IPC Driven Dynamic Associative Cache Architecture for Low energy

SRIRAM NADATHUR, AKHILESH TYAGI
Department of Electrical & Computer Engineering
Iowa State University, Ames, Iowa 50011
{sriramgn,tyagi}@iastate.edu

Abstract

Existing schemes for cache energy optimization incorporate a limited degree of dynamic associativity: either direct mapped or full available associativity (say 4-way). In this paper, we explore a more general design space for dynamic associativity (for a 4-way associative cache, consider 1-way, 2-way, and 4-way associative accesses). The other major departure is in the associativity control mechanism. We use the actual instruction level parallelism exhibited by the instructions surrounding a given load to classify it as an IPC k load (for $1 \leq k \leq IW$ with an issue width of IW) in a superscalar architecture. The lookup schedule is fixed in advance for each IPC classifier $1 \leq k \leq IW$. The schedules are as way-disjoint as possible for load/stores with different IPC classifications. The energy savings over SPEC2000 CPU benchmarks average 28.6% for a 32KB, 4-way, L-1 data cache. The resulting performance (IPC) degradation from the dynamic way schedule is restricted to less than 2.25%, mainly because IPC based placement ends up being an excellent classifier.

1. Introduction

Energy efficiency within a computing system is a desirable characteristic both from a fundamental computing optimality as well as from system engineering perspective. Set associative caches (currently 2-way to 4-way) are widely deployed in processors due to their ability to lower miss rates with acceptable cycle times. The program characteristics have been evolving to include a larger number of working sets of larger sizes over time. This trend favors higher associativities in the future. A full associativity access (k-way associative access for a k-way cache) switches the capacitance in all the ways simultaneously resulting in a maximum energy access per load. Several earlier research efforts [4], [1], [12], [10] used different prediction and control mechanisms to complete some of the load/store instructions within 1-way lookup (direct mapped access). If that misses, a full associativity access is generated. Hence, some of the loads switch the capacitance of only one-way (approximately $1/k$th the capacitance of all the k ways to a first order approximation) instead of all the designed ways. This saving is significant given the cache energy's contribution towards the total processor energy. Almost all current generation processors, including Alpha [8], PentiumPro [9], StrongARM [6] and XScale [5], dissipate from 15% to 45% of their total energy in caches. In this paper, we focus on an energy-efficient scheme for L1 data cache based on dynamic associativity control driven by an IPC (instructions per cycle) classification[1].

The primary motivation behind the proposed cache architecture is to design microarchitecture components that consume energy in proportion to the delivered work. The incumbent microarchitecture design paradigm targets a peak instruction-level parallelism (ILP) such as 4 or 6 or 8. Each component of the microarchitecture is designed to sustain this targeted peak ILP. However, typical programs exhibit a high variance in IPC over time. Moreover, this variance has short temporal and spatial periods. In other words, the IPC variance is visible within extremely small time windows (of less than 10 cycles) at each stage in microarchitecture. This leads to a designed capacitance proportional to the peak IPC, which switches every cycle. Ideally, the switched capacitance would be proportional to the actual IPC (ILP) delivered in a given cycle. Hence, a commitment of silicon resources proportional to the peak IPC hurts both the delay and energy performance for every cycle with IPC less than the peak. One possible solution for this dilemma is to design microarchitecture components that switch capacitance proportional to the delivered IPC leading to a delay and energy performance in line with the actual program progress (IPC for that cycle) [11]. This constitutes the motivation for our work.

The load/store instructions occur in epochs with variable IPC. Some loads belong to high ILP regions of a pro-

[1]This work is supported by NSF grants CCR 024222 & CCR 0209078

gram, and some to the low ILP regions. A high IPC exposes the cache to a larger number of working sets, many of which are conflicting. Let each load/store be classified with the ILP of the constituent program region. The conflicting (not direct-mapped) working sets are the ones to generate pressure on a data cache for higher associativity needs. They constitute a certain percentage of all the loads for a given program (on average) and this percentage appears to be quite predictable according to [2]. Again, assuming for intuitive simplicity, that the conflicting loads are uniformly distributed across the program, a higher ILP epoch would tend to have a higher number of conflicting loads. Hence, higher associativity would support a higher ILP program region more naturally. Similarly, a lower ILP region could still be supported with a lower associativity with an acceptable miss rate. This says that the number of instantiated ways can be proportional to the ILP of the program region around a given load. A simplistic scheme would be to map all the load/stores within ILP 1 region to Way-0, within ILP 2 region to Way 0 and Way 1, within ILP 3 region to Way 0, Way 1 and Way 2; and within ILP 4 region to all the four ways. This simplistic notion forms the basis for this paper, and is refined later to make it feasible. Note that such a mapping from IPC classification to the instantiated ways has the targeted property of energy dissipation being proportional the ILP.

