On Load Latency in Low-Power Caches

Soontae Kim, N. Vijaykrishnan, M. J. Irwin and L. K. John†

Computer Science and Engineering Dept.
The Pennsylvania State University
{sookim,vijay,mji}@cse.psu.edu

†Electrical and Computer Engineering Dept.
The University of Texas at Austin
ljohn@ece.utexas.edu

ABSTRACT

Many of the recently proposed techniques to reduce power consumption in caches introduce an additional level of non-determinism in cache access latency. Due to this additional latency, instructions speculatively issued and dependent on a non-deterministic load must be re-executed. Our experiments show that there is a large performance degradation and associated energy wastage due to these effects of instruction re-execution. To address this problem, we propose an early cache set resolution scheme. It is based on the observation that the displacement values used for address generation are generally small. Our experimental evaluation shows that this technique is quite effective in mitigating this problem.

Categories and Subject Descriptors: B.3 [Memory Structures]: Performance Analysis and Design Aids

General Terms: Performance, experimentation

Keywords: Load latency, low-power, caches

1. INTRODUCTION

Current superscalar processors, such as Alpha 21264 [7], MIPS R10000 [8], and Pentium 4 [9] speculatively issue instructions dependent on a load as early as possible. The speculative issue of dependent instructions is performed assuming that the load will hit in the data cache so that the data will be available after a fixed number of cycles and that data forwarding logic can be used. However, when the load cannot provide the data within the fixed number of cycles, the speculatively issued consumers of the load and their descendants must be squashed from the pipelines and re-issued when the data does become available.

There are several reasons that load latencies could be non-deterministic. An obvious reason is a cache miss [7]. Another source of non-deterministic load latency occurs due to the bank conflicts in multi-banked cache architectures [12].

A new source for non-determinism associated with load latencies arises from the use of low-power cache architectures. There have been several approaches to reducing the leakage and dynamic energy consumed in caches [1, 2, 3, 4, 5]. Non-determinism can arise as probes are limited to a predicted portion of the cache [1, 2], so additional probes are required when a prediction fails or because accesses to the cache lines in leakage-saving mode require additional cycles to wake them up and service the request [4]. This additional non-determinism due to low-power caches is particularly interesting because it not only degrades performance (and consequently increases energy consumption) but can also waste energy in squashing and re-issuing instructions. Thus, this effect can offset the power benefits of an intended low-power cache if proper attention is not paid to this non-determinism conundrum. Further, it must be observed that the penalty of non-deterministic load latency will become larger in (future) deeper pipelines. The penalty depends on the *load resolution latency*, which is defined as the delay from the issue of a load instruction to when its cache access result arrives at the issue queue.

The contribution of this paper is two-fold. First, we analyze the performance and energy impact of non-deterministic load latencies that can arise from the use of low-power caches. We specifically focus our evaluation on a *drowsy cache* scheme [4], a technique that has been shown to be effective and simple for controlling leakage. The second part of the paper focuses on alleviating this non-deterministic latency penalty. For that purpose, we propose a novel *early cache set resolution* scheme. It is based on the observation that the displacement values used for address generation are generally small. Our results show that the early cache set resolution scheme is quite effective in reducing the performance and energy penalties associated with the drowsy cache.

The rest of this paper is organized as follows. Section 2 explains in detail why non-deterministic load latency occurs in low-power caches and gives the impact of non-deterministic load latency on performance and energy consumption. Our proposed early cache set resolution scheme is presented in section 3. We give experimental results on performance and energy in section 4. Finally, section 5 concludes.

2. NON-DETERMINISTIC LOAD LATENCIES DUE TO LOW-POWER CACHES

Proposed low-power caches can be classified into two categories. One tries to reduce dynamic power and the other leakage power. *Way-prediction* and *selective direct mapping* schemes [1, 2] incur non-deterministic load latency when a cache must be re-probed when the first probe fails. The use of a *cache decay* scheme [3] or the drowsy cache scheme

ISLPED'03, August 25–27, 2003, Seoul, Korea.

