache accesses hit in the last accessed set for associativities ranging from 1 to 32.

With this feature, we proposed the *lightweight* set buffer ([21]) for saving power in data caches. The set buffer stores the last accessed cache set including tag and data. It is lightweight because we made use of existing latches inside the cache with a group of set status bits. When the current access hits the same set as last access, the driving of the entire tag and data array is gated off by the status bits saving significant power. The status bit

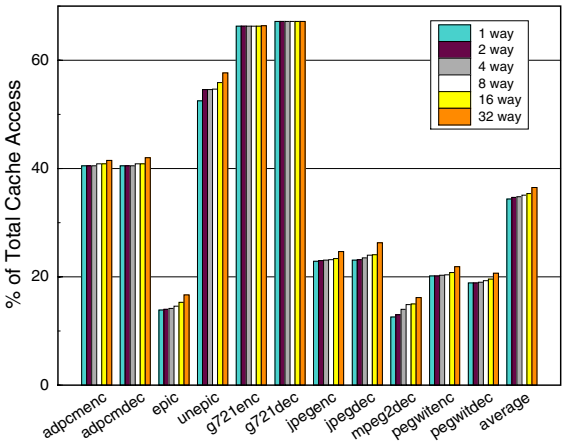


Fig. 1. Percentages of accesses that hit in last accessed cache set (linesize = 32 B).

simply indicates if a set is last accessed or not. A zero bit will lead access to the set buffer and gate off the tag and data array driving. This technique does not slow down the cache access, nor does it require excessive amount of extra hardware.

In this paper, we extend the work in [21]. Compared to [21], we evaluate the power savings of the design in both data and instruction caches. Our experimental results show that the design is effective for both types of caches while with more power savings for instruction caches. On average, there are 31%/35%/36% power savings achieved on 8 KB/16 KB/32 KB set associative data caches respectively, and 51%/58%/66% power savings for 8 KB/16 KB/32 KB instruction caches respectively. Furthermore, we also evaluate the design on a subset of SPEC benchmark programs to see its general applicability to non-multimedia applications. The results show that due to less *set-wise access locality* in these programs, there are comparable power savings in the instruction cache but not the data cache. We observe that on average, 19%/25% power savings are achieved on 8 KB/32 KB data caches respectively.

The rest of the paper is organized as follows. Section 2 reviews previous research in line buffering. In Section 3, detailed design of the proposed set buffer is illustrated. Section 4 gives the energy model of our design. The experimental results are

presented in Section 5. Finally, Section 6 concludes this paper.

2. Related research

The notion of line buffer has previously appeared in literature [19,6]. Su and Despain proposed in-cache two-level hierarchies in which a single line buffer, serving as the first level "cache", is accessed before the main cache [19]. This design is essentially a single-entry filter cache [11] within the original cache. Consequently, a line buffer miss requires additional cycles to access the main cache, degrading the overall program performance. Moreover, the overall energy consumption may increase if the line buffer miss rate is high. To overcome those defects, Ghose and Kamble [6] introduced a concurrent version of line buffers in which cache lines in the same set are organized in one wider line buffer (WLB), and they keep multiple such buffers. Buffering multiple WLB aims at improving WLB hit rate and concurrent accessing with the main cache does not impact performance. The proposed WLB is effectively a fully-associative cache placed on the side of the level one data cache. Both caches are inquired simultaneously, and a hit in WLB cancels the on-going access in the main cache assuming that the former is resolved earlier. The major overhead of this design come from (1) the power dissipated in the (fully-associative) WLB on buffer misses, including miss detection and the line replacement, (2) the power spent in initial main cache driving on WLB hits, and (3) the comparisons of *both* set_index and tag in WLB on each potential hit.

The deficiency in both of the above techniques lies in the possibility of increasing total power when the line buffer or WLB hit ratio is low. This calls for an ultra low-overhead buffering technique that reduces power consumption when the hit ratio is high, and barely increases power when the hit ratio is very low. Our proposed lightweight set buffer design fits into this category. We attach an extra single bit to each cache set to indicate whether or not this set is last accessed. Using this bit, we perform a concurrent gated probing to the cache and set buffer—the bit automatically chooses the access between the main cache and the set buffer. This implies that in the worst case, the bit guides every access to the main cache and we pay only the power maintaining each bit, which turns out to be almost negligible. As we will explain in Section 3, this design does not incur performance degradation either.

3. The lightweight set buffer

For a cache access in a conventional *m-way* set-associative cache, the *set_index* portion of the address is first extracted to index *m* tag and the data arrays. Next, the *m* tags and cache lines are read out and steered into *m* local buffers waiting for tag-compare results. If it is a cache hit, the target word will be selected out from one of *m* data line buffers and returned to the CPU. Notice that if the next cache access hits in the same set, the entire procedure repeats and the same cache set is driven into the tag and line buffers again. The development of lightweight set buffer stems from this observation and incorporates minimum amount of hardware to remove unnecessary activities. Our set buffer is merely the collection of those already existing buffers for holding tags and cache lines.