Related Work: Many researchers have studied techniques to alleviate both cache energy and access time bottlenecks. Techniques encompass partitioning, decomposition and sequentializing way access patterns. Albonesi [1] proposed a technique to partition ways selectively. Sequentializing way-access patterns was proposed by Grunwald *et al.* [4]. Circuit design techniques [7] were also proposed to conserve cache energy. Vijaykumar *et al.* [10] combined predictive schemes with selective access techniques to achieve low energy without compromising on access times.

Overview: The fundamental difference between the proposed scheme and the earlier schemes lies in their intrinsic goals. Even though all the existing schemes try to reduce energy based on the data set requirements, none of them aim at dissipating only as much energy as the work done (IPC delivered). We propose a selective, sequential cache architecture with the explicit objective of adapting its energy needs to the ILP. The ILP could be measured at one of many microarchitecture stages. The issue stage select logic is utilized as the dynamic classifier of the load/store ILP region. The sequential-way-access schedule is fixed *a priori* for each ILP classification based on two factors. The first factor is the distribution of load/stores among the different IPC epochs. This determines the tolerance of a given IPC class loads to a mismatched schedule. The second factor is the distribution of different associativity needs among the loads classified as IPC k for all k. A sequential access schedule (ordering of cache ways) is chosen for each IPC classification on the basis of this distribution. For instance, the schedule $[(0,1);(2,3)]$ calls for probing Ways 0 & 2 in the first cycle; if that results in a miss then Ways 2 & 3 are probed in the next cycle.

The issue stage classifies each load/store as belonging to one of the IPC epochs. If k instructions are selected by the issue wakeup/select logic in a given cycle, then all the load/store instructions among these k instructions are classified as IPC-k load/stores (Figure 2). When a load/store is issued from the load/store queue (LSQ) to the cache, its IPC classification (performed earlier at the issue stage) determines its sequential schedule, say $[(0,1);(2,3)]$. It accesses only one (or two) ways out of the four available ways. Only the tags from the accessed ways are compared. If a miss is indicated, a tag comparison is initiated in the other ways sequentially, before a final miss is signaled. Data access is initiated in each way in parallel with the tag comparison as usual. Correct data placement is fundamental to the efficiency of any sequential access scheme. The same schedule is also used for data placement. We observe in Section 2, that the dynamic IPC at issue stage ends up being a good classifier for load/store instructions, resulting in a good placement with in the cache ways (with little cross-talk between different ways).

Organization: Section 2 details the proposed cache architecture and its access algorithm. Section 3 qualitatively compares the energy dissipation of the proposed scheme against a base cache model and quantifies the impact on performance, through Simplescalar [3] simulations. Section 4 describes the experimental setup and results. Section 5 concludes the paper.

2 IPC Driven Dynamic Associativity Management

This section describes the adaptive associativity cache management algorithms and schemes. Figure 2 presents a schema for the proposed scheme.

IPC based Load/Store Classification: Load/store instructions involve two operations: effective address computation followed by the load/store dispatch from the LSQ. The issue stage wakeup & select logic wakes up all the instructions whose source operands just became available. These instructions are placed in the ready queue. We use

the number of selected instructions at the issue stage to annotate each load/store instruction among these selected instructions as shown in Figure 2. The number of selected instructions at this stage can vary from 0 to k (for a k-wide microarchitecture). If there is a load/store among the selected instructions, then this number of selected instructions ranges from 1 to k. Hence each load/store gets placed into one of the k IPC buckets. Figure 1 shows an example of this classification. The load sitting in the issue ready queue is selected along with the two preceding instructions Insn 1 and Insn 2 and is annotated as IPC-3 class load/store. This annotation is carried along with the load all the way until the cache access.

