RC22543 (W0206-100) June 18, 2002
Computer Science

IBM Research Report

Improving Power-Effectiveness of Cache Structures through Quenching

Ravi Nair
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Improving Power-Effectiveness of Cache Structures through Quenching

Ravi Nair
IBM Thomas J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

June 17, 2002

Abstract

The performance of most processors today is improved through appropriate use of caches, which learn and predict various pieces of information. The effectiveness of a cache generally improves as its size increases because larger caches are able to contain the frequently used information of a larger number of applications. However, large caches consume a lot of power, and much of this power is wasted when some region of an application is able to perform almost as well with a considerably smaller cache. In this paper, we analyze the behavior of a typical cache and show that applications have widely varying behavior, with phases in which the cache is heavily utilized often interleaved with phases in which large parts of the cache remain idle. We use these observations to propose a new type of cache called the *quenched cache*, which periodically quenches the cache by turning it off completely. The effect of quenching is a cache that adapts its power consumption based on actual activity, doing so without elaborate learning or feedback mechanisms in hardware. The paper presents results of experiments to show the effectiveness of quenched caches.

1. Introduction

Moore's Law has contributed to the pervasive use of caches in modern-day microprocessors. With shrinking lithography and transistor size, processor execution pipelines are getting faster. As computers are being called upon to tackle ever-bigger problems, memories are getting larger and their access times are not keeping up with the cycle time of processors. This has led to larger caches, and inevitably to multiple levels of caches even on a single chip.

Ever since their introduction in the early 60's, caches have served an increasingly useful role in processors. While the most predominant use of a cache even today is in the memory hierarchy, we are now seeing a more varied use of cache structures, as in branch prediction. The need to supply fast processor execution pipelines with instructions has led to schemes to predict control flow in application programs. As processor implementations become more sophisticated in their ability to handle instruction-level parallelism, branch prediction schemes are becoming ever more sophisticated. But central to most branch prediction schemes are two caches, the branch

history table, which records a history of outcome of conditional branch instructions, and the branch target table, which maintains a history of control flow target addresses. Today, with the advent of trace caches and prefetch buffers, the cache has transcended its early role as simply a player in the memory hierarchy. In the IBM Power4 microprocessor chip [1], for example, various forms of caching structures account for more than 80% of transistors on the chip.

Unfortunately, caches also consume a significant fraction of the power in such chips. Power is consumed not only when blocks in the cache are accessed by the processor core, but also in simply maintaining the information contained in them. Set-associative caches often improve their performance by selecting late from a set of blocks, all of which are accessed even though only one is selected. Thus there appears to be much to be gained by fundamentally changing the way caches are structured from a power consumption point of view.

In this paper, we analyze cache behavior with a view to developing an inherently power-efficient cache architecture. Ideally, any new cache architecture should allow well-understood, present-day configurations to be used without unreasonable overhead in the form of additional control complexity in the microarchitecture, or in additional requirements from software. We hope the quenched cache proposed in this paper satisfies this criterion.

The rest of this paper is organized as follows. We begin by illustrating the nature of typical activity in a D-cache in section 2. Our observation of snapshots from various benchmark programs motivates the proposal of the quenched cache described in section 3. In section 4 we show results of experiments demonstrating the effectiveness of a quenched D-cache. In section 5 we show that the concepts behind the quenched cache can be applied to cache structures beyond the D-cache. A discussion of related work in section 6 is followed by concluding remarks in section 7.

2. Characterization of Cache Activity

Figure 1 shows the profile of the number of *active* D-cache blocks, i.e. those blocks touched for read or write access, in a series of instruction windows of a processor running the *gcc* benchmark from the SPECint suite. Each instruction window executes 100K instructions in Figures 1(a) and 1(b), and 1M instructions in Figure 1(c). Figure 1(a) shows the profile for an 8K cache consisting of 128 blocks of 64 bytes each, while Figure 1(b) shows a similar profile for a cache 4 times the size. In both cases we see that there are many windows in which the number of active blocks is fewer than total available blocks. Moreover, the number of active blocks reduces as the cache size is increased. Figure 1(c) shows that when the instruction window is increased to 1M instructions, all blocks in the cache get touched in almost all the windows even for a 32K cache. A comparison of the profiles in Figures 1(b) and (c) suggests that many more of the blocks that remain active in the 1M window actually contain inactive information – information that is no longer useful, information that may not be needed for a long time, or information that may be useful but is likely to be replaced by other information due to cache size or associativity restrictions.

The profiles of activity for three other benchmarks in the SPECint suite are shown in Figure 2. It may be observed that there are small regions of some programs, such as *vortex* and *perl*, where a 32K cache gets filled up completely even for a 100K instruction interval. Yet there are several windows in all benchmarks when large portions of the cache go unutilized.