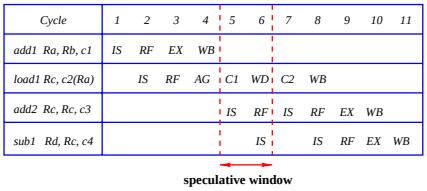

Figure 1: A non-deterministic load in a drowsy cache. IS means instruction issue, RF register file read, EX execution, AG address generation, C1 and C2 cache access, WD waking-up drowsy cache line and sending a signal to the scheduler, and WB write back, respectively. The load resolution latency is five cycles in the figure.

for leakage power reduction also contributes to additional sources of non-deterministic load latency. When a cache line in leakage-saving mode is accessed, the line must be woken up in the drowsy cache scheme or brought in from the next level of memory in the cache decay scheme. The cache decay scheme can increase cache miss rates if it prematurely turns off currently used cache lines, and consequently increase the number of non-deterministic loads. In the drowsy cache scheme, the source of non-determinism is whether a cache line is in a leakage-saving mode or not, since cache lines are periodically placed in a state-preserving leakage-saving mode (*drowsy state* hereafter). In this mode, the cache line cannot be accessed and a single cycle wake-up time is required. The drowsy cache circuit provides a means to automatically transition the cache line to active mode when an access is made. If a careful attention is not given to, these low-power caches might inadvertently increase (rather than decrease) overall energy consumption of a processor when non-deterministic loads occur frequently. In this study, we use the drowsy cache as our evaluation case.

Figure 1 shows when and how non-determinism happens, and squashing and re-issuing the instructions dependent on a load in the drowsy cache. Data cache access takes two cycles in the figure (*C1* and *C2*). For clarity of explanation, the figure is drawn such that other stages take a single cycle. *WD* means waking-up a drowsy cache line and sending a signal to the scheduler to notify it that a non-deterministic load is encountered. The *add2* instruction depends on the prior *load1*, and the *sub1* on the *add2* instruction. The *add2* is issued at cycle 5, assuming that the prior *load1* will hit in the data cache. Also, the *sub1* is issued at cycle 6, assuming that data will be forwarded from the *add2* at cycle 8. At cycle 6, however, the cache line accessed by the *load1* is in the drowsy state. So, the line must be woken up before it can be accessed. Thus, the instructions issued in cycle 5 and cycle 6 must be squashed from the pipelines as they can not have the correct data. The period constituting cycles 5 and 6 is called the *speculative window*. The two dependent instructions on the *load1* are re-issued at cycle 7 and 8, respectively. Although the speculative window is only two cycles in the example, it will be larger in (future) processors that employ deeper pipelines.

2.1 Experimental Framework

We use Wattch [11] tool suite to model our baseline processor in this study. Our CPU models an aggressive out-of-order superscalar processor. It can decode, issue and commit eight instructions per cycle. Table 1 summarizes the simulation parameters of our baseline processor. Wattch is modified to make the load resolution latency seven cycles in the baseline processor.

Table 1: Baseline processor configuration

Parameter	Configuration
Instruction window	128-entry ROB and register file
Fetch queue	32 instructions
Machine width	8-wide decode, issue, commit
Functional units	8 INT add, 2 INT mult/div 4 FP add, 1 FP mult/div
L1 instruction cache	32KB 2-way, 32B line, 1 cycle latency
L1 data cache	32KB 2-way, 32B line, 2 cycle latency, 2 ports
L2 cache	unified 1M 4-way 64B line, 12 cycle latency
Memory	8B-wide, 80 cycle latency
Branch prediction	2K bimodal

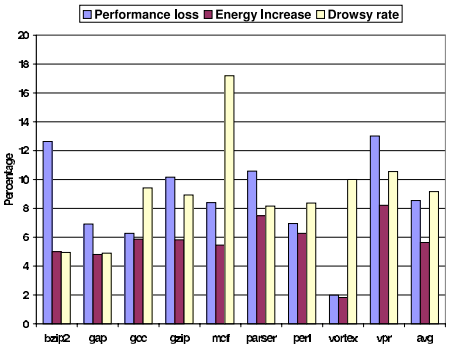

Figure 2: Performance loss, energy increase and drowsy rate for the drowsy cache scheme.

