Reducing Leakage Power in Peripheral Circuits of L2 Caches

Houman Homayoun and Alex Veidenbaum
Dept. of Computer Science, UC Irvine
{hhomayou, alexv}@ics.uci.edu

Abstract

Leakage power has grown significantly and is a major challenge in microprocessor design. Leakage is the dominant power component in second-level (L2) caches. This paper presents two architectural techniques to utilize leakage reduction circuits in L2 caches. They primarily target the leakage in the peripheral circuitry of an L2 cache and as such have to be able to cope with longer delays. One technique exploits the fact that processor activity decreases significantly after an L2 cache miss occurs and saves power during L2 miss service time. Two algorithms, a static one and an adaptive one, are proposed for deciding when to apply this leakage reduction technique. Another technique attempts to keep the peripheral circuits in a lower-power state most of the time. The results for SPEC2K benchmarks show that the first technique can achieve a 18 to 22% reduction in L2 power consumption, on average (and up to 63%), depending on the decision algorithm. The second technique can save 25%, on average (and up to 80%). This comes with a negligible 1 to 2% performance impact, on average, depending on the technique used.

1. Introduction

Power dissipation is a major issue in designing new processors. In particular, CMOS technology scaling has significantly increased the leakage power dissipation so that it accounts for an increasingly large share of processor power dissipation [1,2,3]. A modern L2 cache is very large, 2 to 4MB of data plus tags, and occupies a large fraction of chip area and dissipates a large fraction of the chip leakage power. An L2 cache is typically accessed relatively infrequently and actually dissipates most of the power via leakage. The focus of this paper is, therefore, the reduction of the L2 cache leakage power dissipation.

To overcome this problem, a number of technology, circuit, and architectural approaches have been proposed. Many of such techniques concentrated on reducing the leakage of SRAM memory cells by keeping them in a low-power state which retains data but does not allow access such as body bias control, reduced VDD, high Vth, etc. A number of architectural techniques were proposed to utilize such circuits by targeting SRAM cells, e.g. cache decay [8] and drowsy cache [9]. Recent results have shown that a considerable amount of leakage occurs in the peripheral SRAM circuits, such as decoders, word-line and output drivers, etc [19, 14]. In fact, an SRAM memory cell design can be optimized for low leakage currents without a significant impact on the cell area or performance. Thus approaches that concentrate on cell leakage power alone are insufficient and it is very important to address leakage in peripheral circuits.

Peripheral circuits use very large transistors and thus leakage reduction techniques in these circuits introduce significant additional delays. These delays would significantly increase the L2 cache access time if they are incurred on every access. These leakage reduction techniques are thus not applied in high-performance processors. They are typically used in mobile applications where RAM is put into low-power stand-by mode to reduce leakage current while retaining data. For instance, the Row Decoding scheme [14], reduced *both* the sub-threshold and gate leakage in peripheral circuits while in sleep mode.

How to apply these techniques to L2 cache in high-performance processors is a challenging problem because transitions to and from the low-power mode introduce additional delays. Thus an access issued to an L2, which is in the stand-by mode, will take significantly longer. The main issue is therefore how to achieve power savings without loss of performance.

The approach proposed in this paper uses architectural techniques to drive the application of the above-mentioned circuit techniques to reduce the leakage power in the peripheral circuits of the L2 cache. It is assumed that the SRAM cell design is already optimized for low leakage. Two techniques

This work was supported in part by the National Science Foundation under grants NSF CCF–0311738 and CNS–0220069.

1-4244-1258-7/07/$25.00 ©2007 IEEE

are proposed to decide when to transition from normal to stand-by mode and back. It is shown that this can be achieved without increased hardware complexity or performance loss.

The first technique is based on the observation that a processor may spend a large fraction of an applications' execution time waiting for memory and unable to execute new instructions. This is a direct result of a very long memory access time, which today can reach 300 cycles in a uni-processor. Thus an L2 cache miss leads directly to the processor becoming idle. During such an idle time the L2 cache may be put into the stand-by mode (idle mode or IM) leading to significant reduction in its leakage power. This also allows a very simple and local control for transitioning the L2 to the stand-by mode triggered by an L2 miss. and presents an opportunity to mask the transition delays.

Maximum L2 power savings would be achieved if an L2 cache is put into low-power mode for the entire miss service time. However, as shown later in this paper, this results in a significant performance penalty. This is due to the fact that a processor still has instructions it can execute after an L2 miss, some of which require access to the disabled L2 cache. To mitigate the impact on performance, we propose and compare two novel algorithms for deciding when to put the L2 cache in the low-power mode (stand-by mode).