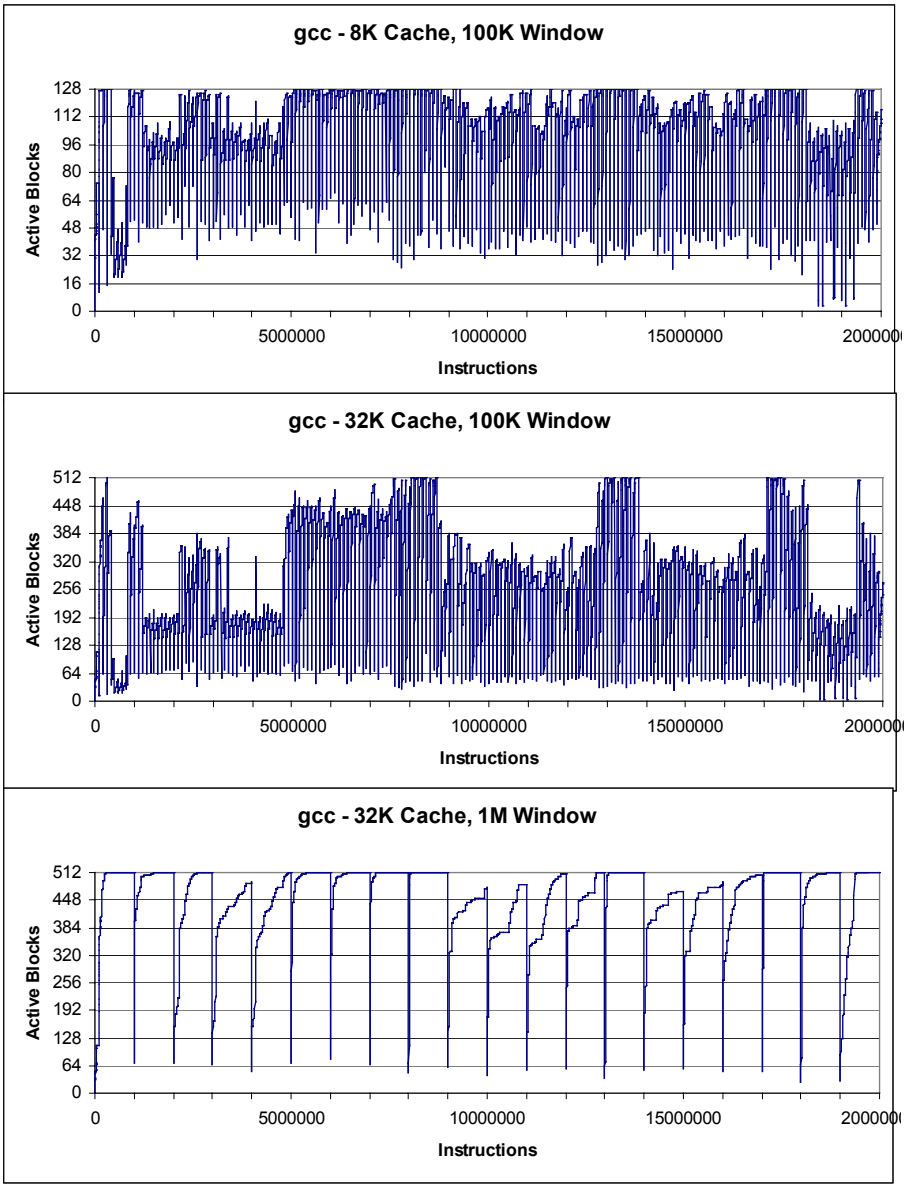


Figure 1: Profile of activity in a cache

In effect, for the applications shown, the active area of the cache is typically smaller than the whole cache, and reduces further as the size of the cache increases. Yet, in typical caches, the whole cache remains powered up all the time and consumes static power. The cache may also consume more dynamic power (that consumed during reads and writes) if the larger cache has a higher associativity compared to the smaller. While reduction of both types of power, static and dynamic, is important, we restrict our attention in this paper to the reduction of static power.

With leakage power becoming an increasing consumer of power in processors, reduction of static power could result in significant power savings of future microprocessors.

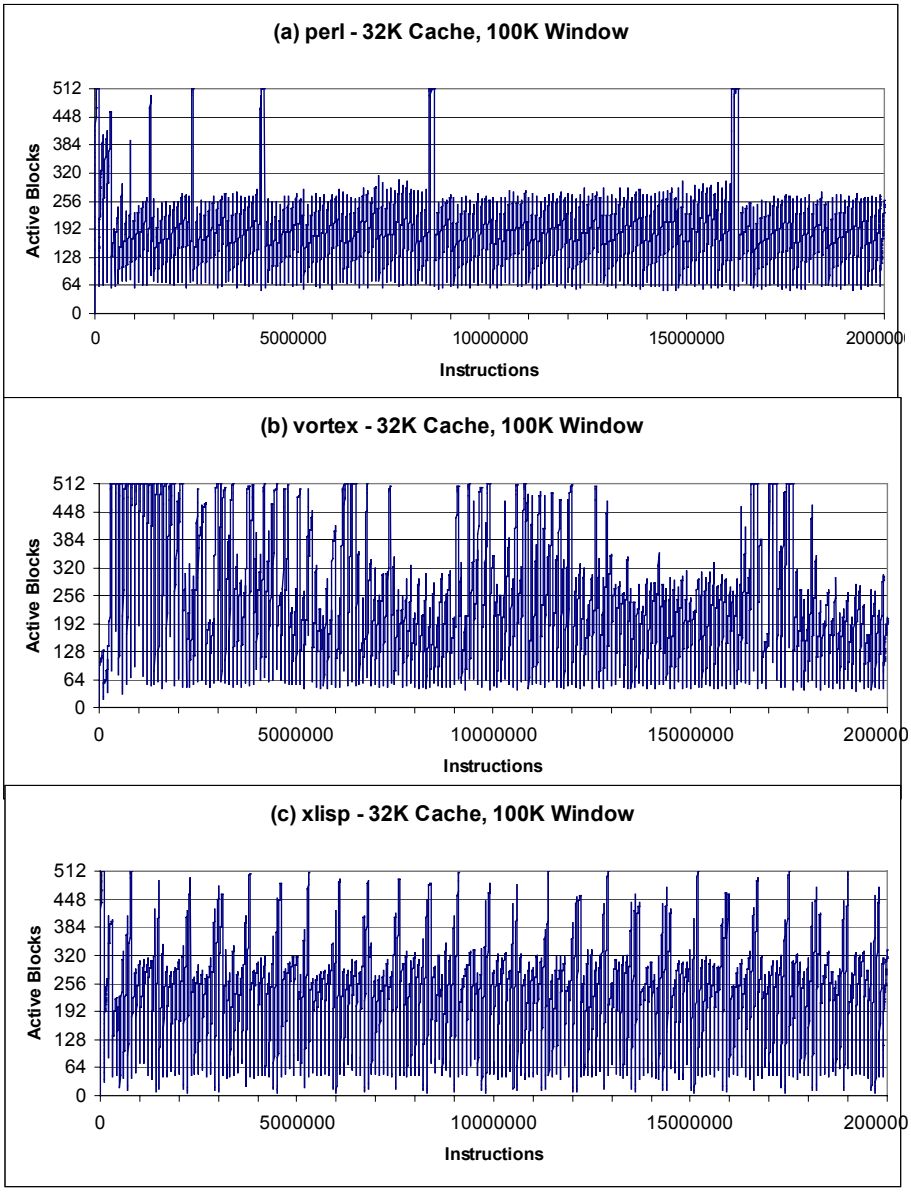


Figure 2: Profile of activity for other benchmarks

The fraction of active blocks in an interval is only an upper bound on the fraction of useful energy consumed by the cache. Not all active blocks in an interval need to be turned on from the beginning of the interval. We define the *active ratio* of an interval as the ratio of the sum of the energy consumed by each active block after it is first touched to the maximum energy consumed during the entire interval assuming all blocks active. As a first order approximation, we estimate the energy consumed in an interval as the product of the time-averaged number of active blocks

in the interval and the length of the interval. Figure 3 shows the average of the active ratios in a 32K cache over all 100K intervals for each of the benchmarks. While it would be preferable to turn off each block as soon as it is known to be no longer useful, the data of Figure 3 suggest that the low-hanging fruit in energy savings can be harvested simply by turning off all the blocks at periodic intervals, and turning them back on when desired. The static energy consumed in such a cache would be smaller than the maximum energy by a factor close to the average active ratio of the application for the given interval. This observation is the motivation for the proposal of a *quenched cache* described in the next section.

Benchmark	Average Active Ratio
compress	0.66
Gcc	0.50
Go	0.43
Ijpeg	0.36
Perl	0.46
vortex	0.58
Xlisp	0.49

Figure 3: Active ratio for 32K cache averaged over 100K instruction windows

3. The Quenched Cache

The cache has indeed proven itself to be a useful container of frequently reused information. However it does so by being conservative – information is kept in a cache block until the cache needs the block to store other information for which it does not have other space. Thus less useful, or even useless, information may continue their stay in the cache well past their period of maximum use. Static energy is consumed in keeping such information around past their useful life. This energy consumption can be minimized if power is supplied to a cache block only when it contains useful information. The precise identification of "dead" values in a cache is a difficult problem [2], however, and the hardware needed for the purpose could itself negate the possible power gains.