3.1. Proposed architecture

3.1.1. The LAB Logic

Fig. 2 depicts the logic design of the lightweight set buffers to a 4-way set associative cache. To be clear, we omit the decoder logic, tag compare and select, and data output driving logic since they comply with standard implementations. The shaded boxes in the graph represent multiple cache ways and we focus our description on only a single way since the rest are repetitive. All the newly added lines, gates and bit arrays are highlighted in bold lines.

An additional bit—latest accessed bit (LAB) is added to each set to indicate whether the set is latest accessed or not. The LAB is implemented as a toggle flip-flop. Each LAB bit stores either a "0" or a "1" representing last-accessed-true or -false. The use of LAB is combined with a wordline

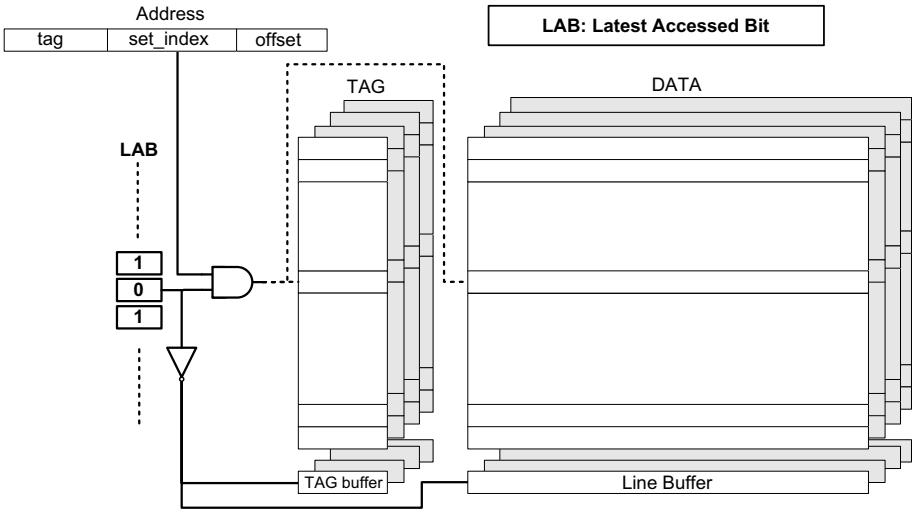


Fig. 2. Logic design of lightweight set buffer utilizing a set status bit.

driving of both tag and data arrays. If the LAB is unset, meaning that it was accessed last time, the corresponding cache set should already have been steered into the set buffer. Therefore, the wordline driving is gated off as shown in dashed lines in Fig. 2. Meanwhile, the set buffer is driven to get ready for tag comparison and data access, as shown in solid lines in the graph.

On the other hand, by the time a LAB determines driving the set buffer, the cache bitlines may have been precharged already. Since the entire data arrays and tag arrays are not activated, the discharging can be avoided for this access. Thus, the next cache access does not need to precharge bitlines again. If the next access goes to the cache arrays, the bitlines are already charged in the current access. If the next access goes to the set buffer, the cache array bitlines remains charged. In other words, the current hit in the set buffer saves bitline precharge power for the next and future accesses.

Another issue is that the data buffered in the sense amplifiers may also be destroyed by precharging. To preserve the stored data, this precharging should be disabled if the LAB corresponding to the current set is "0", i.e. current set is the same as the last set, and enabled when the bit is "1". In other words, precharging the sense amplifiers depends on one of the LABs that is known only after decoding the index. From running CACTI [10], we found that the precharging time of sense amplifiers is short enough to get them ready for storing the new data line, even after decoding. Thus, through selective precharging, the sense amplifiers can lock the data line when they are repeatedly accessed.

Fig. 2 shows only the *logic* design of the LAB controlling tag and data arrays. In reality, the AND gate can be combined with the wordline driving circuit. For example, Fig. 3 illustrates what can be modified on the path from the decoder to wordline driving. The upper portion is an example of the gates between the end of the decoder, an NOR gate, to the beginning of a wordline, an inverter serving as a wordline driver [10]. To incorporate the LAB signal, one solution is to replace

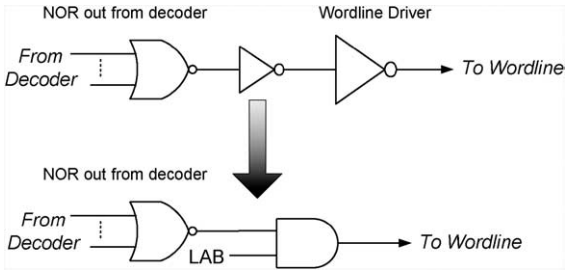


Fig. 3. Circuit changes to wordline drivers.

two inverters with an AND gate. The AND gate now serves as the wordline driver and should be sized according to the wordline length to meet the target rise time. As we can see, when the LAB signal is low, the AND gate will not conduct and the tag and wordlines will not be driven. the modifications required in the cache control circuits is minimal.

