Fast Speculative Address Generation and Way Caching for Reducing L1 Data Cache Energy

Dan Nicolaescu Babak Salamat Alex Veidenbaum
School of Information and Computer Science
University of California, Irvine
Email: {dann,alexv,bsalamat}@ics.uci.edu

Mateo Valero
Computer Architecture Department
Polytechnical University of Catalunia, Spain
Email: mateo@ac.upc.es

***Abstract*— L1 data caches in high-performance processors continue to grow in set associativity. Higher associativity can significantly increase the cache energy consumption. Cache access latency can be affected as well, leading to an increase in overall energy consumption due to increased execution time. At the same time, the static energy consumption of the cache increases significantly with each new process generation. This paper proposes a new approach to reduce the overall L1 cache energy consumption using a combination of *way caching* and fast, speculative address generation. A 16-entry way cache storing a 3–bit way number for recently accessed L1 data cache lines is shown sufficient to significantly reduce both static and dynamic energy consumption of the L1 cache. Fast speculative address generation helps to hide the way cache access latency and is highly accurate. The L1 cache energy-delay product is reduced by 10% compared to using the way cache alone and by 37% compared to the use of Multiple MRU technique.**

I. Introduction

The memory hierarchy organization is critical to achieving high performance and controlling both static and dynamic energy consumption in out-of-order processors. Cache miss rates improve with larger size and higher cache associativity. Recent Pentium processors have used either a 4-way L1 data cache [1] or an 8-way L1 data cache [2], and IBM Power6 will have a 64KB 8-way set associative L1 data cache [3]. Unfortunately higher associativity and size may increase the access latency and significantly increase the dynamic energy consumption of the cache system.

High-performance set associative caches access data and tags in parallel. All tag entries in an indexed set are read and compared with the issued address in parallel. All data lines in a set are read at the same time, even though the addressed data can only reside in one of them. This parallel access to all tags and data entries in a set is inherently energy inefficient, but it is essential for fast cache access. This is also the reason why the dynamic energy consumption of cache access grows with associativity.

The static or leakage energy consumption grows as the process technology shrinks transistor size and threshold voltage. The leakage energy is projected to reach levels comparable to dynamic energy consumption [4]. Large caches are responsible for a high fraction of total chip static power consumption.

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

Given the trend for increasing L1 data cache size and associativity, new solutions need to be found to improve all critical cache parameters: access latency, dynamic energy consumption and static energy consumption.

A number of techniques have been proposed for decreasing the energy consumption of set-associative access (see below). Dynamic energy is reduced by only accessing *one* way of the cache. This requires additional hardware that needs to be looked up before starting the cache operation. The latency of the lookup process increases the cache access time. Static energy is primarily reduced by circuit techniques that lower the voltage of each RAM cell. These techniques require additional time to access the data in the powered down cells.

New techniques are still required in both cases to reduce the time penalty and to further decrease the cache energy consumption. The techniques proposed in this paper for L1 data cache are one possible approach to achieve this.

The new approach proposed in this paper is based on using a Way Cache Unit (*WCU*), a very small fully associative cache that tracks way numbers of recently accessed cache line addresses. The *WCU* is searched prior to L1 cache access. If the *WCU* access was a hit then the way number is read out and only the desired L1 cache way is accessed. This significantly reduces the dynamic energy consumption. The *WCU* is also used to reduce the static energy consumption of the L1 cache in combination with the drowsy cache line technique [5].

The *WCU* was first used to reduce the dynamic energy of highly associative L1 data caches in embedded processors [6] and later also applied to L2 caches in [7]. It works better than way prediction because it is not speculative. However, it does introduce an additional delay in accessing the L1 data cache, degrading the overall performance by approximately 5%.

The new architecture proposed in this paper introduces a fast, 16b address generation unit (AGU) that is used speculatively to compute a part of the effective address, while the rest of the address remains unchanged. This allows a 48b address computation to be completed significantly faster. It also allows pipelined access to the *WCU* CAM to start at the same time as the address calculation. This approach is based on a predictor which is shown to be highly accurate. Finally, the *WCU* reduces the static energy consumption by keeping all L1 lines not in the *WCU* in the drowsy mode.

This paper makes the following contributions to cache

energy management.

- A predictor to determine if an address computation can be performed by the fast AGU is proposed
- A fast address generator enabled by the predictor is proposed allowing access to the Way Cache in the Execute stage
- A very small Way Cache is shown capable of reducing the static energy consumption by putting most L1 cache lines into drowsy state. The Way Cache access is pipelined to further reduce its access time.
- Compares the static and dynamic energy savings and energy-delay product achieved by the Way Cache with other approaches.