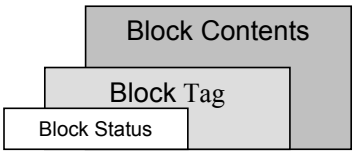
The crest of the graphs in Figures 1 and 2 identify the maximum number of blocks that are touched in each interval. At any given time in an interval only some of the blocks already touched will be truly useful. An alternative to sophisticated schemes to identify dead values in a cache would therefore be to simply turn off the entire cache and restart from scratch (cold start) at periodic intervals. The dead values would be purged, but so would the set of useful information. The effect of losing such information is an increase in the number of misses for the application. Interestingly, the percent increase in miss-rate is highest when the number of active

blocks is well below the total number of blocks – a period when there is enough computation available to both hide the latency and to amortize the cost of the miss.

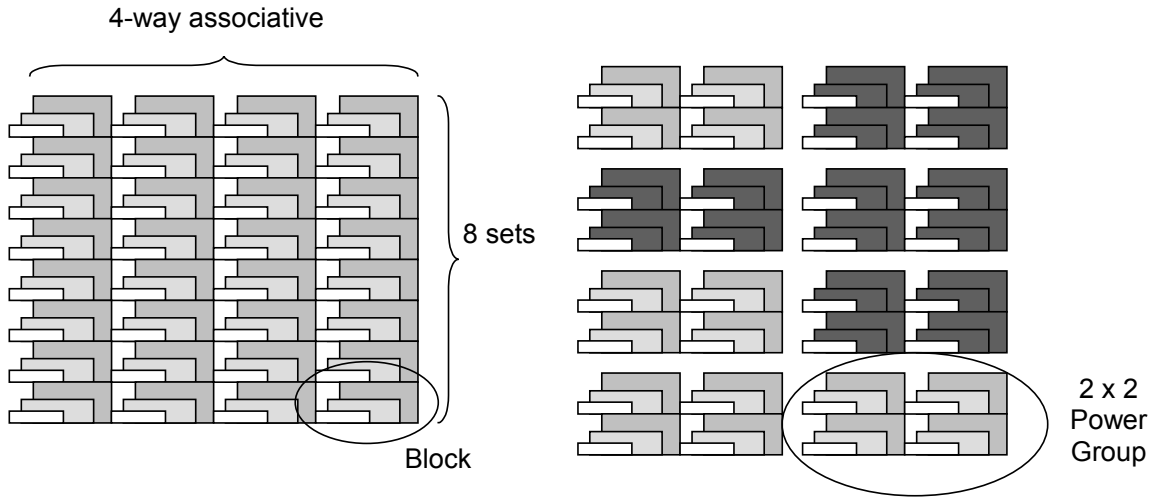
Figures 4(a) and (b) illustrate a conventional unquenched and quenched cache respectively. A quenched cache is composed of groups of cache entries, each having its own power control. For a cache containing ***s*** sets of entries, each with an associativity **a**, we define a **ps x pa** quenched cache as one in which each power group has **ps** sets, each with associativity **pa**, where **ps ≤ s**, **pa ≤ a**. Thus there are **s/ps * a/pa** power groups in the **ps x pa** quenched cache.

In the figure, both caches have 32 blocks arranged in 8 sets with associativity 4. The quenched cache has 4 power islands, each of which is arranged in a 2 x 2 format. Each power island can be turned on independent of the others. Often the block status bits and the address tag of a cache block are maintained in a separate array independent of the block contents. We propose that the status bits are kept in an array that is always kept powered. A special bit in this set is used to indicate whether or not the corresponding block is currently powered on. The only energy consumed when a power island is turned off is that needed to maintain the block status bits.

It is not necessary to be able to turn off each power island independent of the others. We simply turn off the entire cache after ensuring that all dirty cache blocks have written back their values to the next level of the memory hierarchy.



Information in a cache block



(a) Conceptual organization of a traditional cache

(b) Conceptual organization of a quenched cache
(Dark regions represent inactive groups)

Figure 4: The Quenched Cache

In the special case of an **`s x 1`** organization, each power group essentially forms a power column. Such a cache will start essentially as a direct-mapped cache and increase its associativity as needed until the entire cache is turned on or until the end of the quenching interval, whichever occurs first.

In the next section we will show results of simulations of quenched caches of various sizes, configurations, and quenching frequency.

4. Experimental Evaluation

To evaluate the performance characteristics of a quenched cache we used a simulator for the PowerPC instruction set, implemented as an out-of-order microarchitecture in the Dynamic Instruction Formatting (DIF) style [3]. In a DIF microarchitecture there are two pipelines, the *sequential* pipe, a traditional in-order pipeline getting its instructions from a traditional I-cache, and the *parallel* pipe, a wider-issue in-order pipeline getting its instructions from a specially formatted DIF cache. When a code segment is first encountered, it is executed on the sequential pipe and its dependencies are analyzed to arrive at an appropriate schedule. The scheduled code

segment is saved in the DIF cache from where it is extracted and executed when the same entry point is subsequently encountered. A full description of DIF appears in [3]. In this paper we study mainly the D-cache behavior, and for this purpose, we believe that the results indicated are representative of any out-of-order superscalar processor. The DIF microarchitecture however did provide us the opportunity to demonstrate the effectiveness of quenching on another cache structure, the DIF-cache which serves a somewhat different role compared to traditional I- and D-caches.

I-cache size (bytes)	4K			
I-cache miss penalty (cycles)	4			
DIF-cache (groups)	512			
Group Size (wide words)	6			
Branch Units	2			
Integer Units	2			
Load Units	2			
Store Units	1			
Floating Point Units	2			
DIF group size	36			
D-cache hit latency (cycles)	2			
D-cache miss penalty (cycles)	4			
D-cache line size (bytes)	64			
D-cache size (bytes)	4K	8K	16K	32K
D-cache sets	64	64	128	128
D-cache associativity	1	2	2	4

Figure 5: Parameters used in trace simulation

The parameters of the DIF implementation used for the study are shown in the table of Figure 5. These parameters represent a design that is a conservative balance between the supported instruction-level parallelism and clock frequency. We do not believe that the qualitative nature of the results will change for a high frequency implementation. The latencies, especially those for I-cache and D-cache misses will increase to 10 cycles or more in a high-frequency implementation, but we will show that increased misses is not a major contributor to the performance overhead of a quenched cache.

The input to the simulator was a set of instruction and memory reference traces obtained by executing some of the applications from the SPECint suite. Each of the benchmarks was run to completion on a data set chosen to ensure the generation of a reasonable sized trace (around 100 million instructions). The SPECint benchmarks, as a whole, may not be representative of characteristics in real user situations; however, the results of individual programs in the SPEC suite allow considered evaluation of the strengths and weaknesses of a design point.