3.1.2. Only one LAB can store "0"

Since the LAB is used to indicate if it is the last set accessed, at anytime, there can be one and only one LAB that is storing "0", all the remaining LABs are storing "1".

To ensure the above property, one must restore (set to "1") the bits that were zeroed before the last access. This is achieved by remembering the *set_index* portion of the last but one address. If the last access hit an LAB that is "0", no action needs to be taken since the cache set is already marked and should stay marked for the next time. If it is "1", use the stored *set_index* to restore its corresponding LAB and zero the current LAB using the current *set_index*. On a cache miss, all the LABs should be re-initialized to "1" (there is only one LAB needs to be restored) and we do not attempt to reflect the cache line fill in the set buffer. The register to store the old *set_index* is initialized to zero, i.e. it points to the first set. Since the set is invalid when the execution starts, the invalid bit prevents the possible misuse of old *set_index* for the first iteration.

3.1.3. Power overhead

The LABs and the necessary additional register for *set_index* bring only marginal extra power. A flip-flop consumes less than 10 fJ (10×10^{-15} J) on every flipping [13]. Incorporating this figure, we used the power and timing evaluation tool XCACTI [8] with 0.18 μm technology. We measure the additional power dissipation due to LAB and other necessary hardware on updates. A LAB update happens when a currently accessed cache set is different than the last one. In this case, both LABs are flipped on the update. Moreover, an additional decoding for saving and comparing the set index of the old LAB. Taking all the above into account, Fig. 4 tabulates a LAB update power

	8KB	16KB	32KB	64KB
1-way	1.9	1.8	2.2	1.4
2-way	1.7	1.5	1.1	2.7
4-way	1.0	0.9	1.5	1.1
8-way	0.4	0.6	0.7	0.5
16-way	0.4	0.4	0.2	0.5
32-way	0.3	0.3	0.3	0.2

Fig. 4. Power consumed by updating the LABs including decoding the old *set_index* and storing *set_index* in buffers (in percentage of original cache).

in percentage of the original cache power. Most of the numbers are around or well below 2%. And configurations with more bits in *set_index* tend to have higher percentage of LAB power. Also note that LAB update happens only when it is "1" and on cache misses. Therefore, the accumulative power overhead due to LAB updates has less impact on power savings than the data shown here. We will present the overall power savings in Section 5.

3.1.4. Timing the access

The LAB update operation should be done in a timely manner. This is important since the next cache access may come right next clock cycle, which means the LAB should get ready for the new request by then. This can be ensured by updating the LAB as soon as "1" is detected since from this point on the cache behaves normally. Thus, updating LAB is carried in parallel with tag and data array driving and can be finished before the current cache access is over. On a cache miss, however, the miss detection will not reveal until near the end of the access, therefore updating the LAB can only be done after the current access. However, this will not impact the performance since the next cache access will not benefit from the current miss anyway (we do not attempt to reflect the cache refill in the set buffer). Therefore, our solution is that on every cache access that follows a miss, the LAB is used solely for updating.

The LAB does not introduce delay on the tag path which sometimes can be the critical path in a cache access. The tag path consists of decoding

the *set_index*, reading the tag arrays, compare the tags and drive the output data. From Fig. 2, the only gate "added" to the tag path is the AND gate. However, as illustrated in Fig. 3, such an AND gate is fused into the original tag path. The internal transistors can be resized without additional delay. At the same time, The LAB signal is available continually from the flip-flop output. Therefore, the tag and data array wordline driving can start as soon as the decoding signal is steady from the decoder output. From the above analysis, we can see that the modification to the tag path does not impact on its delay, and hence the cache access time (max(data path, tag path)).

3.2. Managing write operations

The write operations need to proceed with care. This is because a line buffer becomes dirty on a write hit. The updates in the line buffer will be present in the data array sooner or later. There are two approaches to the timing of the updates:

- *Write-through:* Every write hit in line buffer is also written in data array.
- *Write-back:* Only when a new line (including a different line in the same set or a different set) is inquired, does the dirty line buffer update the data array.