The rest of the paper is organized as follows. Related work is briefly described Sec. I-A, address prediction and fast address generation are described in Sec. II, followed by a description of the experimental setup: processor configuration, energy models and a description of the proposed architectures in Sec. III-A, III-B and III-C. The experimental data is analyzed and discussed in Sec. III-D.

A. Related Work

A number of techniques has been proposed to address both dynamic and static energy consumption in caches. Due to space limitations only the most relevant techniques are briefly described below.

The dynamic energy reduction techniques target the high energy consumption of set-associative cache access due to parallel tag and data store look-up. A phased cache [8] first accesses the tag store and compares tags in all N ways of the tag store. Only the requested data store way is accessed in the next 'phase' after the tag comparison completes. This technique increases the cache access latency and is not suitable for high-performance L1 caches. It has been successfully applied in L2 caches [9] without significant performance impact.

Way–prediction, a speculative technique that predicts the correct cache way to access, was first implemented in the R10000 processor [10] (also described in [8]). It uses a predictor with an entry for each set in the cache. Each predictor entry stores the most recently used way for the cache set. The tag and data store is accessed only for the way returned by the predictor. In case of an incorrect prediction the access is replayed, accessing all cache ways in parallel. The required predictor, especially for large modern L1 caches, is likely to increase the cache latency even for correct predictions.

Way prediction in L1 I-cache has been implemented by adding a pointer to the next line/way to each I-cache line [11] or in the branch target buffer [12], but these techniques do not work well in D-caches. Finally, a multiple-MRU predictor (MMRU) has been proposed in [13], which allowed a higher prediction accuracy at the expense of maintaining multiple predictions for each set.

The proposed leakage reduction techniques can be divided into state preserving and state destroying techniques.

The gated-V_{dd} [14] technique identifies the cache lines that are not likely to be accessed and gates the power supply for those lines. As a consequence all the information stored in the powered off lines is lost. The gated-V_{ss} [15] is a similar technique that disconnects the ground connection to shutdown the lines. The later technique is able to reduce bitline leakage more effectively. Both voltage gating techniques increase the miss rate of the cache, since they practically shrink the effective cache size. In [16] a similar technique is used to reduce cache leakage by invalidating and turning off cache lines when they hold data not likely to be reused.

Drowsy cache [5] is a state–preserving technique which reduces supply voltage instead of completely gating it off. The advantage of this technique is that it can achieve the same hit rate as conventional caches, since the drowsy lines preserve their information. However, the leakage current dissipation is slightly higher than that of gated-V_{dd} and gated-V_{ss}. It should be noted that the supply voltage of a drowsy line must be restored before accessing it. This wake–up process takes time and thus imposes additional delay in accessing the cache. In order to minimize the number of lines in active mode, all active cache lines are put into drowsy mode periodically, for example every 2000 cycles.

Cache line reuse information is used in [17] to implement cache line allocation policies, similar information can be used to minimize the number of cache lines that in the active state at a given time.

In [18] the Multiple MRU policy was proposed to be used to reduce the wake–up delay in drowsy caches by keeping the likely to be accessed cache lines awake. This technique keeps one or two most recently used lines awake in each set. It also requires a periodic reset to keep the number of active lines small.

A mixed hardware–software approach was presented in [19]. Tag checks are avoided by having a compiler output special load/store instructions that use tags from a previous load. This approach requires changes to the compiler and the ISA and adds hardware complexity.

II. Proposed Architecture

The goal of the architecture proposed in this paper is to use the Way Cache to reduce both the static and dynamic energy consumed by the L1 data cache. The main advantage of the proposed design is that it can accomplish the energy consumed reduction without increasing the L1 cache latency. This is possible because the additional delay of accessing the Way Cache is hidden by the new fast address generation approach.

The effective address of the memory location accessed is computed in the Execute stage of the pipeline and it consists of the addition of a register and an offset. The technique proposed in this paper relies on the fact that for majority of address computations the offset is small and usually does not have a carry beyond bit 19 [20]. As such the address can be computed in a specialized, narrower, and consequently faster, Address Generation Unit (*AGU*). This faster computation can finish before the end of the cycle and it allows for the Way Cache

access to be also performed within the same cycle and to be pipelined.