The first algorithm disables the L2 *N* cycles after a cache miss occurs and enables it again *M* cycles before the miss service completes. Both *N* and *M* are significantly less than the L2 miss service time. We refer to this algorithm as static (SA). The processor can continue to execute instructions during the entire miss service period in this case, with any accesses to the disabled L2 cache buffered. Note that the tag store can be left on to maintain cache coherence, this will not have much of an effect on leakage power savings.

The second algorithm, adaptive (AA), monitors the issue logic and functional units of the processor after an L2 cache miss. An L2 disable signal is asserted if the issue logic has not issued any instructions and functional units have not executed any instructions for K consecutive cycles. This algorithm is more complex than the first one as it requires continuous monitoring of issue logic and functional units. However, it may also allow low-power techniques to be applied to other units of the processor (this is beyond the scope of this paper). Experimental results show that both SA and AA degrade performance at the same level, around 1 percent, on average, although SA requires simpler hardware. On the other hand the leakage power saving with AA is slightly higher, on average, than with SA.

While the idle mode (IM) technique described above works well in many benchmarks, it does not always deliver significant savings. For instance, in benchmarks with very low L2 miss rates. A second technique proposed in this paper uses the low-power stand-by mode as default for the L2 cache and only "wakes" it up on an access (referred to as stand-by mode or SM technique). It then keeps the cache in the normal state for L cycles before returning it to the stand-by mode. Any access to the cache made during these L cycles pays no wakeup penalty. This approach has the potential to keep the L2 cache in stand-by mode for long periods of time, thus saving more power. On the other hand, it pays a wakeup performance penalty on some accesses to the L2.

This paper motivates, describes and evaluates the two techniques described above targeting the leakage power in the peripheral circuitry of the L2 cache SRAM. It is organized as follows. Sec. 2 shows the L2 behavior and power dissipation details. Related work is described in Sec. 3. Sec. 4 presents the motivation for proposed architectural techniques. Sec. 5 describes the two architectural techniques and the circuits used to reduce leakage in SRAM. The methodology and experimental results are presented in Sec. 6.

2. Cache power dissipation

The effectiveness of the techniques proposed in this paper depends on the L2 cache behavior. Table 2 shows miss rates and frequency of loads in SPEC2K benchmarks for a 64–bit, 2GHz processor with a memory latency of 300 cycles (described in Table 1). High load frequencies and L2 miss rates observed motivate the application of stand-by mode during L2 miss service time for power reduction.

The SPEC2K benchmarks were compiled with the -O4 flag using the Compaq compiler targeted for the

Table 1. Processor organization

L1 I-cache	128KB, 64 byte/line, 2 cycles
L1 D-cache	128KB, 64 byte/line, 2 cycles, 2 R/W ports
L2 cache	4MB, 8 way, 64 byte/line, 20 cycles
issue	4 way out of order
Branch predictor	64KB entry g-share,4K-entry BTB
Reorder buffer	96 entry
Instruction queue	64 entry (32 INT and 32 FP)
Register file	128 integer and 128 floating point
Load/store queue	32 entry load and 32 entry store
Arithmetic unit	4 integer, 4 floating point units
Complex unit	2 INT, 2 FP multiply/divide units
Pipeline	15 cycles (some stages are multi-cycles)

Table 2. Miss rates and load frequencies.

	DL1 miss rate	L2miss rate	% loads		DL1 miss rate	L2 miss rate	% loads
ammp	0.046	0.1872	0.22	lucas	0.097	0.6657	0.15
applu	0.056	0.6572	0.26	mcf	0.239	0.4284	0.34
apsi	0.027	0.2778	0.22	mesa	0.003	0.2674	0.26
art	0.414	0.0001	0.17	mgrid	0.036	0.4587	0.30
bzip2	0.017	0.0417	0.24	parser	0.020	0.0688	0.22
crafty	0.002	0.0087	0.28	perlbmk	0.005	0.4576	0.31
eon	0.000	1	0.26	sixtrack	0.012	0.0012	0.22
equake	0.017	0.6727	0.25	swim	0.089	0.6308	0.21
facerec	0.034	0.3121	0.21	twolf	0.054	0.0003	0.23
galgel	0.037	0.0057	0.22	vortex	0.003	0.2314	0.24
gap	0.007	0.5506	0.21	vpr	0.023	0.1476	0.30
gcc	0.046	0.0367	0.21	wupwise	0.012	0.674	0.17
gzip	0.007	0.0468	0.20	Average	0.052	0.313168	0.24

Alpha 21264 processor and executed with reference data sets. The architecture was simulated using an extensively modified version of SimpleScalar 4.0 [5]. The benchmarks were fast–forwarded for 3 billion instructions, then fully simulated for 3 billion instructions.