There are obvious tradeoffs between the two designs. The write-through version has no impact on performance but does not benefit as much as the write-back version. The latter saves power in a more aggressive way since multiple writes can be coalesced in a single write back, but may suffer from performance loss. This is because a new access may need to wait until the write-back finishes. This can be overcome through adopting the pipelined writes mechanism used by Alpha AXP 21064 and other machines [16]. On such pipelined writing, a line buffer first copies the line into a *write delay buffer* and the actual write-back takes place when next time a new write operation is comparing its tag. In other words, on every write access, the cache always perform tag compare for current request but writes data from the write delay buffer. Therefore, this approach does not introduce extra cycles on cache accesses. The only overhead in this approach is the extra write delay buffer. Note that on every read access, this write delay buffer needs to be inquired. To see the power budget of the delay buffer, we measured it using XCACTI tool for various cache configurations. Fig. 5 shows the results in permillage of a normal cache. For all the configurations we tested, the extra power is within two permillage of the original cache. Thus, having the write delay buffers will help reduce the overall power consumption. In Section 5, we will adopt this approach and present the measurements.

read,write	8KB	16KB	32KB	64KB
1-way	1.5,1.7	1.1,1.2	0.7,0.7	0.7,0.8
2-way	1.2,1.3	1.5,1.6	1.1,1.1	0.7,0.7
4-way	0.5,0.6	1.7,1.8	1.3,1.3	0.6,0.6
8-way	0.5,0.6	1.3,1.3	0.4,0.4	0.3,0.3
16-way	1.7,1.7	1.4,1.4	0.3,0.3	0.8,0.8
32-way	0.9,0.9	0.8,0.8	0.7,0.7	0.2,0.2

Fig. 5. Additional power consumed by write delay buffers (in permillage of original cache).

4. Energy modeling

In this section, we describe how the power is modeled for our lightweight set buffer design. We categorize different cache operations in Fig. 6. As we can see, beyond normal cache read/write hits, we need to further distinguish between: (1) hits and misses in a set buffer, and (2) accesses following or not following a cache miss as they consume different amount of energy, especially in using LAB as described in Section 3.1. We list the energy equations in each category in form of amount being added and subtracted, using "+" and "−" respectively, along with standard cache energy consumptions. The baseline energy includes writing from the write delay buffer as described in Section 3.2. Essentially, we save energy when it is a LAB hit, and need to charge the LAB update energy for updating LABs otherwise.

previous $ access	current $ access	LAB access	power consumption description
hit	hit	hit(=0)	−: Energy in driving the entire tag and data array.
		miss(=1)	+: Energy in updating the LAB bits.
	miss	hit(=0)	−: Energy in driving the entire tag and data array.
		miss(=1)	+: Energy in updating the LAB bits.
miss	—	miss(=1)	+: Energy in updating the LAB bits.

Fig. 6. Power consumption components.

5. Experimental results

We implemented our proposed lightweight set buffer technique in an execution driven simulation tool SimpleScalar [3] with cache power model extension XCACTI [8]. We evaluated our design through running a subset of MediaBench benchmark [12] suite (currently there are 11 programs that are running correctly in our system), then we used XCACTI to calculate the energy saving on both L1 data and instruction cache. Without loss of generality, we varied the cache size and set-associativity to cover a range of realistic cache configurations. In order to show the advantage of our design, we also compare the energy saving with the WLB approach.

5.1. Energy saving in data caches

In this set of experiments, we used the following cache parameters for L1 data cache: 8 KB direct-mapped, 16 KB 2-way set-associative, and 32 KB 4-way set-associative, all with 32 B line size. Our purpose is to compare with the WLB design [6] and see if our lightweight set buffer is more efficient.

Our first set of experiments measures the power reduction percentage for both designs. The results are shown in Fig. 7 for three cache configurations. For all the configurations we tested, our lightweight set buffer outperforms the WLB except epic. Epic has very low set-wise access locality (Fig. 1), so buffering 1 set is not enough to save much

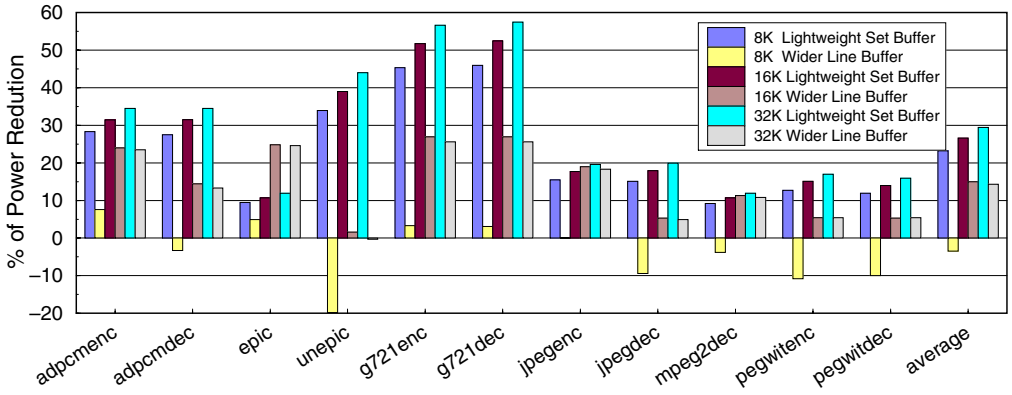


Fig. 7. Overall power reduction compared with WLB.