A. Speeding up address generation

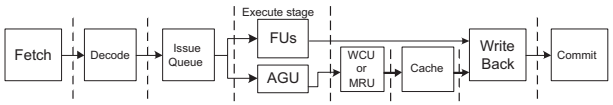

Fig. 1. Original Pipeline

The effective address of memory instructions in most RISC ISAs is computed by addition of an offset to a base. On Alpha 21264 the offset is a 16-bit number and the base is 48 bits. Therefore, a 48-bit adder is used to generate the effective address. Since the offset is 16 bits, a 48-bit wide adder is only used to propagate possible carries from the addition of the least significant 16 bits. The carry propagation terminates upon reaching the first zero in the base and the rest of the bits remain unchanged. Our simulation results show that a 19-bit adder is enough for generating the address of more than 97% of the memory access instructions in SPEC CPU2000 benchmarks. Another observation is that the 3 low–order bits of the offset or the base are often zero and, therefore, no carry is generated from the addition of the least significant bits.

The above properties can be exploited to design a faster address generator (*AGU*). A 3-bit adder adds the low 3 bits of the offset to the base and in parallel a 16-bit adder adds the remaining bits of the offset to the corresponding bits of the base. Then the results of the 3-bit adder and the 16-bit adder and the remaining bits of the base (bit number 19 to 47) are concatenated to create the effective address. Thus the delay of this address generator is equal to that of a 16-bit AGU and can be performed in approximately half the time it takes to perform a 48b add. A schematic of the fast AGU and its operation is shown in Figure 2.

Since the carry bit is not propagated from either the 3–bit adder or the 16–bit adder, the computed address would be incorrect if either adder generates a carry signal. An *OR* of the two carry signals is performed to generate a Carry Out signal. In case of an incorrect address generation (Carry Out=1) the results are discarded and the effective address is computed by the conventional 48b address generator that is started in parallel with the fast AGU (see Figure 3(d)). Obviously, when the generated address is incorrect the WCU access is postponed to the following cycle.

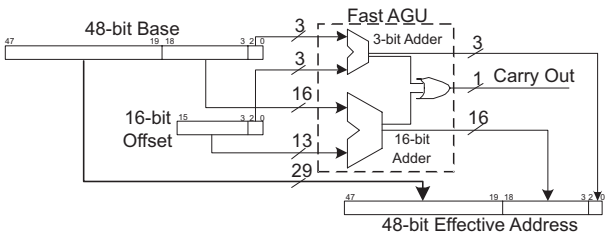

Fig. 2. Fast AGU block diagram

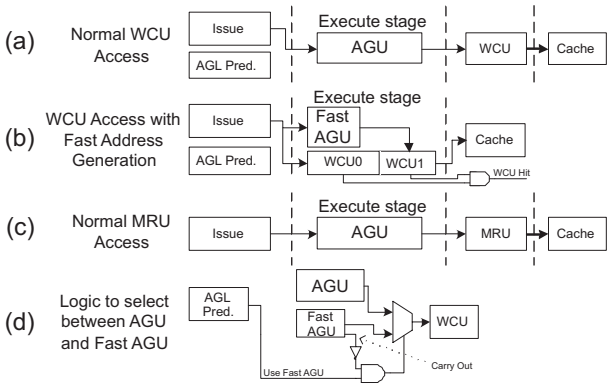

Fig. 3. Modified pipelines

B. Address generation latency predictor

Using the conventional AGU imposes more delay and changes the scheduling of dependent instructions. To handle this issue, an Address Generation Latency (*AGL*) Predictor predicts which of the two AGUs will be used to generate the address and scheduling of dependent instructions is done based on the delay of the predicted AGU.

The AGL predictor is a PC-indexed table. Each table entry is only one bit. The predictor is looked up only for memory instructions during the issue queue stage. The AGL predictor is updated in the write-back stage based on whether the fast AGU generated the correct address. Thus, the fast AGU always computes the address to provide information necessary for updating the predictor.

If the predictor predicted fast address computation for a load and it turned out to be wrong, any dependent instructions that were issued based on this prediction should be replayed. No replay is required in case of mispredictions for stores.

The AGL misprediction recovery is much simpler than branch misprediction recovery. There is no need to flush the pipeline here. It is enough only to issue again the incorrectly scheduled instructions. This technique has been used in Alpha 21264 to reissue the dependents of a load that has a cache miss [11]. The penalty of such a replay is only 1 cycle in our case. Given that a 128 entry AGL predictor is 99% accurate and less than 25% of total instructions are loads, the impact on performance is negligible.