Figure 1 shows the leakage power breakdown of L2 cache components for a 4MB cache. The leakage and dynamic power consumption of the cache were obtained using Cacti5 [13] for a 65nm technology. It is assumed that the SRAM cell leakage reduction techniques have already been applied. Overall, the peripheral circuits are leaking about 90% of total leakage.

The reason is the use of large and more leaky transistors in peripheral circuits, while high Vth and less leaky transistors are used in memory cells. It is also possible to use other leakage reduction techniques in the SRAM cells, e.g. drowsy cache, which is orthogonal to what is proposed in this paper.

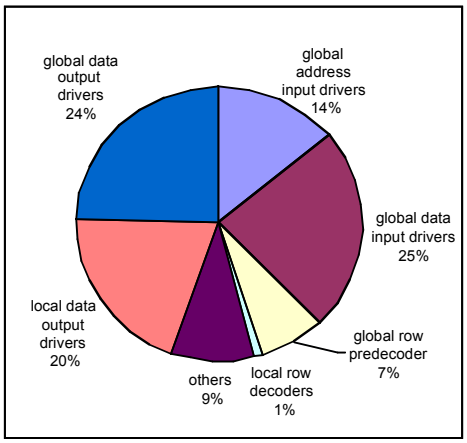

Figure 1. Leakage power of L2 cache components

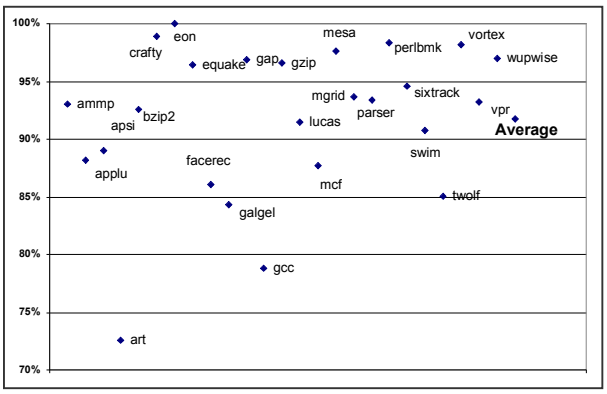

Figure 2. Leakage power as a fraction of total L2 power.

Figure 2 shows the L2 leakage power dissipation as a fraction of the total cache power dissipation (leakage + dynamic) for SPEC2K benchmarks. The L2 cache leakage power dominates dynamic power, with an average above 90% of the total.

The results in Figures 1 and 2 confirm both the importance of targeting the leakage power. as well as the importance of reducing leakage in peripheral circuits.

3. Related work

CMOS scaling leads to significant leakage in the MOSFET transistor [1,2,3]. Many approaches to reducing leakage power have been investigated, at technology, circuit, architecture and compiler/OS levels.

3.1 Circuit-level leakage control

Four main circuit techniques have been proposed to reduce the leakage power. Gated-Vdd turns off the power by using a high threshold transistor. The advantage of this technique is in reducing the leakage power virtually completely. However, it doesn't retain the state of the memory cell. Similarly, Vss can also be gated (gated Vss).

Another technique is voltage scaling which reduces the source voltage. As explained in [18] due to short-channel effects in deep submicron processes, applying voltage scaling reduces the leakage current significantly. This yields a significant reduction in leakage power dissipation, though not as effective as Gated Vdd. This technique retains data. Voltage scaling can be done dynamically and combined with Frequency Scaling (DVFS) [15, 20]. DVFS is widely used by microprocessor companies in existing products, but incurs considerable delay and is not used for fine-grained control.

The third technique, ABB-MTCMOS, increases threshold voltage of a transistor dynamically. The overhead of applying this technique in terms of performance and power makes it inefficient. In another technique it is proposed to reduce cell bias voltage in standby state to reduce cell leakage [16]. This technique was shown to reduce cell leakage more compared to source voltage scaling [14].