power. WLB buffered more data, which potentially improves hit rate and hence saves more power for epic benchmark. On average, we achieve 31%/35%/36% power savings for 8 KB/16 KB/32 KB cache respectively. We observed that with about the same amount of set buffer hit ratio (Fig. 1), highly associative caches tend to benefit more since their total number of sense amplifiers used in the data arrays is higher. The sense amplifiers in the caches usually consume the bulk of overall power [8]. Reducing the tag and data array activity directly translates to reducing sense amplifier power. Therefore, the set buffer are more effective for highly associative caches.

The WLB design works well for medium or large cache sizes as we can see from the figure. For small direct-mapped cache, it increases the overall power because it is not worth complicating the logic and controls in small simple structured caches. The WLB in these caches is simply an overkill. On average, the WLB design reduces 14% and 13% for 16 KB and 32 KB cache respectively, but increases the 8 KB cache power consumption by 4.9%. From our experiments, the WLB hit ratio is about twice as much as our set buffer. However, the power savings is lower than our design because of the over complex design. We attribute this results to the "simpler is better" rule.

Fig. 8 further shows the contribution of data and tag array in the energy reduction separately. The data array alone saves on average 23%/26%/29% for caches of three sizes. The tag arrays ac-

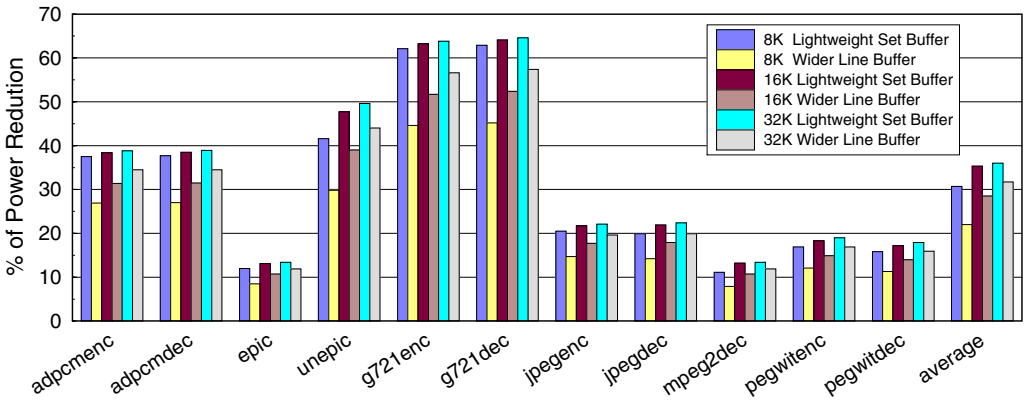


Fig. 8. Tag and data arrays' contribution to power reduction.

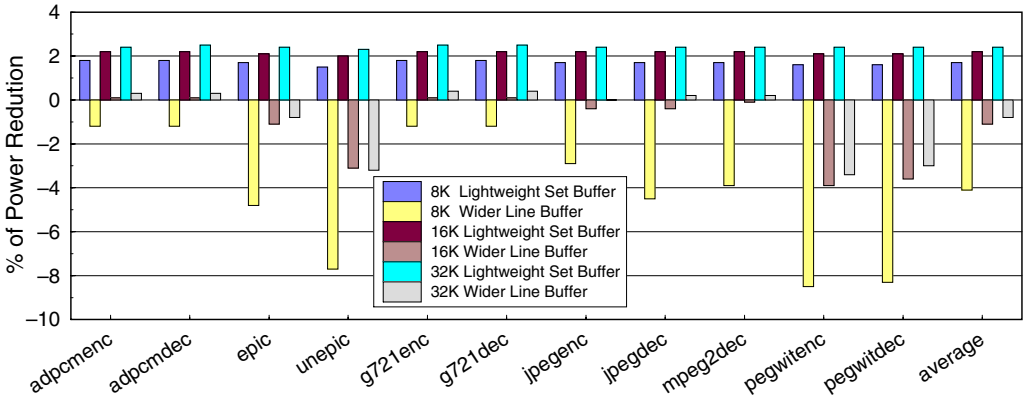


Fig. 9. Power variations when set buffer/WLB hit ratio is low.