The pipeline organization using the proposed techniques is presented in Figure 3(b). In the *Issue* stage of the pipeline a prediction is made for memory access instructions whether their address computation can be done in the fast AGU. Such instructions access the fast AGU and WCU in the execute stage in two steps: the effective address computation which is done on the fast AGU and the Way Cache Unit look up. In order to make WCU access faster, the WCU CAM tag array is partitioned into two slices. One slice keeps the most significant bits (bits 19 to 47) of the address (WCU0 in the figure) and the second one (WCU1) stores the rest of the tag bits. The width of WCU1 CAM entries for a 64KB cache with 64-byte lines is only 13 bits.

While a 19b address is generated in the fast AGU, the high-order bits of the address remain unchanged from the base register. Thus they are available at the beginning of the Execute stage and can be looked up in WCU0 in parallel with the fast AGU. After fast AGU finishes the computation, WCU1 is accessed using the generated address. A WCU hit occurs when both WCU0 and WCU1 indicate a hit. Using this approach the WCU lookup is effectively shortened. Since the fast AGU takes approximately half a cycle and the 13b WCU1 look up is also significantly faster than the original 42bit lookup, both operations complete before the end of the execute cycle. The WCU data array is accessed next without address decoding by using the AND of WCU0 and WCU1 match lines.

III. Experimental Results

A. Benchmarks and processor configuration

L1 I–cache	64KB, 64 byte/line, 2 cycle
L1 D–cache	64KB, 64 byte/line, 2 cycle, 2 R/W ports
L2 cache	2MB, 8 way, 64 byte/line, 20 cycle
Issue	4 way out-of-order
Branch predictor	64K entry g–share, 4K-entry BTB
Reorder buffer	256 entry
Load/Store Queue	64 entry
Arithmetic Units	4 integer, 4 floating point units
Complex Units	2 INT, 2 FP multiply/divide units
Pipeline	15 cycles (some stages are multi–cycle)

TABLE I

Processor configuration

The baseline architecture modeled here was an aggressive 64–bit high–performance processor. The details of the processor are given in Table I. The memory hierarchy latencies are 3/20/200 cycles for the L1/L2/memory accesses, respectively. The latency of the L1 data cache consists of one cycle to compute the effective address and two cycles for the effective cache access. The L1 cache is virtually indexed and physically tagged.

The SPEC CPU2000 benchmark suite was used with the reference data sets. The benchmarks were compiled with the -O4 flag using the Compaq compiler targeted for the Alpha 21264 processor. The architecture was simulated using a modified version of SimpleScalar [21]. The benchmarks were fast–forwarded for 500 million instructions, then fully simulated for 5 billion instructions.

B. Energy Model

The dynamic and static energy consumption of the data cache was modeled using Cacti4 [22]. The MRU predictor was modeled using a SPICE model for a custom SRAM array. SPICE models were also used to model the CAM part and the data array of the Way Cache. The static energy consumption for cache lines in drowsy state was 10% of the energy consumption in normal state. The process technology used was 0.07 microns.

C. Evaluated Architectures

In order to evaluate the proposed technique, several processor architectures were considered. The processor configurations differed only in the way the L1 data cache access was performed. All configurations use a *drowsy* cache, and try to keep as many cache lines as possible in the drowsy state. If an L1 data cache line being accessed is in the *drowsy* state, an extra cycle is spent to "wake up" the cache line into the normal state before starting the (two cycle) cache access.

The results of all the other architectures are presented relative to the baseline *WCU* architecture in Fig. 1. The baseline configuration uses the Way Cache with a latency of one cycle that is accessed after the address computation cycle (see Figure 3(a)).

On a Way Cache miss a standard parallel associative lookup of all ways in the set is performed. Since all lines in the L1 cache that are not pointed to by a Way Cache entry are kept in a drowsy state, an extra cycle is required to wake up all the lines in the set before the standard lookup can commence. After finding the desired address, the Way Cache is updated with the address–way pair.

Another architecture evaluated uses an MRU predictor instead of a Way Cache (see Figure 3(c)). It keeps one line in each set as pointed by the predictor in the normal state and all the other lines in the set in the drowsy state. For example for a 64KB 4–way set associative cache with 64 byte lines, the number of active lines can be as high as 256 lines. The actual number of active lines is actually smaller as shown in a previous study [23]. This configuration is called *MRU*.

The number of active lines when using the MRU predictor can be further reduced by putting all active lines in a drowsy state after N cycles, as proposed in [5]. This configuration is called *MRUR* and uses N=2000.

The MRU prediction accuracy can be increased by using the MMRU predictor which tracks the last 2 lines accessed in a set. The reset technique was also applied to this architecture. The two MRU lines corresponding to each predictor entry are only put in the normal state one by one as they are accessed in order to minimize static energy consumption. This configuration is called *MMRUR*.