Binding of Sequential Access Schedule: The sequential access schedule specifying the temporal ordering of the ways for a given IPC classification can be either hardwired into the cache control or could be dynamically bound. A late binding results in greater generality allowing for per process (through compiler analysis) or even per procedure specification of the access schedule. The block containing the access schedules in Figure 2 is meant to represent the dynamic binding. The compiler could store such a sequential access schedule table into a memory mapped region. At the process initiation, the table could be read into a microarchitectural table, which is read by the cache controller to initialize the sequential access schedule.

Dynamic Associativity Cache Access: When a load, is issued to the data cache subsystem from the LSQ, its IPC bits are used to decode the sequential access schedule table in parallel to retrieve the temporal way access masks into the selected way mask register (as shown in Figure 2). In the first access cycle, Cycle I way mask is used to enable/disable the chosen ways. The Selected Way Mask register shifts down the temporal access schedules down by one position so that the Cycle II way mask is at the head of the register. If this is a hit, the access is done. Otherwise, Cycle II way mask drives the second cycle access way enable/disable signals. Similarly, on a miss, the Cycle III way mask is utilized for a third cycle access. A miss at this point signals a cache miss.

Sequential Access Schedule Determination: How do we determine a good schedule for each IPC classification? Note that as we observed earlier, the intuitive explanation for the IPC based classification is that the fraction of conflicting loads scales linearly within a group of k instructions. Hence, a higher associativity access benefits a higher ILP load. A good access schedule needs to determine the distribution of conflicting loads with respect the IPC classification. We also need to understand the distribution of loads into different IPC classifications.

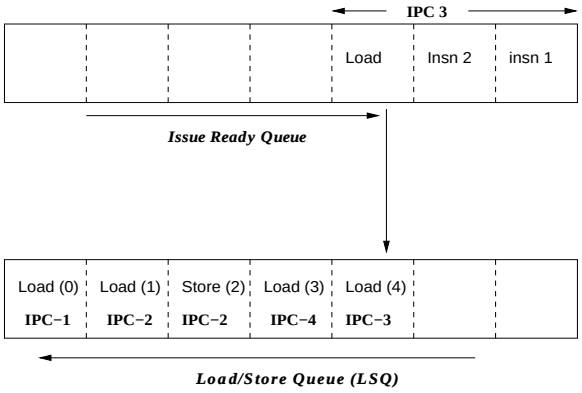

Figure 1. Issue Select Stage Classifies the IPC of Load/Store Instructions

Consider a 4-way associative data cache within a 4 issue superscalar microarchitecture. In order to determine the distribution of load/store instructions into IPC-1, IPC-2, IPC-3, and IPC-4 buckets, we classified load/store instructions in five SPEC 2000 benchmarks: gcc, mesa, equake, gzip and bzip according to issue IPC (Figure 3). Approximately 60-65% of load/store instructions are issued in an IPC-4 group, whereas 20% are issued in an IPC-3 group. The frequency of IPC-2 and IPC-1 load/store instructions was approximately 15% and 5% respectively.

We also need to know the associativity needs of the loads classified as IPC-k in the following sense. Consider all the IPC-4 classified loads. Some of these loads are direct-mappable — they are non-conflicting with respect to a 1-way associative access. We denote this class of loads by $IPC_{4,1}$. Some of these loads are non-conflicting with a 2-way associative access. These loads are denoted by $IPC_{4,2}$. In this way we can classify all the loads into 16 classes. This helps us determine a suitable sequential way access schedule as follows. If $IPC_{4,4}$ dominates the other load sets (highest frequency) then the access schedule for IPC-4 classified loads ought to be a single 4-way access $[(0, 1, 2, 3)]$.

In order to assess $IPC_{i,j}$ for $1 \leq i, j \leq 4$, we counted the number of accesses that map to the same set over a reasonable temporal window. We followed an experimental scheme similar to the one in [2] to derive this data. We implemented the most restrictive version of the cache – direct mapped. We maintain a buffer where all the replaced loads are placed. At certain time intervals, (these are the temporal windows within which the working sets are being captured), we analyze this buffer to see how many loads map into the same set to determine the minimum as-