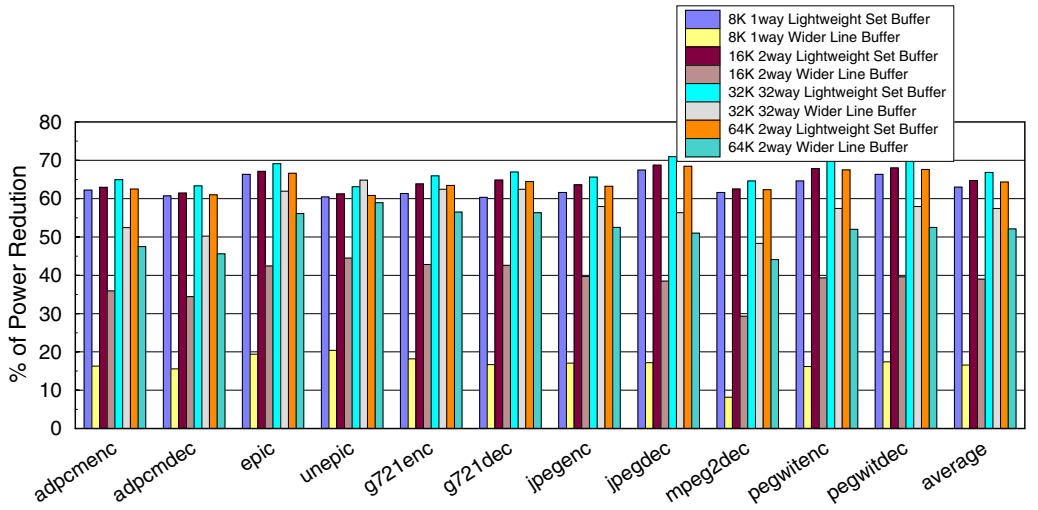


Fig. 10. Power reduction for instruction caches.

count for 4% to 7% additional power savings. This amount is proportional to the power consumed by the tag arrays in the entire data cache, which we estimated is about 6%.

Our lightweight set buffer design has another valuable feature in that it does not increase the overall power consumption even when the set buffer hit ratio is very low. However, the WLB is not as fortunate. To verify this observation, we performed another set of experiments in which we intentionally preset the set buffer and WLB hit ratio a low number, e.g. 3% of total cache accesses in which 1% are write hits. The results are shown in Fig. 9. Not to our surprise, the WLB data for almost all the benchmarks are negative. While in our design, we keep a minor but steady power saving percentages, 1.8%, 1.1%, and 0.6% for 8 KB, 16 KB, and 32 KB respectively. The LAB overhead as shown in Fig. 4 is so low that its effect is not even noticeable across different benchmarks.

We believe the above feature is very appealing since this results a *safe* low power cache design. In such a scenario, the cache saves significant amount of power while the program presents high set-wise access locality. On the other hand, the cache can seamlessly turn to a "power safe mode" where even low set-wise access locality can yield some amount of savings.

5.2. Energy saving in instruction cache

For instruction caches, it is also feasible to adopt our design since the instructions present very good set-wise access locality. We expect to save more power in instruction caches than data caches using our design.

In this set of experiments, we take instruction cache configurations that are adopted by existing processors: 8 KB direct-mapped (Alpha 21164),

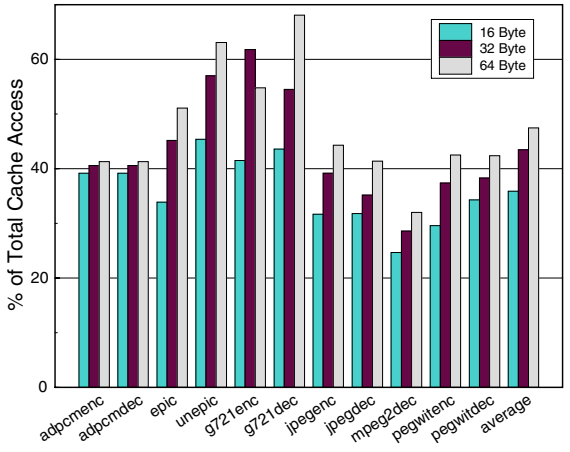


Fig. 11. Effect of line size on set-wise access locality (32 KB, 4-way set-associative).