The techniques proposed in this paper are combined in an architecture that is called "WCU with Prediction" (configuration *PWCU*), which only uses the WCU result if the predictor was correct in specifying the fast AGU. If the prediction was incorrect then the standard (parallel) L1 data lookup is performed ignoring the WCU result. A standard lookup is also performed when the 48b AGU is predicted, without attempting to access the WCU.

Another version of the new architecture is called "Predicted Always access Way Cache Unit" (configuration *PAWCU*). This architecture always accesses the WCU, but in different cycles depending on the AGU prediction. The WCU is accessed in the Execute stage if the fast AGU was predicted, otherwise it is accessed in the next cycle. On fast AGU misprediction detection, the WCU is accessed again using the 48b AGU result in the cycle after Execute stage. Obviously, the PAWCU configuration will have in some cases a longer access latency, but it will also benefit from extra hits in the way cache.

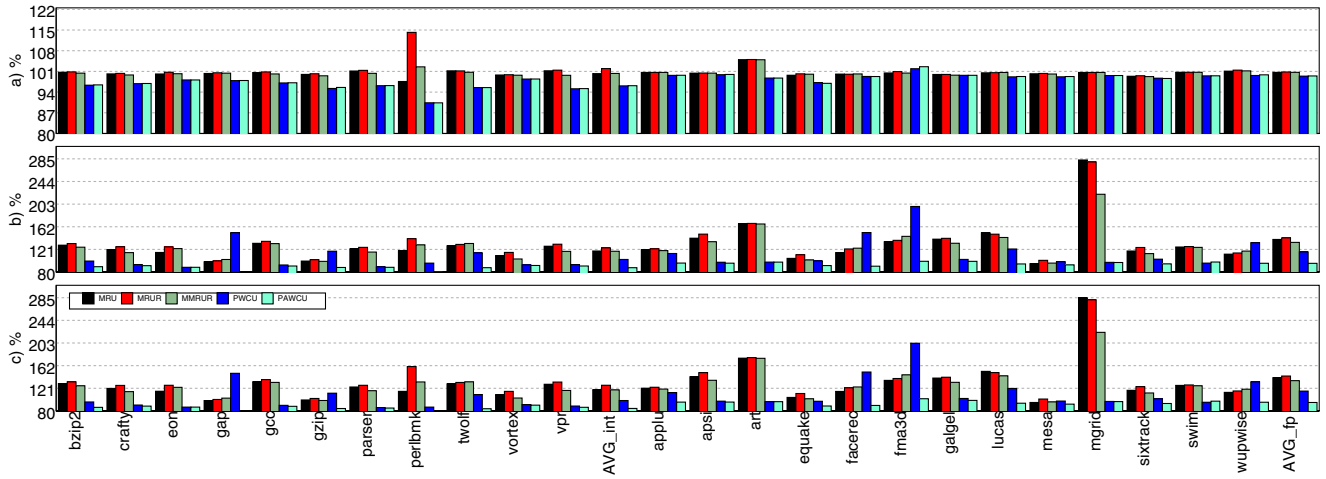

Fig. 4. a) Execution time b) EDP c) ED^2P relative to baseline (16way L1 using a 16entry WCU)