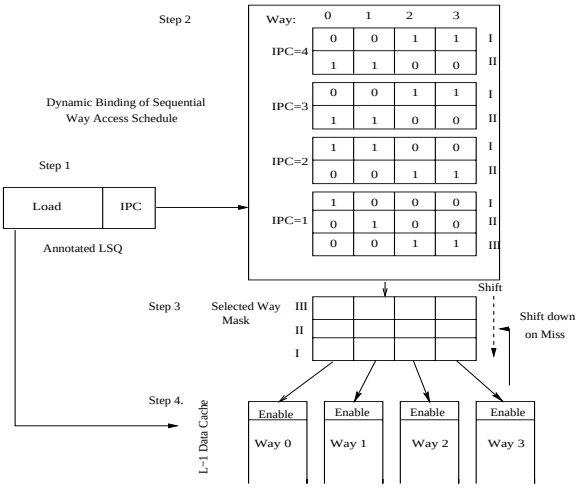

Figure 2. Implementing Sequential Way Access

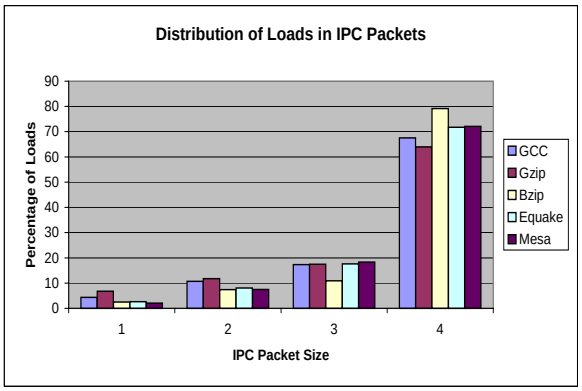

Figure 3. Issue IPC Based Classification of Loads in SPEC2000 Benchmarks

sociativity that will make these loads non-conflicting and we update the respective counter. The resulting observations are shown in Figure 4. The four sub-groups along x-axis are for IPC-1 through IPC-4 classified load/store instructions. Within each sub-group IPC-i, the bars denote $IPC_{i,1}$, $IPC_{i,2}$, $IPC_{i,3}$, $IPC_{i,4}$ from left to right respectively. Note that direct-mapped accesses dominate in each IPC class. However, more interestingly, among the conflicting load/store instructions, 2-way accesses dominate by far for all the four IPC classifications. The 3-way and 4-way accesses in all cases are very rare. This leads us to the conclusion that all the accesses in our sequential way access schedules will be limited to be at most 2-way associative. This, however, begs the following question: why should we ever consider a cache that is more than 2-way associative? In a traditional cache organization, the replacement policy does not make an effort to maintain the multiple working sets orthogonal (not intertwined), if it is possible to do so. Hence, a composite footprint of multiple 2-way associative working sets might appear as requiring 4-way or higher associativity. We believe, that it is this ability of the IPC based classification, to maintain the limited associativity working sets in isolation, that results in its effectiveness despite limiting its schedules to be at most 2-way associative.

The insights gained from the two sets of data (in Figures 3 and 4) are combined to derive the sequential access schedule in Table 1. We tweaked the search space around this point for different access schedules, but these parameters gave us our best energy-delay trade-off so far.

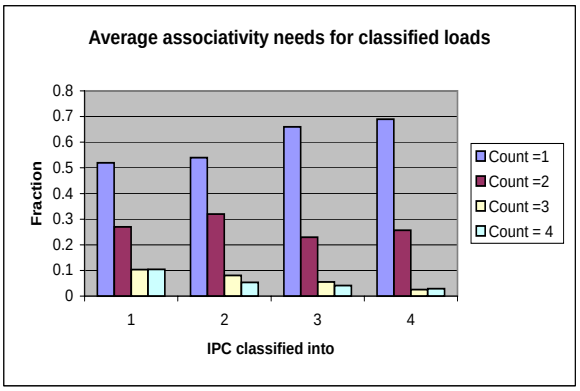

Figure 4. Distribution of associativity requirements at each IPC epoch

IPC Class	I way enabled	II way enabled	III way enabled
1	Way 0	Way 1	(Way 2,3)
2	(Way 0,Way 1)	(Way 2,Way 3)	
3	(Way 2,Way 3)	(Way 0,Way 1)	
4	(Way 2,Way 3)	(Way 0,Way 1)	