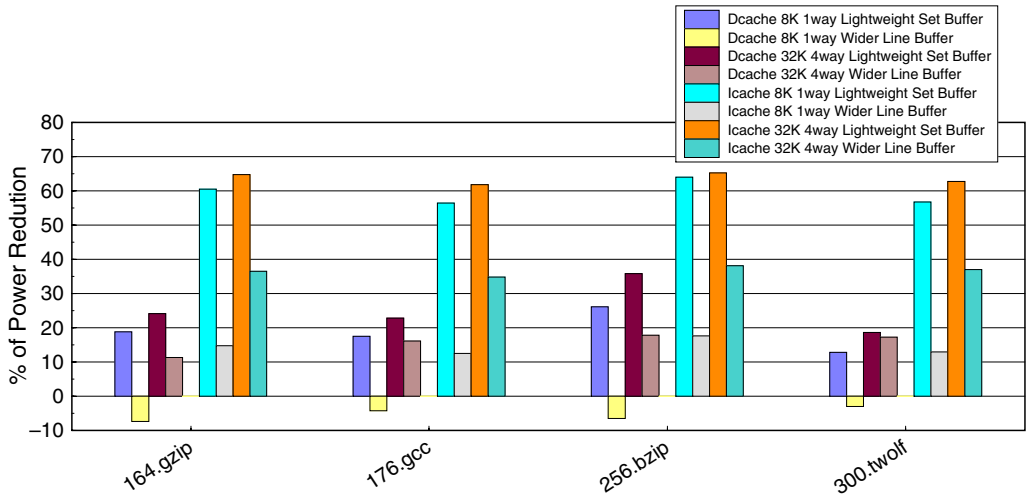


Fig. 12. Effectiveness on SPEC benchmarks (32 KB, 4-way set-associative).

16 KB 2-way set-associative (IBM PowerPC 405), 32 KB 32-way set-associative (Intel XScale), and 64 KB 2-way set-associative (AMD Athlon). The results are shown in Fig. 10. On average, we achieve 51%/58%/66%/60% power savings for 8 KB/16 KB/32 KB/64 KB caches respectively. The primary reason for this significant saving is the high hit ratio in the set buffer (around 60% for all the benchmarks) which is a nature feature of instruction cache accesses.

It is worth noting that the energy saving in I-cache is not only affected by cache size, but also set-associativity. In Fig. 10, we observed that 32 KB 32-way cache saves the most power, while 16 KB and 64 KB 2-way cache stand at the second tier, finally the 8 KB 1-way cache. This is because when the cache set associativity is increased, the number of lines contained in a set is increased and thus the set-wise access locality. This shows our design is an effective alternative among various low-power techniques for high associativity caches.

5.3. Impact of line sizes

The set-wise access locality is a function of not only the cache set associativity, but also the cache line sizes. Fig. 11 plots this metric with different line sizes for 32 KB, 4-way set-associative caches. We can see that for most programs tested, the last cache set hit rate increases with line size. This is intuitive since larger line sizes increase set-wise access locality. However, we did not choose large line sizes since the cache may be ill-configured which might increase cache miss ratio.

5.4. General purpose benchmarks

We further evaluated the design for general purpose benchmark programs. We chose four SPECInt 2000 benchmark programs and collected the power saving results for both data and instruction caches (Fig. 12). As expected general purpose programs have less *set-wise access locality* which results in less overall power savings. As an example, for a data cache with 32 KB and 4-way set associativity, we achieved around 25% power savings for these programs while on average 36% for multimedia programs.

6. Conclusion

In this paper, we designed a lightweight set buffer in data cache and instruction cache to achieve favorable power savings without performance degradation. The proposed technique works well for

multimedia applications that have high set-wise access locality. Compared to previous approaches, our technique requires much less hardware overhead yet still yields better results. Moreover, the lightweight set buffer does not over-spend power even when the set-wise access locality is low. This is a accomplishment that could not be achieved previously.

References

[1] D.H. Albonesi, Selective cache ways: on-demand cache resource allocation, Journal of Instruction-Level Parallelism 2 (2000).

[2] M. Allarm, M.H. Anis, M.I. Elmasry, High-speed dynamic logic styles for scaled-down CMOS and MTCMOS Technologies, in: ACM/IEEE International Symposium on Low Power Electronics and Design, 2000, pp. 155–160.

[3] D. Burger, T. Austin, The SimpleScalar Tool Set, Version 2.0, Technical Report 1342, University of Wisconsin-Madison, Computer Science Department, 1997.

[4] B. Calder, D. Grunwald, J. Emer, Predictive sequential associative cache, in: The 2nd IEEE Symposium on High-Performance Computer Architecture, 1996, pp. 244–253.

[5] R. Canal, A. Gonzalez, J.E. Smith, Very low power pipelines using significance compression, in: The 33rd Annual IEEE/ACM International Symposium on Microarchitecture, 2000, pp. 181–190.

[6] K. Ghose, M.B. Kamble, Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation, in: ACM/IEEE International Symposium on Low Power Electronics and Design, 1999, pp. 70–75.

[7] S. Heo, K. Barr, M. Hampton, K. Asanovic, Dynamic fine-grain leakage reduction using leakage-biased bitlines, in: The 29th Annual International Symposium for Computer Architecture, 2002, pp. 137–147.