Figure 6 shows the cumulative active ratio and relative miss rate of a quenched 32K cache compared to an unquenched cache for two benchmarks, *gcc* and *perl*. The cumulative active ratio at a given instruction is an indicator of the energy relative to an unquenched cache expended by the quenched cache from the beginning up to that instruction. Only the first 20 million instructions are shown. We can see that the cumulative active ratio dips and rises until it

smoothes out. By the end of the program, the cumulative active ratio asymptotically reaches 0.5 for *gcc* and 0.46 for *perl* as indicated in the table of Figure 3.

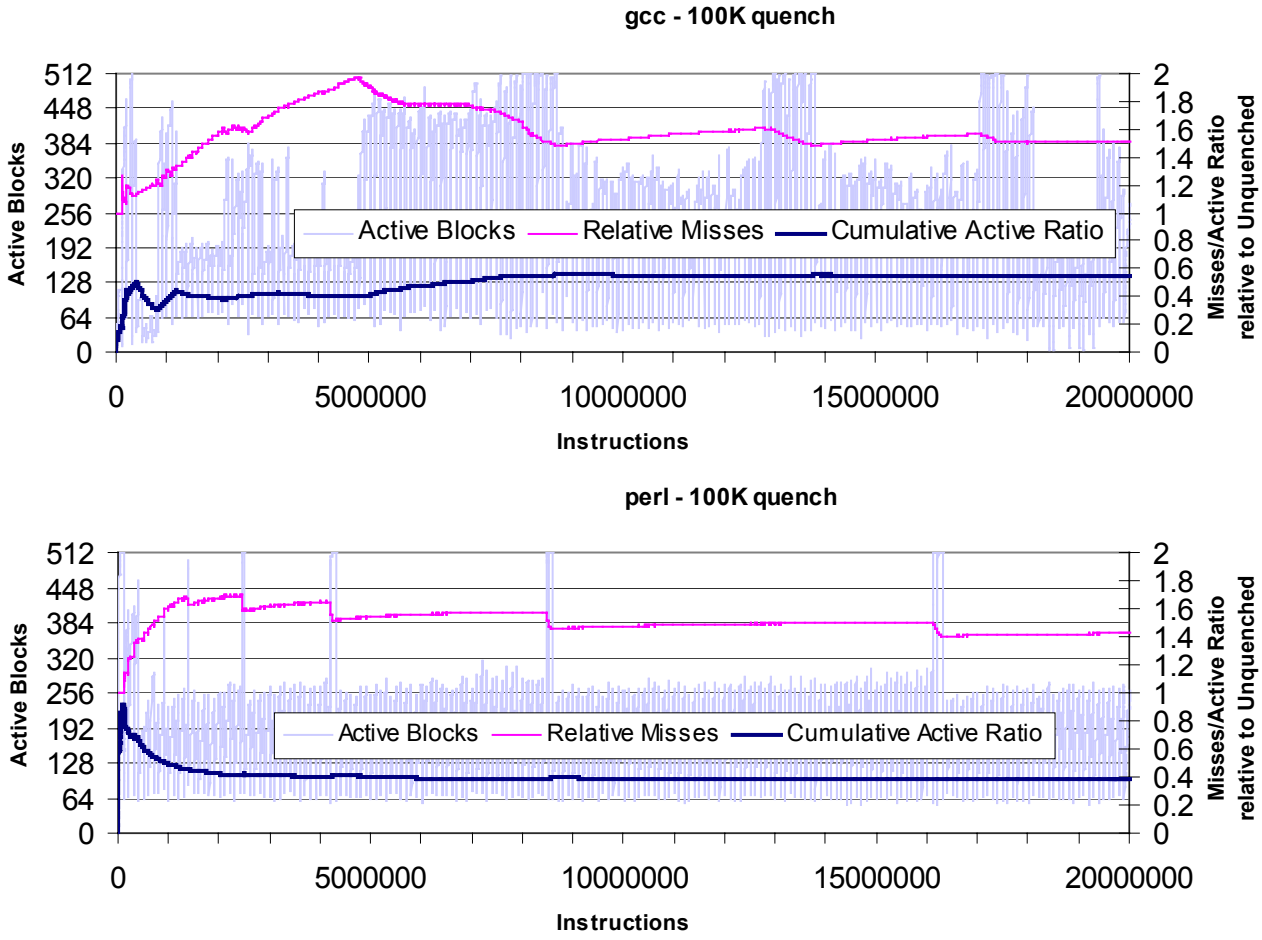


Figure 6: Miss rate and activity in quenched cache relative to an unquenched cache

The graphs also show the total number of misses at each point, expressed relative to the number of misses in an unquenched cache. This plot shows roughly a 50% increase for a quenched case for *gcc*, and a 40% increase for *perl*.

An interesting behavior is observed from these graphs – the relative number of misses appears to increase during periods of low activity and decrease during periods of high activity. This is directly in contrast to the behavior of the active ratio, suggesting that a greater fraction of useful information is lost because of quenching during periods of low activity. In order to explore this further, we created a scatter plot of the excess misses in a quenched cache against the inherent (warm) misses in the unquenched cache. These plots for an 8K cache, shown in Figure 7, do confirm that excess misses tend to be higher in regions where the unquenched cache had fewer

misses. An explanation for this is that, during periods of low activity, the processor is busy performing computation on a small working set, a large portion of which is useful and needs to be brought back in after quenching. On the other hand, during periods of high activity, there is more of a streaming data behavior with frequent replacement of cache lines – of all the blocks brought in during this period, very few will need to be brought back in after quenching.

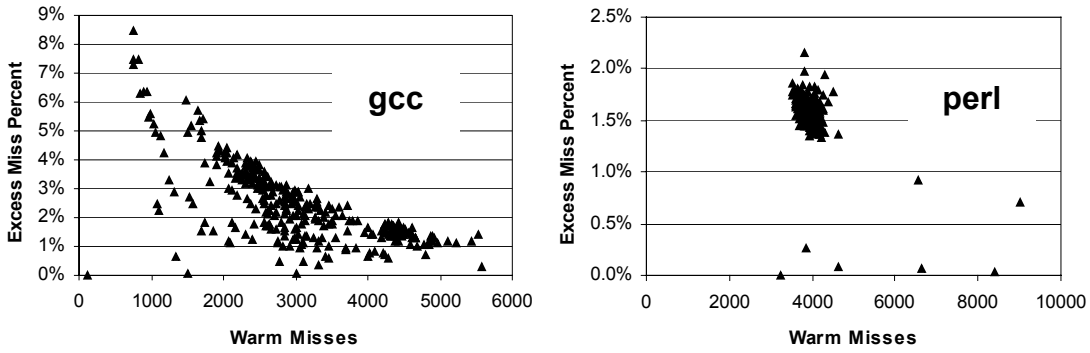


Figure 7: Excess misses due to quenching as a function of misses in unquenched cache

This is the first important property of quenching – quenching degrades performance minimally where the performance is already bad and hence limits its overall impact on performance.