Device scaling leads to threshold voltage fluctuation, which makes the cell bias control to reduce the leakage difficult to achieve. In response, [14] proposed a Replica Cell Biasing scheme in which the cell bias is not affected by V_{DD} and Vth of peripheral transistor. [14] also proposed a circuit to reduce leakage in the decoder and word-line driver. It was shown that using RCB and the new row decoding scheme, the gate and sub-threshold leakage was reduced by 88% with only a 10% area overhead.

[12, 17] proposed a forward body biasing scheme (FBB) in which the leakage power is suppressed in the unselected part of cache by utilizing super Vt devices. They also propose techniques to minimize the associated FBB transition latency. They have shown savings of 64% in cache *cell* leakage power.

3.2 Architectural techniques

A number of architecturally driven cache leakage reduction techniques have been proposed. Powell et al proposed applying *gated-Vdd* approach to gate the power supply for cache lines that are not likely to be accessed [6]. A similar idea referred to as *gated Vss* is investigated in [7] in which the ground voltage is being disconnected for such cache lines. Both of these techniques result in the data loss in the gated cache line. This leads to an increase in the cache miss rate and loss of performance.

Kaxiras et al. proposed a cache decay technique which reduces cache leakage by turning off cache lines not likely to be reused [8]. Flautner et al. proposed a drowsy cache which reduces the supply voltage of the L1 cache line instead of gating it off completely [9].The advantage of this technique is that it preserves the cache line information but introduces a delay in accessing drowsy lines. However, the leakage power saving is slightly lower than the gated Vdd and Vss techniques.

Nicolaescu et al [10] proposed a combination of way caching technique and fast speculative address generation to apply the drowsy cache line technique to reduce both the L1 cache dynamic and leakage power.

Bai et al optimized several components of on-chip caches to reduce gate leakage power [11].

The research mentioned above primarily targeted the leakage in the SRAM cells of a cache. Given the results in Figure 1, cache peripheral circuits are even more important to address in the L2 cache.

4. Architectural motivation

A load instruction missing in the L2 cache prevents dependent instructions from being issued. The dependent instructions fill up the reorder buffer (ROB), the instruction queue (IQ), and/or the load and store queues (LQ/SQ) while the miss is being serviced. The load may take 200 to 300 cycles to be serviced and will reach the top of one of the queues during this time. It will cause the queue to fill up with subsequent instructions and then stall the processor. Only after the L2 cache miss is serviced will the stall condition be removed.

Thus, the processor can become completely idle, i.e. it is not issuing, executing, or committing any instructions while waiting for the L2 miss to be serviced. The IPC as measured during the L2 miss service time for SPEC2K programs thus decreases significantly compared to program average, as shown in Figure 3. This idle time can be quite large as will be shown in Section 5.

These results indicate that the L2 cache can be put into a stand-by mode for a significant fraction of execution time. However, doing so right after an L2 miss is not the best approach, as shown below.

Figure 4 shows the fraction of independent instructions issued during an L2 miss service. Independent instructions follow a load miss but do not depend on it or any other miss which may occur during the L2 miss and thus can be executed during miss service. They may also access the L2 and would be delayed if the L2 was in stand-by mode.

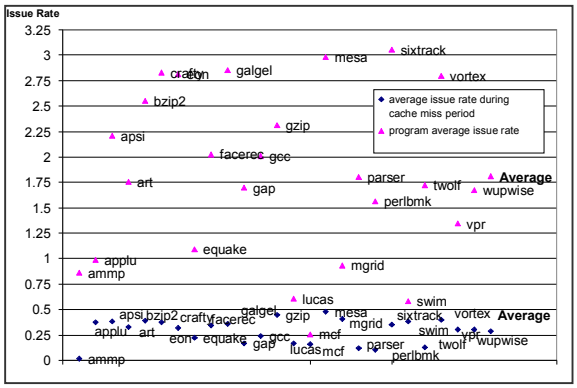

Figure 3. Instruction issue rates

In spite of a significant decrease in issue rate during cache miss service, the percentage of independent instructions is not negligible, particularly in applu,

lucas, mcf, swim and mgrid. Therefore, any technique to reduce power during a cache miss period has to carefully consider execution of dependent instructions to avoid a performance penalty.