Table 1. Sequential Way Access Schedule

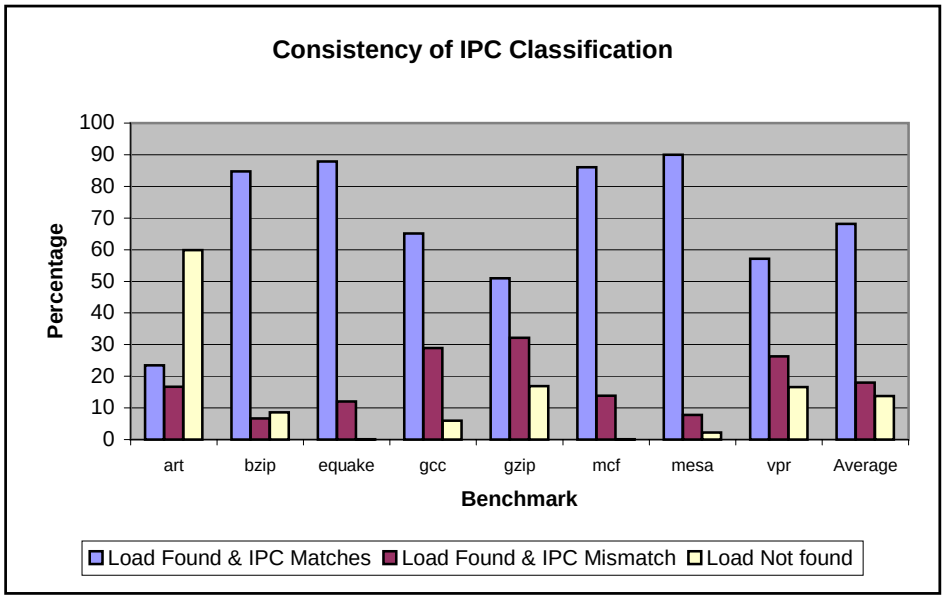

Figure 5. Consistency of IPC Classification

Placement & Lookup Algorithm: Both the placement & lookup proceed as follows. If the IPC classifier is 1, only Way 0 is used for tag comparison and data access. If there is a miss, then Way 1 is enabled in the next cycle. If there is a miss there as well, then (Way 2, Way 3) are enabled in the next cycle. A miss here results in L1 miss leading to an L2 access. This creates a pseudo-associative cache controlled by IPC classifiers. There are multiple hit times (in this example three, only two for the most frequent case of IPC 4). If desired, all the IPC classifiers can be reduced to only two cycle access only. Note that on a miss, the data is also placed into a specific way according to the way-ordering in Table 1. We currently place the data in direct-mapped way by forcing it into the first way enabled mapping. Based on the PSAC [4] observation, we expected to see at least 70-75% of the cache traffic completely contained within single ways with no cross traffic (only one cycle hit). Figures 3 and 4 show this figure to be close to 66%. Note that we do not use any prediction that comes in the critical path for cache access. The IPC classifiers can also be compiler generated, and LSQ validated. In this study, we have only considered dynamically generated (by issue stage) IPC classifiers.

2.1 Consistency of IPC Classification

Since we rely on classifying loads at run time, it is important that a load classified as belonging to a particular IPC epoch is classified as belonging to the same IPC epoch for all its occurrences (if it appears again). If not, the recurring load can probe a different way, leading to misses as well as energy degradation. The methodology to capture the consistency of IPC classification is as follows. We employ a 1024 entry buffer to store the replaced loads. Every time a load occurs, the buffer is checked to see if it has occurred before. If so, the current IPC classification is verified with the previous IPC classification. If they are the same, a counter is incremented; if different, another counter is incremented. If the load is not found in the buffer, a new entry is created for the load. Note that such a scheme captures IPC classification consistency within a time window corresponding to 1024 misses. This is a significantly large temporal window (with a 99% hit rate, and 20% load/store frequency, it captures a temporal window with 512000 instructions).

The observations from this experiment are shown in Figure 5. It is seen that, on average, the fraction of loads getting reclassified as belonging to the same IPC epoch is about 65% to 70%. The mis-classified loads are a mere 16%. This vouches for the consistency of the IPC based classification methodology.