Variation with Cache Size

For a given application, as the cache size increases, one can expect relatively fewer blocks in the cache to be utilized. Hence there is more to be gained from quenching as cache size increases. This is evident from the data in Figure 8. When the cache size becomes small compared to the application working set, both the unquenched and quenched caches have peak activity most of the time and hence there is no power benefit to be gained from quenching.

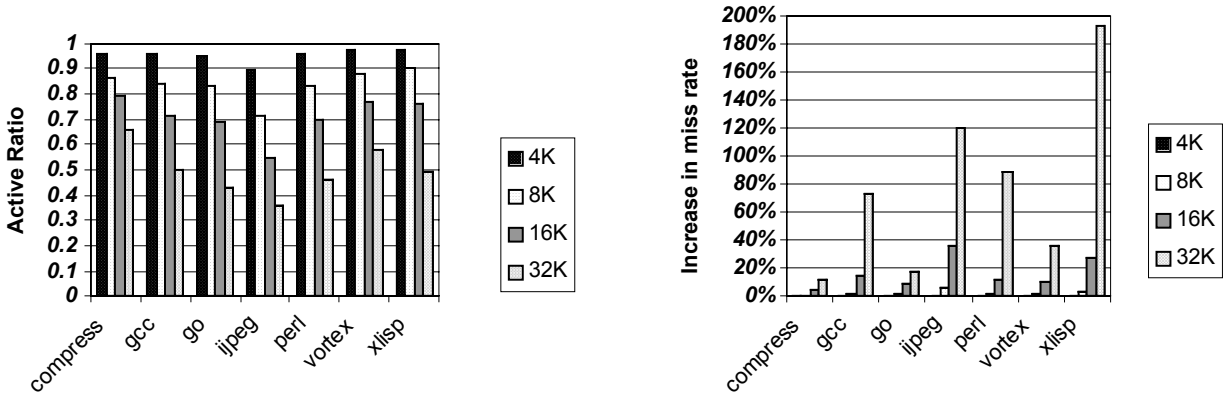


Figure 8: Active ratio and increased miss rates for caches of various sizes quenched at 100K interval

Note that as cache sizes get smaller, they are inherently consuming less power – the need to conserve power at 4K is much less than the need to conserve power at 32K when the cache inherently consumes 8 times the power.

This leads us to another important aspect of quenching – during periods of low activity, and when the cache is relatively large compared to the working set of the application, quenching conserves power without having to resort to sophisticated dynamic measures to resize and reconfigure the cache.

Figure 8 also plots the increase in miss rate for the different applications. It is clear that there is a dramatic increase in miss rate of a quenched cache compared to an unquenched cache as the size of the cache increases. However, this is less alarming than it looks because the miss rates in an unquenched cache are already quite low when the cache size is large, as seen in Figure 9.

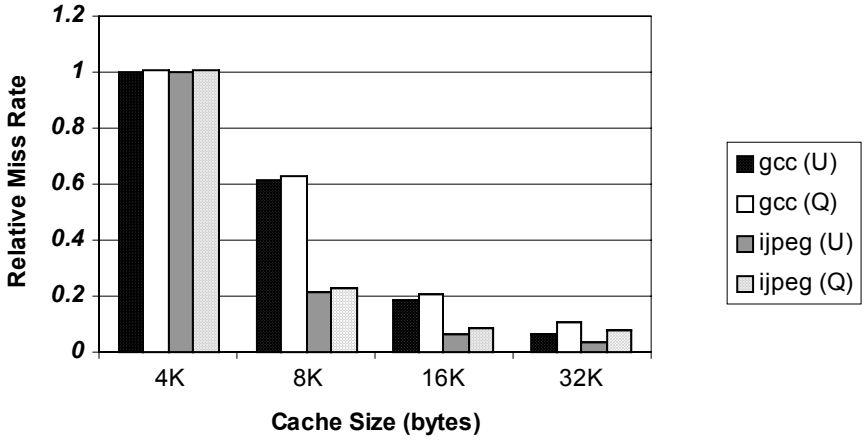


Figure 9: Miss rates for unquenched (U) and quenched (Q) caches relative to miss rate for unquenched 4K cache

Figure 10 plots the performance effect due to increased miss rates in a quenched cache. We see that the performance degradation due to increased miss rate is worst where the power savings are the highest, namely for large caches, but that even in this case, the degradation is limited to below 1.2% for the chosen set of design parameters.

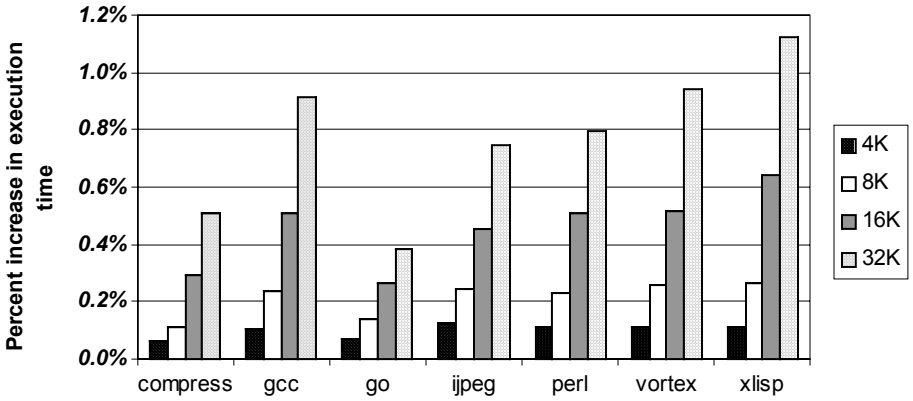


Figure 10: Degradation in performance of quenched cache due to miss rate increase

Overhead due to quenching

Increase in miss rate is only one of the causes for performance overhead due to quenching. There are two other sources of performance overhead, the overhead incurred during the quenching operation at the end of the quenching interval, and the overhead incurred while turning on each cache block on demand.

The quenching operation involves two phases, one of writing the dirty lines back to the next level, and the other of turning off the cache and waiting for transients to settle down. The time taken to write dirty lines back depends on the number of lines to be written back and the ratio of the cache linesize to the width of the bus used to write back.

The table in Figure 11 shows the average number of blocks that are turned on during a 100K interval and the average number of dirty lines at the end of the 100K interval. It is observed that the total number of dirty lines is often considerably less than the number of lines activated during the 100K interval.