When a non-deterministic load is detected, two recovery mechanisms can be employed. First, all instructions issued in the speculative window can be squashed and re-issued as done in Alpha 21264 processor. The other scheme squashes and re-issues only instructions dependent on a load and their descendants as used in Pentium 4 processor. However, this recovery scheme requires more complex mechanisms to keep track of, and to selectively squash, the load's dependent instructions and their descendants in the pipelines. Hence, we use the first recovery mechanism in our evaluation.

In implementing the drowsy cache scheme, we assume one cycle wake-up time and that all cache lines in the same cache set are woken up to avoid having to wait for tag comparisons to complete when a cache line is in the drowsy state. All cache lines are put into the drowsy state every 2000 processor cycles (*drowsy window size*).

Simulations are done on nine of SPEC2000 integer benchmarks [10] using their PISA binaries. All benchmarks are executed for one billion committed instructions. For all simulations, the train input sets are used. In this paper, we measure only dynamic energy overhead (of both processor core and caches). For energy results, we used energy values produced by Wattch.

2.2 Impact of non-deterministic loads on performance and energy

The frequency of non-deterministic loads depends on how many times cache lines that are accessed by loads are in the drowsy state. We call it *drowsy rate*. The drowsy rate also

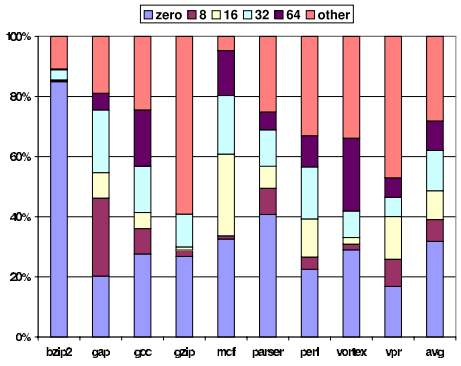

Figure 3: Distribution of displacement values.

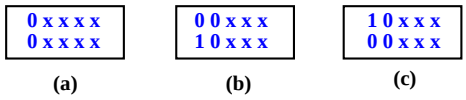

Figure 4: No carry generation to index bits.

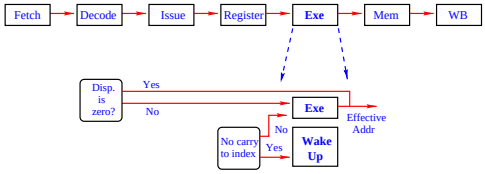

Figure 5: Baseline pipeline (upper) and early cache set resolution logic (lower).

determines the occupancy time of cache ports as they must be occupied another cycle when the cache lines are in the drowsy state.

To see the impact of non-deterministic load latency on performance and energy consumption, we compare the performance and energy consumption of a processor when a drowsy cache is employed to that with a normal cache (*perfect scheduling*). Figure 2 shows normalized performance degradation and energy increase due to the use of the drowsy cache. Overall, the performance degrades by 8.5%, the dynamic energy increases by 5.6% and the drowsy rate is 9.1%. In addition, one can expect that leakage energy also increases due to the performance penalty.

We observe that performance loss and energy increase are generally proportional to the drowsy rate except for the *mcf* and *vortex* benchmarks. Even though the drowsy rate is high, the performance loss and energy increase are low if programs are dictated more by a high cache miss rate. This is observed in the *mcf* benchmark, where the performance loss due to the non-determinism induced by the drowsy cache is small due to a very high data cache miss rate (16.5%). In the *vortex* benchmark, the performance is more dictated by a high instruction cache miss rate (4.2%). In contrast, the *bzip2* benchmark is very sensitive to the drowsy rate as it already has a high IPC.