3 Cache Energy-Delay Model

The N-way set associative cache has N Static-RAM arrays each for data and tag. For a given CPU address, tag decoding and the data accesses are both performed in parallel. Every cache access results in decoding the address

issued by the CPU, which asserts exactly one of the word-lines. A word-line is common to all 4 ways since a single set-address designator corresponds to all the ways in a given set. The bit lines are changed to reflect the selected bit cell state, followed by a sense amplifier that accelerates this change. Since the wordlines stretch across all four ways, their capacitance contributes significantly to the energy (E_{wl} per way) and delay. For every access, since tag resolution would not be completed before data is available in the data-output drivers, energy (E_{bl} per way) is spent in the bit lines of all four ways (of a set). Added to that is the energy spent in redundant tag comparisons (E_{cmp}) and sense amplifiers (E_{sa}). Once tag resolution is done, the energy is spent in the multiplexor that drives one of the data output drivers (E_{drv}). The energy spent per access is developed as follows. Let us define the array energy to be: $E_{arr} = E_{wl} + E_{bl}\ (+\ E_{cmp})$. There are constant components of energy (E_{cons}) that involve the drivers, inverters and multiplexors. Then, the energy per access is roughly given by:

$$E = (E_{dec}+4*E_{arr})_{tag}+(E_{dec}+4*E_{arr})_{data}+E_{cons}$$

Even if the access results in a hit in the cache, almost three fourths of the total energy spent is redundant.

If the tag and data array accesses were to be sequentialized as proposed in Section 2, the schedule selected data array accesses are initiated in parallel with tag access/comparison. Note that both tag and data access proceed only in the scheduled way and not in all the ways. Now that the decoders drive smaller number of ways, cache access delay is reduced (set associative mapping tends towards direct mapped cache architecture). The data and tag array energy, in the 1-way (2-way) access schedule would be roughly 25% (-50%) of the original energy. In the best case, a 1-way access may result in a hit. But the scheme can result in two or three cycle accesses as well. In general, for a n cycle hit, where $n = 1, 2$ or 3, the energy is roughly given by:

$$E = n*[(E_{dec}+E_{arr})_{tag}+(E_{dec}+E_{arr})_{data}]+E_{cons}$$

For reasons discussed in Section 2 and validated in Section 4, the proposed scheme results in good placement, which leads to a single cycle access for most loads. Hence, the energy saved in the arrays during every single cycle access offsets the extra energy used in the decoders during every multi-cycle hit. These energy computations, for varying hit times are formulated in Table 2.

The influence of capacitance reduction is not just evident in energy savings but also manifests itself in cache access latency reduction. For every cycle, the proposed schedule accesses a maximum of two ways, instead of four. Even in a cache design that involves partitioned word lines, the drive necessary is roughly one half of what is needed in a four-way access design. Hence the cache access latency comes down significantly. CACTI based simulations quantize this reduction to be 13% (from $1.695ns$ to $1.475ns$).

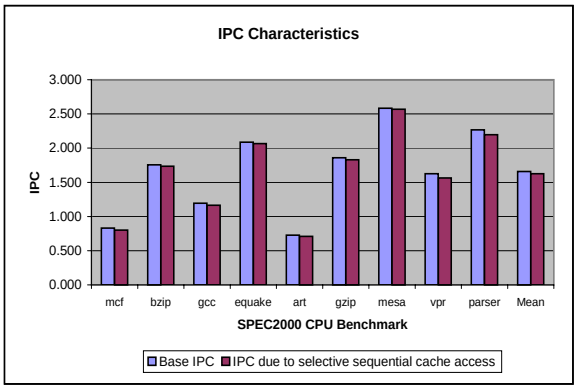

Figure 6. Performance Characteristics of IPC driven cache architecture

4 Experimental Methodology

The experimental evaluation of the modified cache control is simulation based. The first exercise determined the distribution of load/stores among different IPC classifications in a base model (Figure 3). These distribution numbers were critical in developing a practical way-access schedule detailed in Section 2. The performance and energy benefits of the selective way-access schedule were then evaluated as follows.

4.1 Simulation Environment

The SimpleScalar simulator [3] version 3.0. targeting Alpha ISA [8] is employed to evaluate the IPC driven sequential selective cache access scheme. The simulation models a current generation 4-way dynamically scheduled processor microarchitecture with two levels of instruction and data cache memories. We employ nine SPEC2000 benchmarks to compare the performance of the proposed microarchitecture against base models.