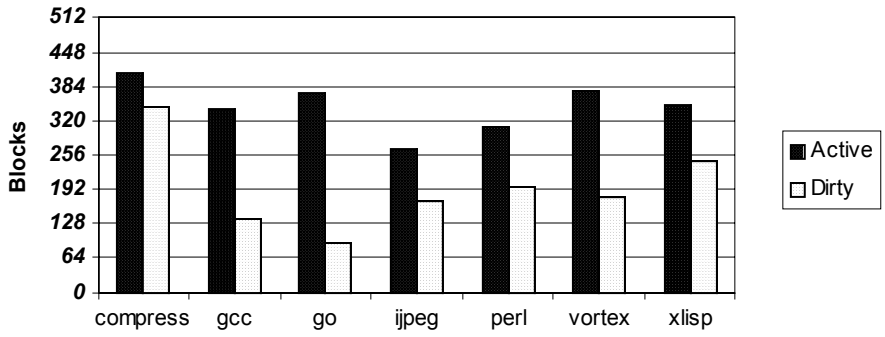


Figure 11: Average number of blocks in a 32K (512 block) cache which are (a) active (b) dirty at end of a 100K interval

Each writeback of a dirty line takes a number of cycles that depends on the length of the cache block and the width of the writeback bus. We assume here a 32 byte writeback bus, which implies a pipelined throughput of 2 cycles per writeback for a 64 byte line. We will assume an additional 100 cycle penalty for starting up the writeback process and for allowing the cache to settle down after quenching.

When a miss occurs, the group containing the block must be turned if it is not already on. Turning on a part of the cache causes an electrical disturbance that could take a few cycles to settle down. Some of these cycles can be hidden under the miss latency – the island could be turned on as soon as a miss is detected, rather than after the data arrives from the next level. The actual additional overhead depends on the implementation; in the absence of real data, this paper will assume an overhead of 10 cycles.

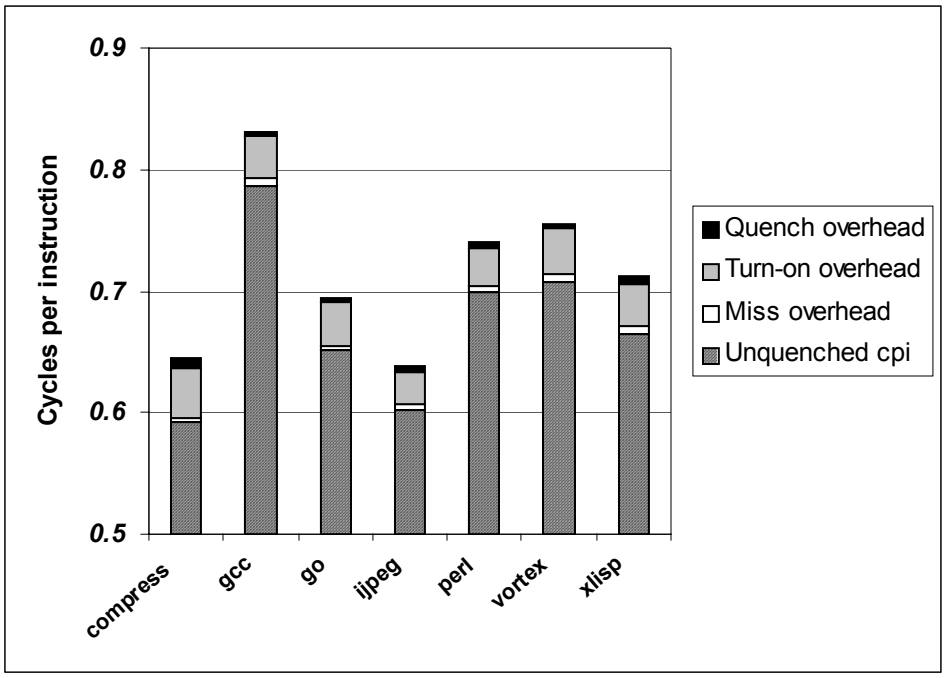


Figure 12: Overhead of quenching in a 32K cache (100K quench interval)

The result of these assumptions is the performance stack shown in Figure 12. The total performance overhead due to quenching is not too severe – typically less than 0.05 cpi. With smaller bus sizes, this overhead will increase. But it is expected that other techniques will be called upon to reduce this overhead when it becomes too high. For instance, it is possible to initiate the writeback process several hundred cycles ahead of the end of the interval.[1]

Interestingly, the major component of the performance overhead due to quenching is the effort to turn on the cache blocks. One way to reduce this overhead is to have more than one cache block in each power island. For example, 4 cache blocks are turned on simultaneously in a 4 x 1 power group. In practice, the resulting gain in performance must be evaluated against a possible loss in power savings. Figure 13 shows that the number of islands turned on, and hence the turn-on overhead, reduces quite dramatically as the size of the power island increases, because fewer islands need to be turned on during the quenching interval. However, this is offset by increased power consumption either because blocks are turned on earlier than they need be, or because additional blocks that are not needed also get turned on when an island is turned on. The choice of island size will probably depend on the area overhead due to partitioning the cache into power islands, and on the settling delay – a 2x1 or 4x1 island size appears to offer a good compromise between performance loss and power savings.

[1] This possibility was brought to the attention of the author by Viji Srinivasan.

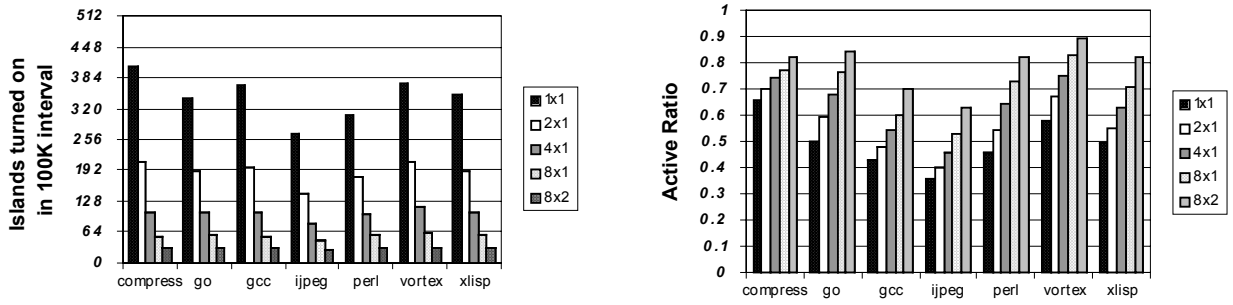


Figure 13: Turn-on overhead and active ratio for different sizes of power islands

Frequency of quenching

There remains an important question – that of the frequency of quenching. As the length of the quenching interval increases, more entries get activated, and dead entries remain on for a longer period of time. The graph in Figure 14 for one of the benchmarks, *ijpeg*, shows how the active ratio increases as the interval between quenches is increased. The graph also shows that savings in energy in small caches are more difficult to obtain – the quenching interval must be reduced in order to get such savings.