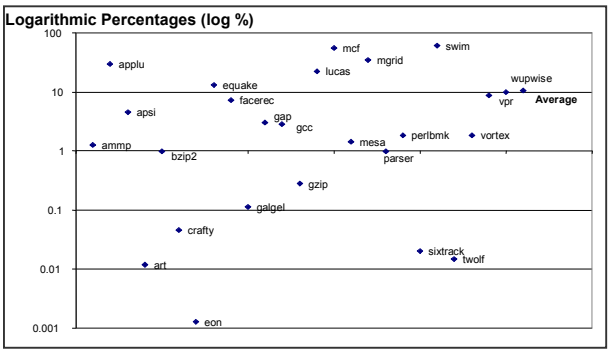

Figure 4. Independent instruction frequency

5. Proposed techniques

The techniques proposed in this paper aim to put the L2 cache into the stand-by, low power mode. Let us start with the circuit techniques used, which insert appropriately sized sleep transistors for both Vdd and Vss. In order to reduce the delay of going into and out of stand-by mode, we divide the peripheral circuits into local and global. The former are primarily local output drivers in each SRAM sub-array, which will be controlled locally with a local SLP signal (lSLP) asserted when the sub-array is not selected and thus incur a reasonably small delay.

Global peripheral circuits include pre-decoder and global word-line drivers, input and output data drivers, and address input driver. These large-transistor circuits will be controlled by the global SLP signal. A Sleep Transition Latency (STL) is incurred by the pre-decoder/ global word-line driver and other global peripheral circuits. Note that global

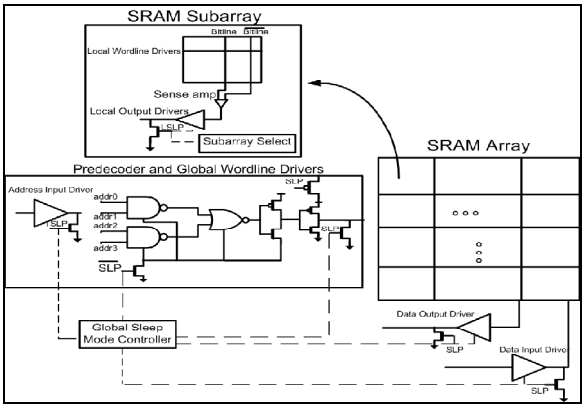

Figure 5. Circuits for leakage control

output driver transition can be allowed to take even longer as it can be overlapped with data read.

The SRAM cell leakage is assumed to be controlled by other techniques in both the baseline and our enhanced architectures (CACTI-5 accounts for this). Note that the local sleep signal can be used to transition a sub-array of SRAM cells and their corresponding sense amplifier to/from low-power mode, but we do not assume its use.

The modified L2 cache architecture is shown in Figure 5. It has a sleep input signal SLP to put its global peripheral circuits into stand-by mode using circuit techniques discussed above.

Our first architectural technique (IM) asserts the L2 SLP signal after an L2 cache miss and de-asserts it again when the cache miss is serviced. The processor is not disabled and can generate L2 accesses during this time. Such load/store instructions are stored in the

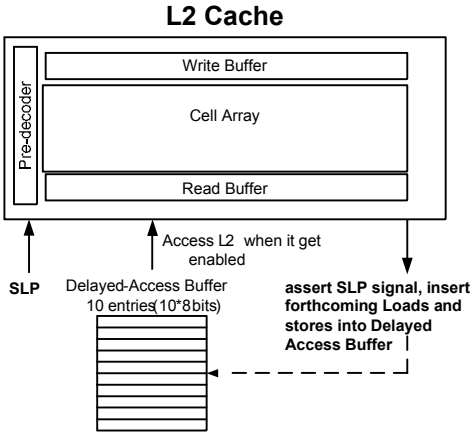

Figure 6. Modified L2 cache architecture.

delayed-access buffer. This buffer allows the processor and the L1 cache to continue executing while L1 misses are stored in this buffer. Our evaluation showed that a 10-entry delayed-access buffer is large enough to keep all loads instructions waiting to access L2 during cache miss period.

The performance when L2 is disabled for its entire miss service time relative to a baseline in which the L2 is always enabled is shown in Figure 7. The IPC degradation is 10%, on average, and between 25% and 50% for some benchmarks: applu, lucas, mcf, swim and mgrid. Disabling the L2 postpones the issue of independent instructions and impacts the performance significantly. Two algorithms are proposed next to avoid this performance degradation.

