Lightweight Set Buffer: Low Power Data Cache for Multimedia Application

Jun Yang Jia Yu
Computer Science and Engineering Department
University of California, Riverside
Riverside, CA 92521
email: {junyang,jiayu}@cs.ucr.edu

Youtao Zhang
Computer Science Department
The University of Texas at Dallas
Richardson, TX 75083
email: zhangyt@utdallas.edu

ABSTRACT

A new architectural technique to reduce power dissipation in data caches is proposed. In multimedia applications, a major portion of data cache accesses hit in the same cache set continuously before going to a different set. This feature allows us to remove unnecessary driving power in data arrays as long as the same cache set is accessed incessantly. Power saving is achieved through buffering and accessing the cache set instead of the main data array. The proposed technique does not incur performance degradation and accomplishes up to 57% of power reduction for data caches.

Categories and Subject Descriptors

B.3.2 [**Hardware**]: Memory Structure—*Design Styles*

General Terms

Design, Performance, Experiment

Keywords

Low power, cache, multimedia

1. INTRODUCTION

As the speed gap between the memory and CPU continues to increase, modern processors tend to enlarge their on-chip caches to reduce the number of accesses to long latency memories. For this reason, the on-chip caches remain as a major chip power consumer. For example, the Intel Pentium Pro dissipates 33% [3] and the StrongARM 110 dissipates 42% [7] of its total power in caches. Consequently, there have been increasing interests in designing low-power on chip caches.

The multimedia applications have become an important workload for general-purpose processors nowadays. Through study, we found that multimedia applications present high *set-wise access locality*—a set is continuously accessed for a period of time. For direct-mapped caches, it means cache access switch to a different *line* after several continuous accesses. For set-associative caches, there should be a slight increase in set-access locality since a set contains multiple lines. Figure 1 shows the set-wise access locality for a 32KB cache with different set-associativities running MediaBench benchmarks [6]. On average, we observe around 37% of total cache accesses hit in the last accessed set for associativities ranging from 1 to 32.

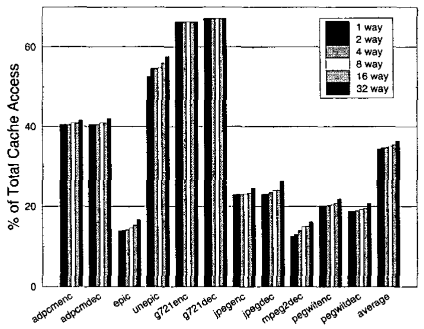

Figure 1: Percentages of accesses that hit in last accessed cache set.

With this feature, we propose to use *lightweight* set buffer that stores the last accessed cache set. The set buffer is lightweight because we made use of existing internal latches inside the cache with only minor modifications. Power is saved when an access is directed to the set buffer instead of the entire data array. The direction is guided by a single bit per set which indicates whether this set is last accessed or not. A true bit value will lead access to the set buffer and block data array driving. This technique does not slow down the cache access, nor does it require excessive amount of extra hardware. Our experiments show an average of 23%/26%/29% of power savings on a 8KB direct-mapped/16KB 2-way/32KB 4-way set associative cache.

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 [8, 2]. 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 [8]. This design is essentially a single-entry filter cache 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 [2] introduced a concurrent version of line buffers in which cache lines in the same set is 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 ongoing access in the main cache assuming that the former is resovled 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 the latest 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

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 cache lines are read out and steered into m line buffers waiting for tag-compare results. If it is a cache hit, the target word will be selected out from one of m 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 line buffers which already contain the needed information. 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 line buffers.

3.1 The Structure

Figure 2 depicts the logic design of the lightweight set buffer, an additional supporting bit array and how they interact with the standard tag and data array. 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.

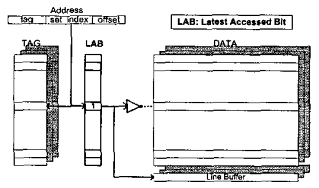

Figure 2: The design of a normal data cache with lightweight set buffer.