The baseline model can fetch and issue up to 4 instructions per cycle. The dynamic scheduler window is 64 entries deep and the load/store queue can store up to 32 entries. The memory system consists of split Level-1 caches, the L-1 data cache being a 32KB, 4-way set-associative cache and the L-1 instruction cache being a 64KB direct mapped cache. Data cache access is assumed to have a 3 cycle latency. There is also a 256KB 4-way set associative unified L-2 cache with a 6 cycle hit latency. Memory access is assumed to take takes 60 cycles. The model uses a 2 level

IPC classifier	First Cycle Hit	Second Cycle Hit	Third Cycle Hit
1	$E_{dec} + E_{arr} + E_{cons}$	$2 * E_{dec} + 2 * (E_{arr}) + E_{cons}$	$3 * E_{dec} + 4 * (E_{arr}) + E_{cons}$
2,3,4	$E_{dec} + 2 * (E_{arr}) + E_{cons}$	$2 * E_{dec} + 4 * (E_{arr}) + E_{cons}$	

Table 2. Energy Consumption Methodology

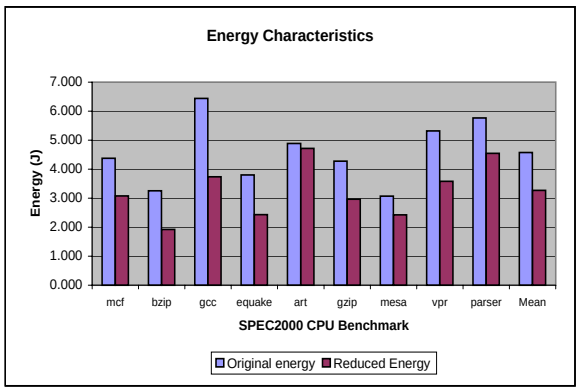

Figure 7. Energy Characteristics of IPC driven cache architecture

branch prediction mechanism called *gshare* branch predictor which uses an 8-bit global history table and a 4K entry BTB. The simulations were run for at least 500 million instructions and all simulations are appropriately forwarded through the transient phases based on [12]. Energy numbers are related to the number of cache accesses and more specifically to the number of ways accesses. The energy per access is quantified through CACTI 2.0 [13] based simulations. CACTI also provides a breakup of L-1 data cache access energy which is utilized to compute the energy gains from the proposed design based on Table 2.

4.2 Performance Characteristics

The metric used to compare the performance of the proposed scheme against the base scheme is Instructions per Cycle (IPC), which is reflective of the ILP delivered. The base microarchitecture is evaluated separately without accounting for any penalty attributable to mis-placement, which leads to multi-cycle accesses. The proposed classification scheme is integrated into the Simplescalar environment by adding a detailed IPC-aware sequential cache probing/placement scheme for every L-1 data cache access. The IPC based classifier is implemented in the scheduler stage to guide cache access. The only contribution to performance degradation could be attributable to any load that takes more cycles than usual, to return data.

The performance degradation numbers are reported in Table 3. Figure 6 shows that degradation remains within conservative limits of 2.25% and is even around 1% for four benchmarks. These numbers indicate that the proposed IPC based classification leads to a better overall energy-performance design point.

4.3 Energy Characteristics

CACTI 2.0 [13] based simulations of a 32 KB, 4-way set associative cache (with a line size of 16 bytes) are used to obtain the energy-delay benefits of the proposed scheme. We obtain a breakup of total energy/delay in terms of the decoder, wordline, bitline and sense-amp components for both tag and data arrays. The energy associated with the comparators (tag), multiplexors and drivers are also recorded. Note that these are the per-access energy components for the L-1 cache. The respective energy components are scaled according to the number of accesses to the different ways obtained from Simplescalar simulations.

The implementation of the proposed scheme in Simplescalar simulator also takes care of decomposing the scaling factors for sequential access schedules. This becomes necessary because, every time a load results in a first cycle miss, the decoder has to be exercised again to access the next way in the schedule. This doubles the decoder energy for that load. At the same time, every access might result in the switching of one or two ways, depending on the IPC-epoch it originates from. The net energy for the whole simulation cycle time is obtained as the product of the energy per access obtained from CACTI and the number of accesses, obtained from the simulator modeling. These figures are tabulated in Table 3 and shown in Figure 7.