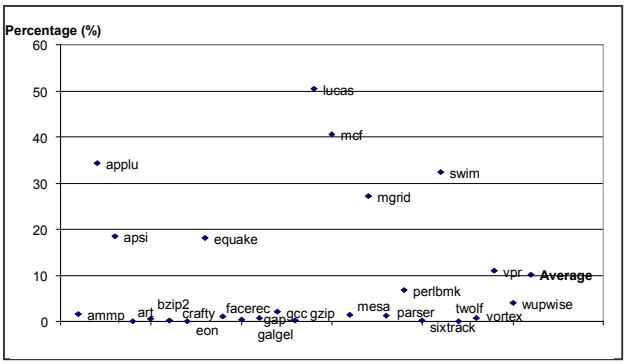

Figure 7. Performance degradation when L2 is disabled for entire L2 miss service time.

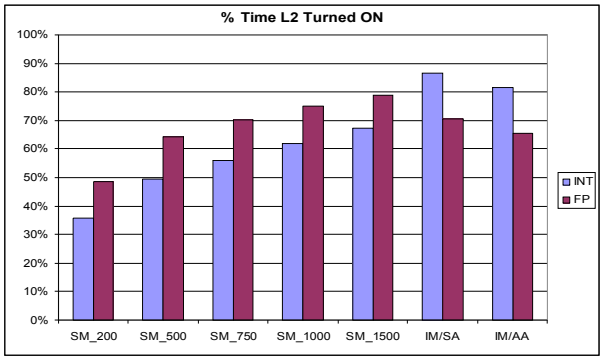

Figure 8. Fraction of execution time L2 is active.

Static algorithm (SA)

This algorithm puts the L2 in stand-by mode N cycles after the cache miss occurs and enables it again M cycles before the miss is expected to compete. We refer to this algorithm as a static algorithm (SA) as it deals with all L2 misses in the same way.

The SA algorithm allows independent instructions to utilize available resources in the ROB, IQ and LQ/SQ and complete execution during the L2 miss service. Our experimental evaluation showed that choosing N=M=50 cycles minimizes the impact on performance.

Adaptive algorithm (AA)

In this algorithm the issue logic and functional units of the processor are monitored after an L2 miss. The L2 is put into stand-by mode if the issue logic has not issued any instructions and functional units have not executed any instructions for K consecutive cycles. The algorithm attempts to predict that there are no more instructions that will access the L2.It was experimentally determined that K=10 cycles is a good power/performance trade-off.

The IM technique puts the L2 cache into the stand-by mode only during part of the L2 miss service time. The SM technique attempts to maximize the time L2 cache spends in the stand-by mode by starting the L2 cache in stand-by mode and "waking it up" on an L1 cache miss. While it is possible to put the L2 back into stand-by as soon as the access is finished, it is likely to be inefficient in terms of performance. The approach we propose keeps the L2 cache on for J cycles after it was turned on and allows other L1 misses to access L2 without a performance penalty. A larger J thus leads to lower performance degradation but also lower energy savings.

Figure 8 shows the fraction of total execution time that L2 cache was kept active by different techniques (SM_200 is for J=200 cycles). Not unexpectedly, the short turn-on period leads to a larger fraction of time the L2 can be in low-power state. But at the same time it may decrease the IPC.

For J=500 or less, the average idle time for L2 in f.p. codes is higher than for the IM technique with either SA or AA algorithms. It is always better in integer codes.

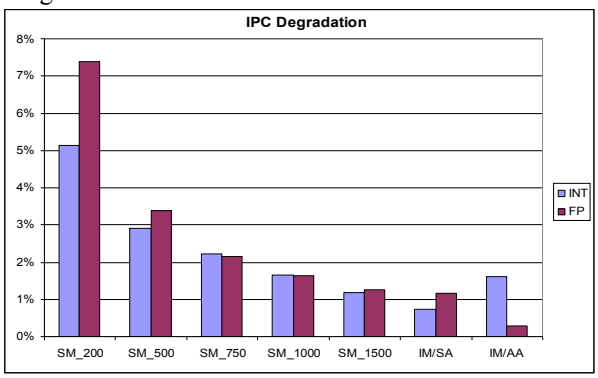

Figure 9. Decrease in IPC.

Figure 9 shows the corresponding IPC decrease. While large for SM with small values of J, it becomes comparable or better at J=1500. For one benchmark (art) the IPC loss is 40% for J=200.

6. Power reduction

This section presents the results for power reduction, energy-delay product, and IPC degradation for individual benchmarks. First, let us describe power and timing assumptions used.