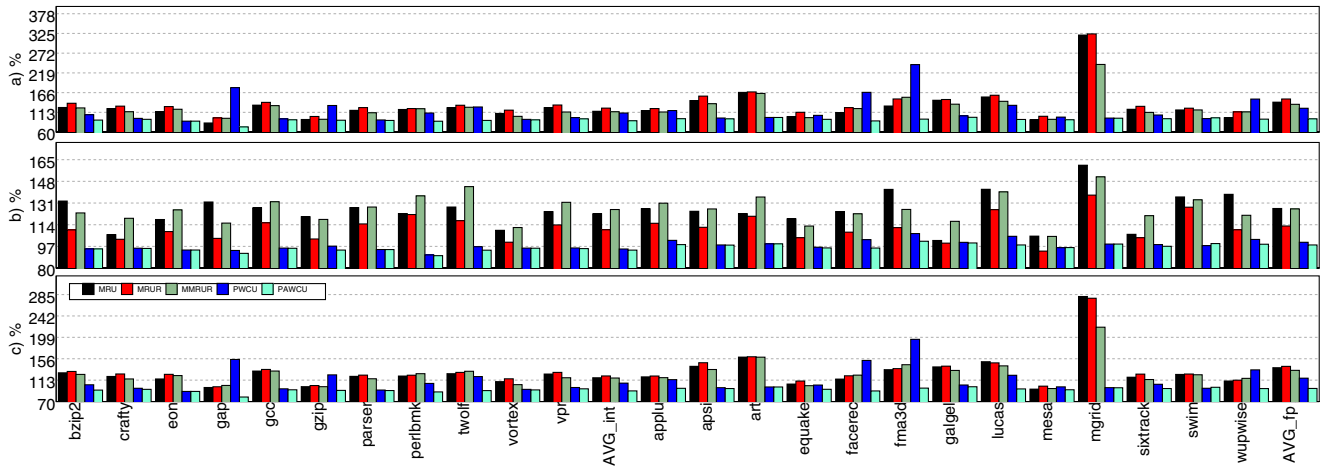

Fig. 5. a) dynamic, b) static, c) total energy relative to baseline (16way L1 using a 16entry WCU)

D. Analysis

The main advantage of the design proposed in this paper is that it allows for a reduction in the L1 data cache dynamic and static energy consumption, but it does so without suffering from a performance penalty. The baseline configuration incurs a 5% performance degradation for SpecINT due to the addition of the WCU. Figure 4 shows the relative execution speed for a 16–way set associative L1 cache using the various techniques. It can be observed that the PWCU and PAWCU techniques have the best performance of all simulated configurations, with a relative speed of 96%, on average, for the SpecINT benchmarks. No significant speedup was observed, on average, for the SpecFP benchmarks, although some benchmarks showed an improvement. The WCU designs have better performance because in case of a successful prediction for the narrow AGU, no extra cycles are spent accessing the Way Cache. The execution time is faster than that of the baseline configuration for PAWCU (shown as less than 100%). This almost completely eliminates the baseline performance degradation. The difference in speed between the PWCU and PAWCU configurations is less than 0.1%.

The average relative dynamic, static and total energy for the same 16 way set associative L1 are presented in Figure 5. Of the WCU designs, the PAWCU configuration has the best dynamic energy savings due to the fact that it has an almost identical Way Cache hit rate as the baseline WCU configuration. This is explained by the fact that the PAWCU configuration accesses the Way Cache regardless of the AGU prediction results. Thus, from the Way Cache access point of view, the only difference between the PAWCU and WCU designs is timing.

The MRU configurations save less dynamic energy due to a lower way prediction rate. This can be observed for benchmarks like apsi, art, galgel and mgrid where the dynamic energy savings of the MRU configurations are dramatically less than that of the WCU configurations. MRU predictors have a worst case behavior when different lines from the same set are read/written in consecutive accesses. This type of access occurs in some benchmarks and it is reflected in

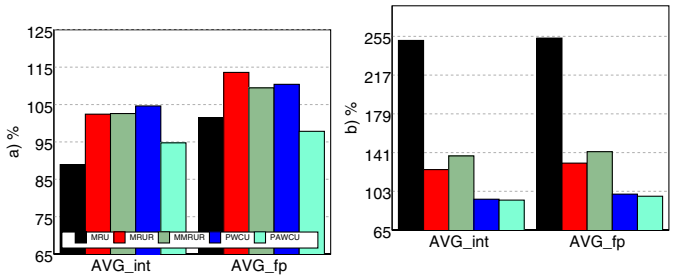

Fig. 6. Relative a)Dynamic and b)Static Energy for a 4way L1 using a 16entry WCU

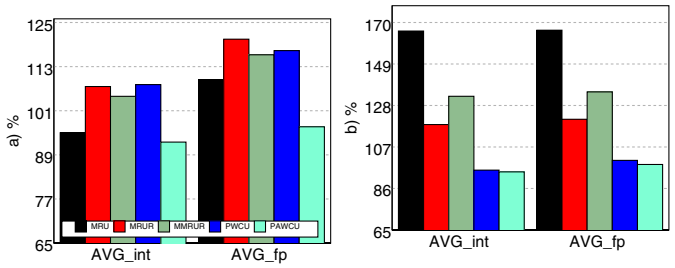

Fig. 7. Relative a)Dynamic and b)Static Energy for a 8way L1 using a 16entry WCU

the higher missprediction rate of the MRU predictors for benchmarks like art, fma3d and mgrid. The effect of the improvement in the prediction rate due to using a Multiple–MRU predictor can be observed in the results for the mgrid benchmark: 320% for MRU vs 225% for MMRUR.

Due to the periodic reset of the predictor, the MRUR configuration has lower dynamic energy savings compared to MRU: 125% vs 115% on the average for SpecINT. The dynamic energy savings of the multiple MRU predictor configuration are also throttled by the reset policy and are, on average, just 1% better than MRU for SpecINT.