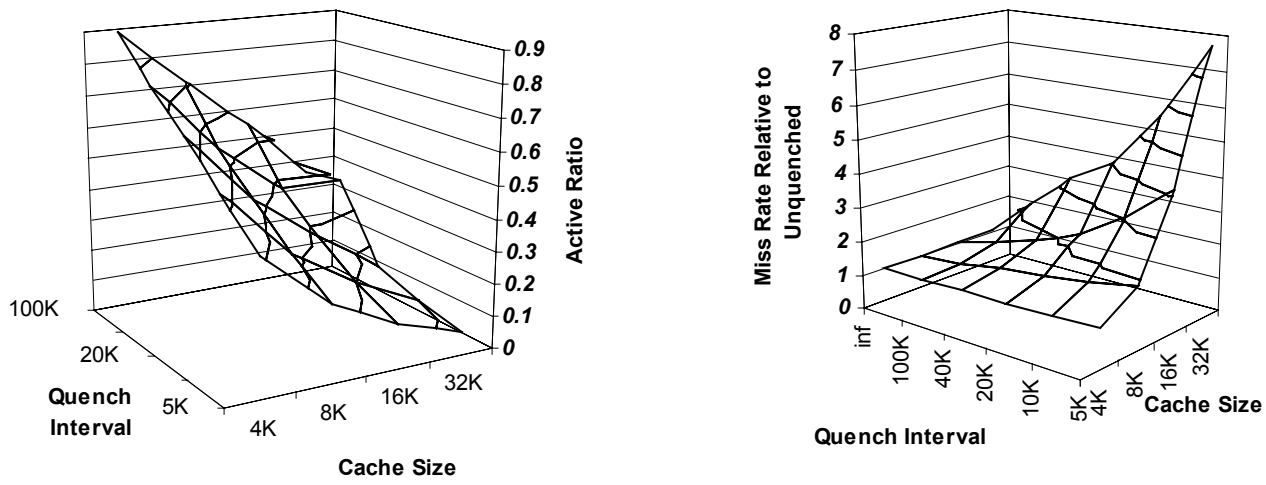


Figure 14: Variation of *ijpeg* miss rate and activity in a quenched cache as a function of quenching interval

Countering the savings in energy with smaller quenching intervals is the increase in miss rate as shown in second graph of Figure 14. Interestingly, it is possible to bring the active ratio down to 0.07 for a 32K cache using a quenching interval of 5000 instructions – the cost for this is a miss rate that is 7.8 times the miss rate of an unquenched cache. Perhaps 5000 instructions is unreasonable as a quenching interval – assuming 1 in 3 instructions is a load or a store, the

number of cache references within a 5000 instruction interval (about 1600) is considerably smaller than the number of 4-byte words in the cache (8000). A good tradeoff between power savings and quenching overhead is obtained when the quenching interval is a small multiple of the cache size in bytes. Thus a 50-80K quenching interval appears appropriate for a 32K cache, as a 6-10K interval does for a 4K cache. This is simply an empirical rule-of-thumb – further research could help pin down an appropriate quench rate as a function of cache size.

5. Extension to other type of caches

The concept of a quenched cache may be extended to any cache structure. Most cache structures exploit temporal and spatial locality, with applications often having active regions that are just a small fraction of the total cache. As demonstrated in the last section a quenched implementation dynamically discovers the active areas of such a cache. I-caches enjoy a further advantage in that the cache blocks do not have to be written back when quenched and hence incur lower performance penalty. There are situations, though, when the overhead of purging a potentially useful cache block is non-trivial. Examples of this include correlation-based branch predictors, which typically take a longer number of instructions in learning historical behavior, and trace caches or DIF caches which have to relearn how to schedule long traces of instructions.

In order to test the characteristics of one such cache, we simulated a quenched DIF cache. A DIF cache is like a trace cache, except that it also learns a schedule for the instructions in the trace. Thus there is the overhead of relearning the schedule whenever the contents of a useful DIF block are purged. The results of simulation are shown in Figure 15.

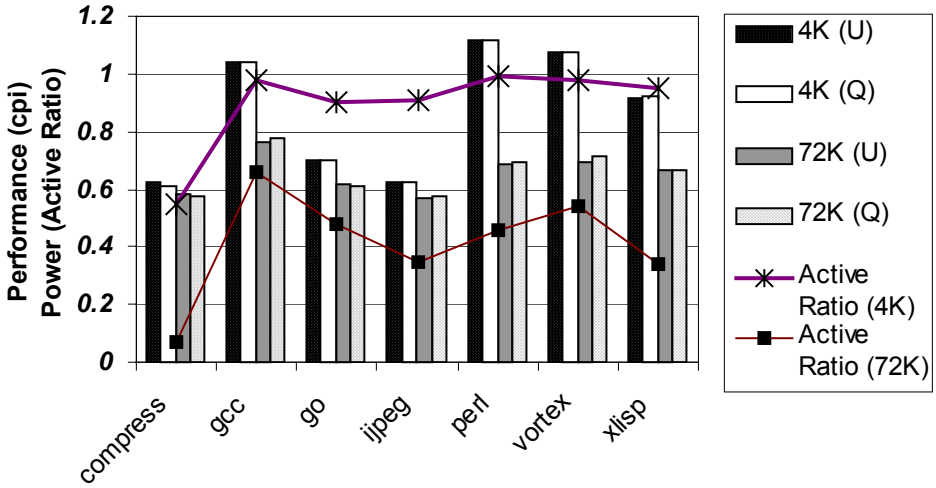


Figure 15: Performance and activity in a quenched DIF-cache

The power savings due to quenching are obvious for the larger 72K cache. (One can also see why a 72K cache would be useful to have in the first place – applications like *gcc*, *perl*, *vortex*, and *xlisp* enjoy a tremendous performance boost with the large DIF cache.) In the case of

compress, 93% of the energy is saved due to quenching because of its very small instruction footprint.

The histogram also shows the performance overhead due to quenching of the DIF cache. (We did not include the transient time in these plots). It is encouraging to note that the overhead of relearning potentially useful information is minimal in all cases, presumably because the relearning overhead is quickly amortised over a sufficient number of reuses. A more interesting observation is that in the case of both *compress* and *go*, there is actually an improvement in performance because of cache quenching. The reason for this is the following. Each scheduled trace of instructions in a DIF block typically includes several predicted conditional branches. When one of these conditional branches is mispredicted, the effective size of the group, and hence the exploited instruction-level parallelism, is smaller. This learned group could keep mispredicting, especially in applications whose instruction footprints are small, as in *compress* and *ijpeg*, providing little opportunity for capacity miss replacement. When purged, however, the incorrect trace starting at a given instruction is forgotten and the currently applicable longer trace is learned.

We believe that the observations made here apply to other types of learning caches, including branch prediction and value prediction structures. There are likely to be situations in some of these structures where performance benefit through unlearning wrong information by quenching offsets degradation due to loss of useful information.

6. Related work

Albonesi [4] proposed a cache design which changed the associativity of the cache according to application demands. Yang et al [5] proposed a more dynamic scheme that monitored miss rate in the cache in order to determine how to reorganize the cache. In an extension to the Albonesi scheme, they proposed reorganization, not only by changing the associativity of the cache, but also by changing the number of sets in the cache. In contrast to both these schemes, the quenched cache does not attempt to reconfigure the cache. As far as the processor is concerned, the cache is traditional in its design, with a fixed set size and fixed associativity. This eliminates all of the complexity associated with configurable schemes, especially that of changing indexing and of remapping blocks in a cache.