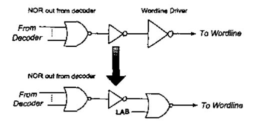

Figure 3: Circuit Changes to Wordline Drivers.

As mentioned before, we add a single bit to each cache set to indicate whether the set is latest accessed or not. These bits form the Latest Accessed Bit (LAB) arrays and are accessed by the same *set_index* address field as being used for both tag and data arrays. If an LAB is set, the corresponding cache set should already have been steered into the set buffer by the last access. Therefore, the word line driving in the data array is gated off by an inverter as shown in the broken lines in the graph. This can guarantee that the data array is not activated. Meanwhile, the set buffer is driven to get ready for data access, as shown in solid lines in the graph. It is now clear to see when the LAB is set, power saving is achieved through not activating the entire data array.

In reality, what is added may not be an inverter. Figure 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 [5]. To incorporate the LAB signal, it is only necessary to replace the wordline driver with an NOR gate. An NOR gate at this place will not conduct if the LAB is high and will translate to an inverter if the LAB is low. As we can see that, the modifications required in the cache control circuits is minimal.

3.2 Maintaining the LABs

In this section, we will first describe how to fill and update the LAB arrays, and then analyze their power overhead. Since the LAB indicates the last accessed set, we can observe that at anytime, there can be one and only one bit that is set in the LAB array.

To ensure the above property, one must clear the bits that were set 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 set, no action needs to be taken since the bit is already set and should stay set for the next time. If it is unset, use the stored *set_index* to clear its corresponding LAB and set the current LAB using the current *set_index*. On a cache miss, the LAB array should be flushed and we do not attempt to reflect the cache line fill in the set bufer.

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 0 is detected since from this point on the cache behaves normally. Thus, updating LAB is carried in parallel with 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 performace 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.

Power Overhead. The LAB array and the necessary additional register for *set_index* bring only marginal extra power. We used the power and timing evaluation tool XCACTI [4] to measure the additional power dissipation due to the extra hardware. The LAB's read and write power in permillage of the original cache are shown in Figure 4. As we can see, reading LAB consumes negligible power while writing is more demanding. This is because updating an LAB requires extra decoding of the stored *set_index*. Most of the numbers are around or well below 1%. And configurations with more bits in *set_index* tend to have higher permillage of LAB power. Also note that although LAB read happens on nearly every cache access, LAB write happens only when the LAB is unset and on cache misses. Therefore, the accumulative power overhead due to LAB-read and LAB-write has less impact on power savings than the data shown here. We will present the overall power savings in section 5.

read,write	8KB	16KB	32KB	64KB
1-way	0.1,19.3	0.1,18.2	0.1,22.5	1.1,14.7
2-way	0.4,17.6	1.0,14.9	1.0,11.2	1.0,27.7
4-way	0.2,10.0	1.6,9.7	1.5,15.2	0.6,11.0
8-way	0.1,3.7	1.4,5.6	0.2,7.2	0.2,5.5
16-way	2.6,3.6	2.5,3.7	0.1,1.6	2.1,4.5
32-way	2.3,2.6	2.7,3.3	2.5,3.1	1.2,1.6

Figure 4: Additional power consumed by LAB arrays and *set_index* buffers (in permillage of original cache).

3.3 Complications on 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. 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. Figure 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

Figure 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 Figure 6.

cache	cache hit not following a miss	read hit	(1) set buffer hit (LAB=1)
			(2) set buffer miss (LAB=0)
		write hit	(3) set buffer hit (LAB=1)
			(4) set buffer miss (LAB=0)
	cache miss not following a miss		(5) LAB=1
			(6) LAB=0
	(7) rest access		

Figure 6: Power Consumption Components

In category 1/3/5, we save the energy of reading the entire data array, which specifically are the energy consumed in cache wordline, bitline and sense amplifier. As we mentioned, we also need to pay the energy overhead for accessing and updating LAB. In category 1-6, we pay the energy overhead for accessing LAB. In category 2/4/7, we pay the energy overhead for LAB update.