Over nine benchmarks, the average savings for the L-1 data cache energy is about 28.6%. In a majority of the benchmarks, the savings are at least 20% or higher. Note that the performance/IPC compromised to achieve these significant energy-cuts are a mere 2% on an average. Coupled with the fact, that the scheme has no negative implications on the cycle time makes it an impressive scheme for effective energy reduction in caches.

Spec2000 Benchmark	Base IPC	Reduced IPC	% IPC Deg	Energy (nJ)		% Energy Saved
				Original	Reduced	
mcf	0.829	0.801	3.401	4.373	3.075	29.677
bzip	1.755	1.736	1.128	3.253	1.921	40.933
gcc	1.194	1.165	2.454	6.441	3.740	41.939
equake	2.088	2.065	1.092	3.802	2.431	36.067
art	0.727	0.709	2.395	4.883	4.715	3.442
gzip	1.859	1.828	1.667	4.276	2.967	30.624
mesa	2.583	2.568	0.612	3.069	2.425	20.971
vpr	1.625	1.565	3.729	5.319	3.582	32.649
parser	2.267	2.197	3.079	5.766	4.548	21.133
Mean	1.659	1.626	**2.173**	4.576	3.267	**28.604**

Table 3. Performance and Energy Characteristics

5 Conclusions & Future Work

In this paper, an IPC-aware sequential associative cache architecture is proposed. This novel cache-access methodology creates a sequential schedule for cache accesses based on the ILP delivered at every cycle. The resources allocated for cache access tend to dissipate energy in proportion to the actual ILP delivered (and not the peak ILP). The overall IPC degradation remains within tolerable limits of 2%, while the energy savings are at least 20% for a 32KB, 4-way set associative L-1 data cache. There are also cache access time advantages (13%) obtainable as a result of reduced switched capacitance. (This reduction is not leveraged in our study/results). This complements the hypothesis that least energy is achieved at a design point which balances resources and requirements.

Many interesting problems arise in the context of scalability. Consider a 2-port data cache. Can two loads with disjoint access schedules (say one accesses Ways 0 & 1 and the other accesses Ways 2 & 3), be supported by a single port (the extra decoders being the only additional cost)? Also, can the access schedules be tweaked not to result in any more IPC loss, but at the same time are more way-disjoint? We are experimenting along these directions.

References

[1] D. H. Albonesi. Selective cache ways: On-demand cache resource allocation. *International Symposium on Microarchitecture*, pages 248–, 1999.

[2] B. Batson and T. N. Vijaykumar. Reactive-associative caches. In *Proceedings of the international symposium on parallel architectures and compiler techniques (PACT)*, 2001.

[3] D. Burger, T. Austin, and S. Bennett. Evaluating the future microprocessor: The SimpleScalar Toolset. *Technical Report CS-TR-96-1308, University of Wisconsin, Computer Science Dept.*, 1997.

[4] B. Calder, D. Grunwald, and J. Emer. Predictive sequential associative cache. *Proc. 2nd Symp. High-Performance Comp. Arch., San Jose, CA*, pages 244–253, January 1996.

[5] I. Corporation. Intel xscale microarchitecture. *http://developer.intel.com/design/intelxscale*, 2001.

[6] J. M. et. al. A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor. *IEEE Journal of Solid-State Circuits*, 11(31):1703–1714, 1996.

[7] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. *ISCA*, June 2002.

[8] Keller.J. The 21264: A Superscalar Alpha processor with out-of-order execution. *Presented at the 9th Annual Microprocessor Forum, San Jose, CA.*, 1996.

[9] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. *ISCA*, pages 132–141, 1998.

[10] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture, Austin, Texas*, pages 54–65, 2001.

[11] P. Ramarao and A. Tyagi. An adiabatic framework for a low energy micro-architecture and compiler. *Workshop on interaction between Compilers and Computer Architecture (INTERACT - 7)*, 2003.

[12] T. Sherwood, E. Perelman, and B. Calder. Reducing set-associative cache energy via way-prediction and selective direct-mapping. *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture, Austin, Texas*, pages 54–65, 2001.

[13] P. Shivakumar and N. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. *Technical Report WRL 2001/2, DEC/Compaq Western Research Labs, Palo Alto, CA.*, August 2001.