Various researchers, including Wood, Hill and Kessler [6] and Burger et al [7] have shown that a large percentage of blocks existing in a cache at any given time are never reused. Kaxiras, Hu and Martonosi have exploited this observation in their proposal [8], which has both on and off power controls for each cache block. Their proposal includes a hardware scheme for each cache block to determine when it should be turned off. The quenched cache differs from this in that it does not require a *local* (per-block) heuristic hardware mechanism to determine when a cache block should be turned off; rather it simply provides a *global* mechanism to turn off the entire cache. The somewhat reduced power savings due this global approach is offset in the quenched cache by the considerable simplicity of design – other than the provision of gated-V_{dd} [9] power islands, there is practically no difference in the design of a quenched cache from the design of conventional caches.

Flautner et al [10] have proposed a drowsy cache, which reduces static power consumption by putting cache blocks in a low-power state where they retain their information but cannot be accessed. The authors propose putting the entire cache in the drowsy mode at periodic intervals. In contrast to drowsy cache, a cache block in a quenched cache consumes no energy in the off state. In periods of very low activity, which are common in many computers, the quenched cache saves a lot more power than a drowsy cache. The quenched cache implementation is also simpler, not having to deal with an additional V_{dd} line to each cache block, and not having to be concerned about the added risk of losing information in the low-V_{dd} state. Our work addresses the performance issues involved with writing back and turning off the cache, and shows why they are not of particular concern. The quenched cache therefore trades off minimal performance overhead with considerably greater power savings, simplicity, and robustness.

7. Conclusion

Modern microprocessors employ large caches of various types that consume a large fraction of the energy on a chip. Caches help improve performance of processors, but they tend to be underutilized and hence inefficient in their power consumption. It is well known that most blocks in traditional caches often contain information that is not needed immediately. This paper confirms this through profiles of activity in typical D-caches running a variety of applications.

The cache activity profiles point to significant improvement in power-performance of caches, simply by purging all information in the cache periodically. This quenching action does incur the overhead of relearning some of the purged information. However, only a small fraction of the cache typically needs to be relearned, and the cost of relearning is offset by the energy saved from turning off unneeded cache blocks.

The quenched cache differs from a traditional cache only in that the blocks are grouped into power islands, each of which is independently turned on when the first miss to one of its blocks occurs. The size of the power island determines the area overhead and potential energy savings – a power island with 1 or 2 blocks is ideal from the energy savings point of view, but incurs more area overhead compared with a power island consisting of many more blocks.

The degree of power savings is also dependent on the length of interval between quenches. When the quenching interval is small the power savings is greater because inactive blocks remain turned on for a shorter period of time. However, short quenching intervals are associated with greater relearning and turn-on/turn-off overhead. The paper quantifies this overhead and empirically determines that a balance between the increased overhead and energy savings in a D-cache is obtained with a quenching interval (in instructions) that is roughly 2 to 3 times the cache size (in bytes).

Trace simulations indicate that the performance overhead of quenching is minimal for SPEC-type applications using sizes of caches that are common today. In reality, larger caches are becoming the norm – they are needed to boost performance on the occasional but important workload that needs them. Quenching results in greater energy savings on the more frequent applications that do not need large caches, and it does so without complicated schemes either to

detect application requirements or to reconfigure the cache. Moreover, the energy savings in a quenched cache automatically increases as cache activity reduces. It is important to note that, because of its similarity in implementation to a traditional cache, the performance and power characteristics of a quenched cache should be identical to those of the traditional cache when quenching is inhibited.

The paper demonstrates the usefulness of quenching in cache structures beyond the traditional I- and D-caches. Its effectiveness was shown for a special structure called the DIF cache that contributes to improved performance in wide-issue processors by caching scheduled traces of instructions. As with D-caches, large DIF caches help in improving performance of some programs with large instruction footprints, and quenching helps keep the power consumption minimal in the many other situations where the footprint is small.

One interesting aspect of quenching is its potential to actually improve performance in certain situations. A cache is a store of learned information, used as a hint of the future behavior of a program. Occasionally, the cache collects wrong information or information that is outdated, and the result is a performance-reducing hiccup every time it is encountered. Schemes to unlearn such information are seldom perfect or complete. The paper illustrates situations where unlearning simply by quenching actually improves performance. Quenching acts as a renewal mechanism, and as in nature, this renewal provides a new opportunity to learn correctly.

This paper did not address circuit design aspects – it is hoped that the results here will provide an impetus for the development of circuit techniques that lead to implementations of quenched caches having minimal area overhead in comparison to traditional caches. With the incorporation of ever-larger number of components on a chip, increasing importance is being given to the design of power-efficient microprocessors. Different types of cache structures dominate the area of modern microprocessor chips – redesigning all these caches as quenched caches can help tremendously in improving the power-efficiency of these chips.

Acknowledgement

The author wishes to thank Viji Srinivasan and Dan Prener for useful discussions and comments on the manuscript.

References

[1] J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, "POWER4 system Microarchitecture," IBM Journal of Research and Development, vol. 46, no. 1, pp. 5-26, 2002.

[2] A. N. Lai, C. Fide, and B. Falsafi, "Dead-Block Prediction and Dead-Block Correclating Prefetchers," Proceedings of the 28th International Symposium on Computer Architecture, 2001.

[3] R. Nair and M. Hopkins, "Exploiting Instruction Level Parallelism in Processors by Caching Scheduled Groups," Proceedings of the 24th International Symposium on Computer Architecture, pp. 13-25, June 1997.

[4] D. H. Albonesi, "Selective Cache Ways: On-Demand Cache Resource allocation," Proceedings of the 32nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 32), pp. 248-259, Nov. 1999.

[5] S.-H. Yang, M. D. Powell, B. Falsafi, and T. N. Vijaykumar, "Exploiting Choice in Resizable Cache Design to Optimize Deep-Submicron Processor Energy-Delay," Proceedings of the 8th International Symposium on High Performance Computer Architecture (HPCA 8), 2002.

[6] D. A. Wood, M. D. Hill, and R. E. Kessler, "A Model for Estimating Trace Sample Miss Ratios," ACM SIGMETRICS, pp. 79-89, June 1991.

[7] D. Burger, J. Goodman, and A. Kagi, "The Declining Effectiveness of Dynamic Caching for General-Purpose Microprocessors," Tech. Report TR-1216, University of Wisconsin, Madison, Computer Sciences Department.

[8] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage power," Proceedings of the 28th International Symposium on Computer Architecture, pp. 240-251, 2001.

[9] M. D. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated Vdd: A Circuit Technique to Reduce Leakage in Cache Memories," Proceedings of the 2000 International Symposium on Low-Power Electronics and Design (ISPLED), pp. 90-95, July 2000.

[10] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," Proceedings of the 29th International Symposium on Computer Architecture, 2002.