The PAWCU configuration saves more dynamic energy when compared to the PWCU because it has a higher number of WCU hits. The gap, fma3d and wupwise benchmarks have low AGU predictor hit rates, and this is reflected in the fact that for these benchmarks the dynamic energy savings for the PWCU configuration are smaller than the ones for the PAWCU configuration.

The static energy results show that the MRU-noreset configuration has for most benchmarks the lowest savings due to the fact that it keeps the highest number of lines in the high leakage state (one line per set). The static energy includes the energy overhead incurred while waking up drowsy lines. As a general trend the WCU configurations have higher static energy savings due to the fact that they only keep 16 cache lines in the normal state, i.e. one line per WCU entry. The MMRUR design saves less static energy compared to the MRUR design due to the fact that it keeps more lines in the active state.

The total data cache energy consumed by the PAWCU configuration is the lowest total energy of all configurations at 94%. Recall that the baseline configuration also used WCU, so it is already energy optimized. The best MRU configuration

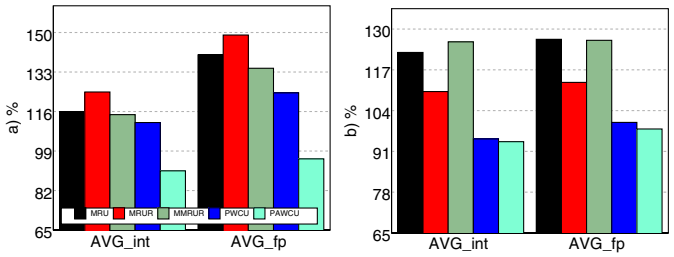

Fig. 8. Relative a)Dynamic and b)Static Energy for a 16way L1 using a 16entry WCU

(MMRUR), on the other hand, consumes 25% more energy than the baseline.

The Energy–Delay Product and Energy–Delay^2 Product results in Figure 4 follow the energy savings trend, but the WCU designs have higher savings due to their better performance i.e. a smaller delay factor in the energy-delay products. The Energy-Delay^2 Product for the PAWCU configuration is the lowest at 90%. The best MRU configuration (MMRUR), on the other hand, has a 27% higher ED^2 than the baseline.

The accuracy of the MRU predictor decreases with increased L1 cache associativity, whereas the Way Cache accuracy is not influenced by the cache associativity. All the results presented up to now were for the 16–way set associative cache. The decreasing efficiency of the MRU predictor can be observed in results for relative dynamic and static energy savings in Figures 6– 8 for 4, 8 and 16–way set associative caches. It can be observed that for 4–way caches the MRU design has better dynamic energy savings for SpecINT than the WCU configurations. For higher associativities or for SpecFP this advantage disappears. The WCU approach always has highest static energy savings.