As shown in Section 2, the global peripheral circuits controlled by the selected circuit techniques account for 70% of all the leakage power, according to CACTI-5. The power reduction using some of the same leakage control circuits reported in [16, 14] was 88%. In addition, local peripheral circuits account for another 20%. These are the values used to obtain power savings shown in Figure 10.

Total dynamic power was computed as N*Eaccess/Texec, where N is the total number of

accesses (obtained from simulation), Eaccess is the single access energy from CACTI-5 and Texec is the total execution time. Leakage power computations are similar, but leakage energy is dissipated on every cycle.

The time delay for transition to/from stand-by mode, STL, used in this work is 10 processor cycles (5ns). The simulator accounts for this delay as required by each technique/algorithm.

Figure 10 shows a) the leakage power savings for both techniques, b) the energy-delay product reduction, and c) IPC degradation associated with both techniques. On average, the SM technique reduces leakage power and the energy-delay product slightly more than IM. Performance-wise, the IM technique does better, on average.

The IM results for two algorithms show that the SA and AA degrade performance equally, by about 1%. However, the worst performance loss occurs under AA, in mcf, by 16%.

So far an 88% leakage reduction was assumed when using circuit techniques from [14,16]. Other circuit techniques report different savings, e.g. [12] reports a 64% reduction (in the cell array). We evaluated the savings assuming a 65% leakage reduction in individual circuits. The result was that

total power was reduced somewhat proportionally. That is, 25% lower circuit leakage savings (from 88 to 65%) result in approximately 25% lower total power savings.

7. Conclusions

This paper presented two architectural techniques to reduce leakage in the L2 peripheral circuits. The first (IM) achieves 18 or 22% average leakage power reduction, depending on the detection algorithm used, with a 1% average IPC reduction. The second technique (SM) achieves a 25% average savings with a 2% average IPC reduction.

The L2 hardware complexity using the IM technique is minimal: a 10-entry delayed-access buffer. The AA requires counters to monitor the functional units and issue logic after an L2 miss to detect when units become idle. There is also an area overhead in L2 SRAM due to the leakage reduction circuits used (for instance, [14] reported a 10% area increase).

The two techniques benefit different benchmarks, which indicates a possibility adaptively selecting the best technique. This is subject of our ongoing research.