5. EXPERIMENT EVALUATION

We implemented our proposed lightweight set buffer technique in an execution driven simulation tool SimpleScalar [1] with cache power model extension XCACTI [4]. We evaluated our design through running a subset of MediaBench benchmark [6] suite (currently there are 11 programs that are running correctly in our system), then we used XCACTI to calculate the energy saving on L1 data 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 compare the energy saving with previous work, Wider Line Buffer approach [2].

In this set of experiments, we used the following cache parameters for L1 data cache: 8KB direct-mapped, 16KB 2-way set-associative, and 32KB 4-way set-associative, all with 32B line size. Our purpose is to compare with the WLB design [2] 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 Figure 7 in three separate graphs for three cache configurations. For all the configurations we tested, our lightweight set buffer outperforms the WLB. On average, we achieve 23%/26%/29% power savings for 8KB/16KB/32KB cache respectively. We observed that with about the same amount of set buffer hit ratio (Figure 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 [4]. Reducing the data array activity directly translate 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 the 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 16KB and 32KB cache respectively, but increases the 8KB 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 disign. We attribute this results to the "simpler is better" rule.

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 Figure 8. 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 8KB, 16KB, and 32KB respectively. The LAB overhead as shown in Figure 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.

6. CONCLUSION

In this paper, we designed a lightweight set buffer in data 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 overspend power even when the set-wise access locality is low. This is a accomplishment that could not be achieved previously.

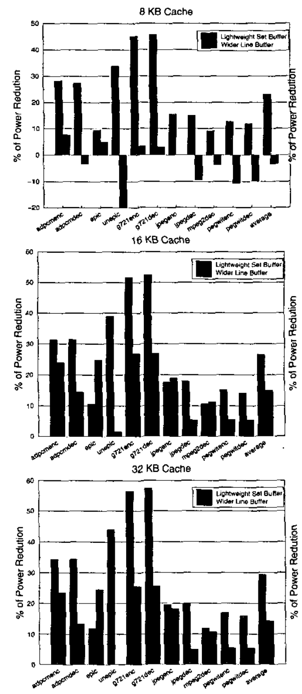

Figure 7: Overall power reduction compared with WLB.

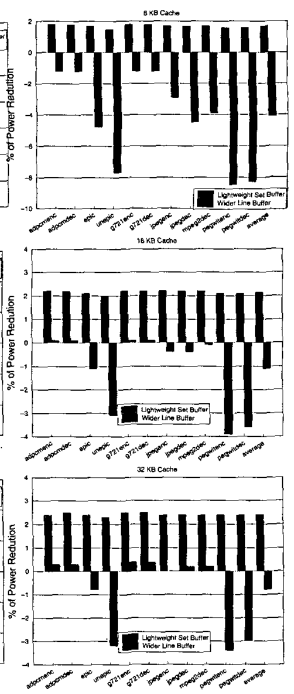

Figure 8: Power variations when set buffer WLB hit ratio is low.

7. REFERENCES

[1] D. Burger and T. Austin, "The SimpleScalar Tool Set, Version 2.0," *Technical Report 1342, Univ. of Wisconsin-Madison, Comp. Sci. Dept.*, 1997.

[2] K. Ghose, M. B. Kamble, "Reducing Power in Superscalar Processor Caches using Subbanking, Multiple Line Buffers and Bit-Line Segmentation," *ISLPED'99*, pp. 70–75, 1999.

[3] S. Gunther and S. Rajgopal, *Personal communication.*

[4] M. Huang, J. Renau, S. M. Yoo, J. Torrellas, "L1 Data Cache Decomposition for Energy Efficiency," *ISLPED'01*, pp. 10–15, 2001.

[5] N. P. Jouppi and 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.

[6] C. Lee, M. Potkonjak, W. H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications," *MICRO-30*, pp. 330–335, 1997.

[7] Montenaro J. Et al., "A 160MHz 32b 0.5W CMOS RISC Microprocessor," *International Solid-State Circuits Conference*, 1996.

[8] C. Su, A. Despain, "Cache Design Tradeoffs for Power and Performance Optimization:A Case Study," *ISLPED'95*, pp. 63–68, 1995.