[8] M. Huang, J. Renau, S.M. Yoo, J. Torrellas, L1 Data cache decomposition for energy efficiency, in: ACM/IEEE International Symposium on Low Power Electronics and Design, 2001, pp. 10–15.

[9] K. Inoue, V.G. Moshnyaga, K. Murakami, A history-based I-cache for low-energy multimedia applications, in: ACM/IEEE International Symposium on Low Power Electronics and Design, 2002, pp. 148–153.

[10] N.P. Jouppi, S.J.E. Wilton, An enhanced access and cycle time model for on-chip caches, Research Report 93/5, Compact Western Research Lab, July 1994.

[11] J. Kin, M. Gupta, W.H. Mangione-Smith, The filter cache: an energy efficient memory structure, in: ACM/IEEE 30th International Symposium on Microarchitecture, 1997, pp. 184–193.

[12] C. Lee, M. Potkonjak, W.H. Mangione-Smith, MediaBench: a tool for evaluating and synthesizing multimedia and communications, in: ACM/IEEE 30th International Symposium on Microarchitecture, 1997, pp. 330–335.

[13] R.P. Llopis, M. Sachdev, Low power, testable dual edge triggered flip-flops, in: ACM/IEEE International Symposium on Low Power Electronics and Design, 1996, pp. 341–345.

[14] V.G. Moshnyaga, K. Inoue, M. Fukagawa, Reducing energy consumption of video memory by bit-width compression, in: ACM/IEEE International Symposium on Low Power Electronics and Design, 2002, pp. 142–147.

[15] J. Montenaro et al., A 160 MHz 32 b 0.5 W CMOS RISC Microprocessor, in: International Solid-State Circuits Conference, 1996.

[16] D. Patterson, J. Hennessy, Computer Architecture: a Quantitative Approach, Second ed., Morgan Kaufmann Publishers, 1996.

[17] M.D. Powell, A. Agarwal, T.N. Vijaykumar, B. Falsafi, K. Roy, Reducing set-associative cache energy via way-prediction and selective direct-mapping, in: ACM/IEEE 34th International Symposium on Microarchitecture, 2001, pp. 54–65.

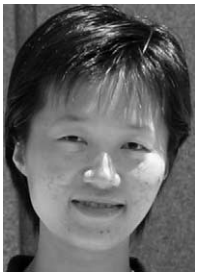
[18] P. Ranganathan, S. Adve, N. Jouppi, Reconfigurable caches and their application to media processing, in: 27th Annual International Symposium on Computer Architecture, 2000, pp. 214–224.

[19] C. Su, A. Despain, Cache design tradeoffs for power and performance optimization: a case study, in: ACM/IEEE International Symposium on Low Power Electronics and Design, 1995, pp. 63–68.

[20] O.S. Unsal, R. Ashok, I. Koren, C.M. Krishna, C.A. Moritz, Cool-cache for hot multimedia, in: ACM/IEEE 34th International Symposium on Microarchitecture, 2001, pp. 274–283.

[21] J. Yang, J. Yu, Y. Zhang, Lightweight set buffer: low power data cache for multimedia applications, in: ACM/IEEE International Symposium on Low Power Electronics and Design, Seoul, Korea, August 2003, pp. 270–273.

[22] K. Usami et al., Automated low-power technique exploiting multiple supply voltages applied to a media processor, IEEE Journal of Solid-State Circuits 33 (3) (1998) 463–471.



Jun Yang received the BS degree in computer science from Nanjing University, China, in 1995, the MA degree in mathematical sciences from Worcester Polytechnic Institute, MA, in 1997, the PhD degree in computer science in the University of Arizona in 2002. She is an assistant professor of the computer science and engineering in the University of California at Riverside. Her research interests are in the areas of secure program execution, temperature-aware microarchitecture designs, and network processor designs. She is a member of ACM and IEEE.

Jia Yu is a PhD candidate in the Department of Computer Science and Engineering at University of California, Riverside. Her research interests include low-power microprocessor architecture, network processor architecture, and design automation. She received her Master's degree from Singapore-MIT Alliance in 2002, and a Bachelor's degree from the Zhejiang University, China in 2000.

Youtao Zhang received the PhD degree in computer science in the University of Arizona in 2002. He is an assistant professor of the computer science at the University of Texas at Dallas. His research interests are in the areas of computer architecture, program profiling, program analysis and optimization, and data compression. He is the recipient of US NSF Career Award in 2005, the distinguished paper award of the IEEE/ACM International Conference on Software Engineering (ICSE) conference in 2003, the most original paper award of the International Conference on Parallel Processing (ICPP) conference in 2003. He is a member of the ACM and the IEEE.