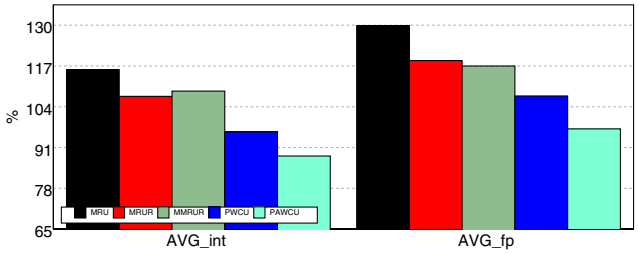

Fig. 9. Relative ED2P for a 4way L1 using a 16entry WCU

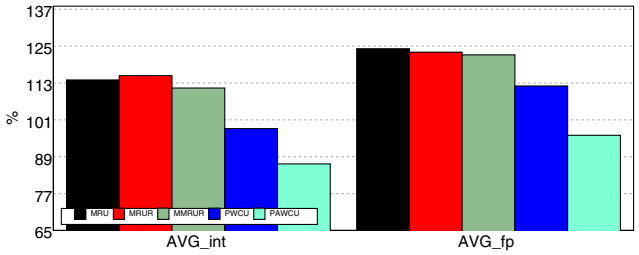

Fig. 10. Relative ED2P for a 8way L1 using a 16entry WCU

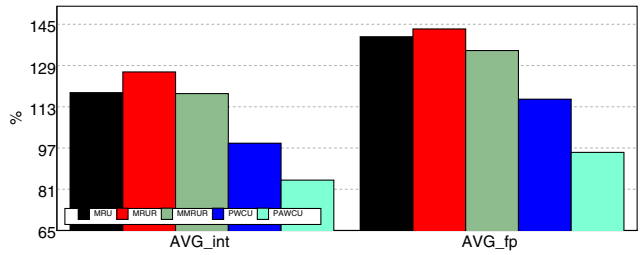

Fig. 11. Relative ED2P for a 16way L1 using a 16entry WCU

IV. Conclusion

This paper introduced a novel architecture that enables significant dynamic and static energy savings for highly associative L1 data caches. The proposed technique needs only a small amount of additional hardware: a 128-entry 1 bit predictor, one 16- and one 3-bit adder and a 16-entry way cache (i.e. a 16-entry CAM). Using fast address generation and pipelined way cache look up, the performance penalty introduced by way cache lookup was reduced to almost zero.

By putting all cache lines into drowsy mode except those that are present in the way cache, this technique achieves, on average, 30% more static energy reduction than using Multiple MRU with reset for a 16-way set associative L1 data cache. When the technique is applied to the 64KB cache with 64-byte lines, at most 1.5% (16 out of 1024) of the lines are kept active. Therefore, it is likely that no further reduction in static energy consumption is possible using the state–preserving techniques. Furthermore, it is hardly worth reducing the L1 cache static energy any further. The way cache also enables us to avoid the parallel look up of all ways in a set during a cache access. This results, on average, in 27% more dynamic energy saving than for Multiple MRU with reset for the above cache configuration. The WCU architectures that is always accessed regardless of fast or regular AGU prediction/use is shown to deliver the highest average savings and the best energy-delay product.

Acknowledgment

The authors would like to thank Rubén González, Adrián Cristal for fruitful discussions.

References

[1] G. Hinton and et al, "The microarchitecture of the Pentium 4 processor," *Intel Technology Journal*, vol. 4, 2001.
[2] D. Boggs and et al, "The microarchitecture of the Intel Pentium4 processor on 90nm technology," *Intel Technology Journal*, Feb. 2004.
[3] J. Davis and et al, "A 5.6GHz 64KB dual-read data cache for the POWER6 processor," in *ISSCC*, 2006.
[4] S. Borkar, "Low power design challenges for the decade (invited talk)," in *ASP–DAC*, 2001.
[5] K. Flautner and et al, "Drowsy caches: simple techniques for reducing leakage power," in *ISCA*, 2002.
[6] D. Nicolaescu and et al, "Reducing power consumption for high-associativity data caches in embedded processors," in *DATE*, 2003.
[7] R. Min, W.-B. Jone, and Y. Hu, "Location cache: a low-power l2 cache system," in *ISLPED*, 2004.
[8] K. Inoue and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," in *ISLPED*, 1999.
[9] C. McNairy and D. Soltis, "Itanium 2 processor microarchitecture," *IEEE Micro*, vol. 23, no. 2, pp. 44–55, Mar./Apr. 2003.
[10] *MIPS R10000 Microprocessor User's Manual*, MIPS Technologies, Inc., 1996, version 2.0.
[11] R. E. Kessler, "The Alpha 21264 microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, Mar./Apr. 1999.
[12] W. Tang and et al, "Simultaneous way-footprint prediction and branch prediction for energy savings in set-associative instruction caches," in *WPMRT*, 2001.
[13] A. Veidenbaum and D. Nicolaescu, "Low energy, highly-associative cache design for embedded processors," in *ICCD*, 2004.
[14] M. Powel and et al, "Gated-v_{dd}: A circuit technique to reduce leakage in deep-submicron cache memories," in *ISLPED*, 2000.
[15] Y. Li and et al, "State-preserving vs. non-state-preserving leakage control in caches," in *DATE*, 2004.
[16] S. Kaxiras and et al, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *ISCA*, 2001.
[17] E. S. Tam and et al, "Active management of data caches by exploiting reuse information," *IEEE TC*, vol. 48, no. 11, pp. 1244–1259, 1999.
[18] S. Petit, J. Sahuquillo, J. M. Such, and D. Kaeli, "Exploiting temporal locality in drowsy cache policies," in *CF*, 2005.
[19] E. Witchel and et al, "Direct addressed caches for reduced power consumption," in *MICRO-34*, 2001.
[20] R. Gonzalez and et al, "A content aware integer register file organization," in *ISCA*, 2004.
[21] D. Burger and T. M. Austin, "The SimpleScalar tool set," University of Wisconsin, Tech. Rep. TR-97-1342, 1997.
[22] "Cacti4," http://quid.hpl.hp.com:9081/cacti/.
[23] K. So and R. Rechtshaffen, "Cache operations by mru change," in *ICCD*, 1986.