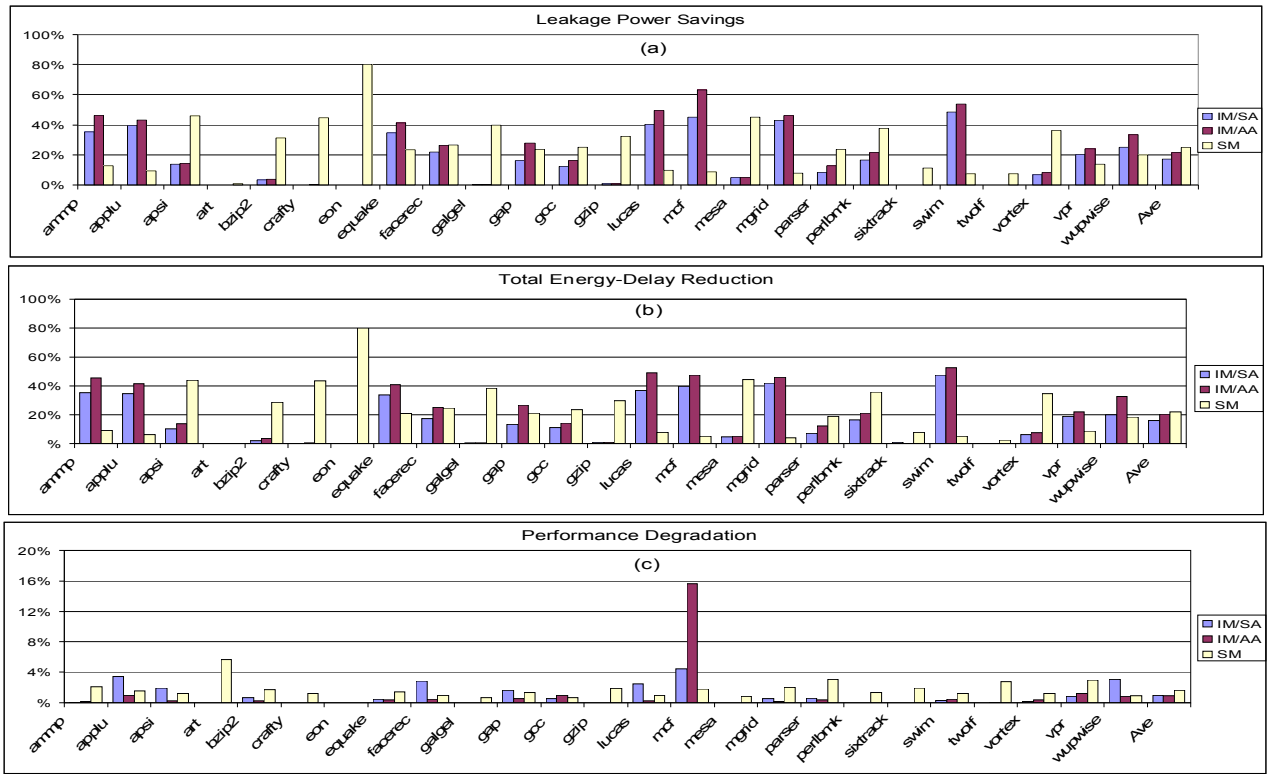

Figure 10. (a) leakage power saving (b) total energy-delay reduction (c) IPC degradation.

8. References

[1] T. Skotnicki et al,.The end of CMOS scaling: toward the introduction of new materials and structural changes to improve MOSFET performance. *IEEE Circuits and Devices Magazine*, Jan.-Feb. 2005, Vol. 21, Issue: 1.

[2] S. Borkar et al,. Platform 2015: Intel® Processor and platform evolution for the next decade. *Intel Technology Magazine*, March 2005.

[3] M. Bohr. Nanotechnology goals and challenges for electronic applications. IEEE Transactions on Nano-technology, Vol 1, No. 1, p.56, March 2002.

[4] F. Hamzaoglu et al,. Analysis of Dual-VT SRAM cells with Full-Swing Single-Ended Bit Line Sensing for On-Chip Cache. IEEE Trans. on VLSI Systems, vol. 10, April 2002, pp. 91-95.

[5] SimpleScalar4 tutorial, Simp[leScalar LLC. http://www.simplescalar.com/tutorial.html.

[6] M.D. Powell et al,. Gated V_{dd}: A circuit technique to reduce leakage in deep-submicron cache memories. in Proc. *IEEE ISLPED*, 2000 .

[7] Y. Li et al,. State-preserving vs. non-state-preserving leakage control in caches. in Proc. *IEEE DATE*, 2004.

[8] S. Kaxiras et al,. Cache decay: exploiting generational behavior to reduce cache leakage power. in *ISCA*, 2001.

[9] K. Flautner et al,. Drowsy caches: simple techniques for reducing leakage power. in Proc. *IEEE ISCA*, 2002.

[10] D. Nicolaescu et al,. Fast Speculative Address Generation and Way Caching for Reducing L1 Data Cache Energy. Proc. *IEEE ICCD*, 2006.

[11] R. Bai et al,. Total leakage optimization strategies for multi-level caches in Proc. *ACM Great Lakes symposium on VLSI*, 2005.

[12] C. H. Kim et al,. A forward body-biased low-leakage SRAM cache: device, circuit and architecture considerations. IEEE Trans. on VLSI Systems, vol. 13, no. 3, Mar. 2005, pp. 349-357.

[13] Cacti5, http://quid.hpl.hp.com:9082/cacti/.

[14] Y. Takeyama et al,. A Low Leakage SRAM Macro with Replica Cell Biasing Scheme. *IEEE Journal Of Solid- State Circuits*, Vol. 41, No. 4, April 2006.

[15] D. Marculescu. On the use of microarchitecture-driven dynamic voltage scaling. In *Workshop on Complexity-Effective Design*, June 2000.

[16] K. Nii et al., A 90-nm low-power 32 KByte embedded SRAM with gate leakage suppression circuit for mobile applications, IEEE J. Solid-State Circuits, vol. 39, no. 4, pp. 684-693, Apr. 2004.

[17] A. Agarawal et al., DRG-Cache: A Data Retention Gated-Ground Cache for Low Power, DAC 2002. pp. 473-478.

[18] K. Flautner et al,. Automatic performance setting for dynamic voltage scaling. in Journal of Wireless Networks, pages 260–271, 2001.

[19] Bharadwaj S. Amrutur et al,. Speed and power scaling of SRAMs, IEEE Journal of Solid State Circuits. Feb 2000, vol. 35, no. 2.

[20] Pentium M processor on 90 nm process with 2-MB L2 cache. Intel Corp. .Jan. 2005. www.intel.com/design/mobile/datashts/302189.htm