3. EARLY CACHE LINE WAKE-UP

To wake up drowsy cache lines early, we need a scheme that is both cost-effective and very accurate, as we do not want to increase execution time and power consumption due to the extra hardware required to implement the scheme. We propose a novel *early cache set resolution* scheme. It is based on the observation that most of loads have small displacement values. The scheme needs a few small comparators and has perfect accuracy with reasonable coverage.

We consider only load instructions using a displacement addressing mode, the most commonly used addressing mode in load/store instructions. The load instructions use a base register and a displacement to determine the address from which data is accessed. If the cache index bits of this address can be determined a cycle earlier (i.e., overlapped with the normal register read cycle), we can use this cycle to wake up a cache line in the drowsy state. The key to achieving this is in the displacement values.

Figure 3 shows the distribution of displacement values of load instructions for each benchmark. Each bar has six portions. The *zero* portion means the percentage of loads with zero displacement value. The *"8"* portion indicates the percentage of loads that have displacement values between -8 and 7, inclusive. The *"16"*, *"32"* and *"64"* portions can be similarly interpreted. The *other* portion is the percentage of loads that have displacement values not between -64 and 63. The displacement field has 16 bits in the baseline architecture. As can be observed, a large percentage of loads have small displacement values. The average percentage values are 33.4%, 4.7%, 9.7%, 12.4%, 10.4%, and 29.4% from the *zero* portion to the *other* portion, respectively. Around 50% of loads have displacement values between -16 and 15. This range is the same as that covered by one 32-byte cache line (as used in our baseline configuration). Using this information, when it is certain that the calculated address will have the same index bits as those of the base address (*no carry to index* condition), we do not need to wait until the address is generated to wake up a cache line. This "free" cycle can be utilized to wake up a cache line early. Next, we focus on hardware needed to detect the *no carry to index* conditions.

Figure 4 illustrates three conditions for *no carry to index* we use. As further refinement shows only incrementally better results and requires additional hardware, we use only these three conditions. The three cases show two 5-bits offsets of base address (upper) and displacement (lower) when a cache line is 32 bytes. In the figure, "x" means *don't care* bits. When the MSB (Most Significant Bit) 11-bits of the displacement are zero and a pair of the LSB 5-bits of the base address and the displacement belongs to one of three cases shown, this is a *no carry to index* condition. When the *no carry to index* condition is detected, we can use only the base address to index the cache set to wake it up a cycle early. In this case, address generation and cache line wake-up are performed in parallel, so a performance loss is avoided.

Next, we illustrate the operation of our early cache set resolution scheme in the pipeline. Figure 5 shows our baseline processor pipeline and an early cache set resolution scheme. When a load instruction enters the decode stage, its displacement value is checked. If the value is zero, address generation is eliminated and the base address of the load is used to wake up the cache line in the address generation stage. This eliminates unnecessary power consumption due to address generation and incurs no extra cycle to wake up the drowsy cache line. Otherwise, the address is generated

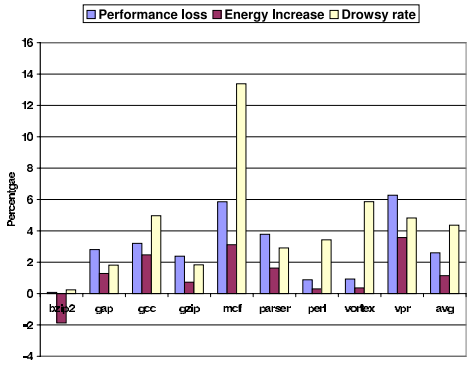

Figure 6: Results of the early set resolution scheme.

as in a normal pipeline. The *no carry to index* condition is checked right after the base address is read from the register file. If the condition is true, the cache line wake-up is performed in parallel with address generation. Displacement value checking is performed with hardwired 11-bits and 5-bits comparators in the decode stage, so it does not require an extra cycle and is done deterministically. The *no carry to index* condition checking is implemented with hardwired 2-bits and 4-bits comparators, so it is performed fast with little power consumption. We evaluated the power consumption of those comparators with CACTI [6], and it is included in the overall processor energy consumption in our experiments. To support the wake-up logic, we use an extra decoder logic for the data cache. While this incurs an additional area penalty, there is no energy overhead as decoding is just done a cycle earlier.

4. RESULTS

In this section, we evaluate proposed early set resolution scheme over the perfect scheduling scheme. Performance and energy values are normalized over those of the perfect scheduling scheme. With the drowsy cache scheme, leakage energy in the data cache is reduced to around one-fourth of the unoptimized value [11], which depends on applications and system parameters.

Figure 6 shows the results when the early cache set resolution scheme is employed. Overall, the performance degrades by 2.6%, the energy increases by 1.1%, and the drowsy rate is 4.4% (as compared to 8.5%, 5.6%, and 9.1%, respectively, in the original drowsy scheme). Thus, a large percentage of performance loss and energy wastage are eliminated. This is mainly due to the decreased drowsy rate by using the early set resolution scheme.

Now, let us look in detail at the behavior of individual benchmarks. The *bzip2* benchmark shows good results. From figure 3, its percentage of loads with zero displacement is 85%. This eliminates most of non-deterministic loads and consequently performance loss. In addition, address generation is avoided for those loads, eliminating the energy consumption for unnecessary address generations. This resulted in overall decrease of energy consumption. By contrast, the *mcf* and *vpr* benchmarks still maintain relatively large performance loss and energy increase. This can be explained by the fact that even if the early set resolution scheme eliminates a large percentage of non-deterministic loads, the drowsy rates for these benchmarks are still high compared to other benchmarks. The average coverage of the early set resolution scheme is 45.6%. This means that the scheme effectively covers most of loads that have displacement values between the zero portion and the 16 portion, inclusive, of figure 3.

5. CONCLUSIONS

This paper revealed that introducing additional non-deter minism in load latencies by employing low-power caches can result in a large performance and energy wastage. To alleviate the energy and performance impact of non-deterministic load latency due to the use of the drowsy cache, we proposed a novel early cache set resolution scheme. When cache sets can be determined early, the sets are woken up before accessing the data cache. Our results demonstrate that the early set resolution scheme is quite effective in reducing performance and energy penalties incurred by the drowsy cache. Specifically, the energy increase when the proposed scheme is employed is around or below 1% of the overall processor energy that does not account for penalties associated with non-deterministic load latency.

6. ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grants 0073419 and 0082064, career award 0093085, and GSRC.

7. REFERENCES

[1] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *Proc. International Symposium on Low Power Electronics and Design*, 1999.

[2] M. D. Powel, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proc. International Symposium on Microarchitecture*, December 2001.

[3] S. Kaxiras, Z. Hu and M. Martonosi. Cache decay: Exploiting generational behavior to reduce leakage power. In *Proc. International Symposium on Computer Architecture*, July 2001.

[4] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proc. International Symposium on Computer Architecture*, July 2002.

[5] S. Yang, M. D. Powel, B. Falsafi, K. Roy and T. N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance i-caches. In *Proc. International Symposium on High-Performance Computer Architecture*, 2001.

[6] G. Reinman and N. Jouppi. An integrated cache timing and power model. Technical Report, Compaq Western Research Lab, 1999.

[7] R. E. Kessler. The Alpha 21264 microprocessor. *IEEE MICRO*, 19(2):24-36, April 1996.

[8] K. C. Yeager. The Mips R10000 microprocessor. *IEEE MICRO*, April 1996.

[9] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker and P. Roussel. The microarchitecture of the pentium 4 processor. In *Intel Technology Journal*, Q1 2001.

[10] SPEC2000 benchmarks. http://www.specbench.org/osg/cpu2000/

[11] D. Brooks, V. Tiwari and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proc. International Symposium on High-Performance Computer Architecture*, 2000.

[12] H. Neefs, H. Vandierendonck, and K. De Bosschere. A technique for high-bandwidth and deterministic low latency load/store accesses to multiple cache banks. In *Proc. International Symposium on High-Performance Computer Architecture*, 2000.