Journal of Instruction-Level Parallelism 2 (2000) 1-6 Submitted 2/00; published 5/00

Selective Cache Ways: On-Demand Cache Resource Allocation

David H. Albonesi ALBONESI@ECE.ROCHESTER.EDU
Dept. of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627-0231 USA

Abstract

Increasing levels of microprocessor power dissipation call for new approaches at the architectural level that save energy by better matching of on-chip resources to application requirements. *Selective cache ways* provides the ability to disable a subset of the ways in a set associative cache during periods of modest cache activity, while the full cache may remain operational for more cache-intensive periods. Because this approach leverages the subarray partitioning that is already present for performance reasons, only minor changes to a conventional cache are required, and therefore, full-speed cache operation can be maintained. Furthermore, the tradeoff between performance and energy is flexible, and can be dynamically tailored to meet changing application and machine environmental conditions. We show that trading off a small performance degradation for energy savings can produce a significant reduction in cache energy dissipation using this approach.

1. Introduction

Continuing advances in semiconductor technology have fueled dramatic performance gains for general-purpose microprocessors. These improvements are due both to increasing clock rates as well as to increased support for exploiting instruction-level parallelism and memory locality using the additional transistors available in each process generation. However, a negative byproduct has been a significant increase in power dissipation, due to the fact that the dynamic power (the dominant contributor to power consumption in most current high-speed CMOS circuits) is proportional to both clock frequency and to the switching capacitance (which increases as more functionality is added). Thus, despite herculean attempts to reduce voltages and design lower power circuits, power dissipation levels have steadily increased with each new microprocessor generation.

Consider, for example, the characteristics of four generations of Alpha microprocessors as shown in Table 1. In the six years separating the first (21064) and most recent (21264) generations, power dissipation has doubled from 30W to 60W, despite the fact that the *energy per transition* (calculated in the last column by dividing the power dissipation by the clock frequency [8]) has declined during this period. This example shows how rapid increases in both clock frequency and chip functionality (and thus switching capacitance) are outpacing circuit design attempts to keep power dissipation to reasonable levels. Power projections for the next implementation of the Alpha line (21364 [7]) indicate that this trend is expected to continue. Similarly, while the UltraSparc I microprocessor [9] dissipated 28W at 167MHz, the UltraSparc III design [10] is estimated to dissipate 70W at 600MHz, despite the fact that the energy per transition is comparable in both generations.

Table 1: Characteristics of four generations of Alpha microprocessors. Some of the data reflects projections made in the referenced papers and therefore may differ from actual characteristics at product shipment.

Product	Characteristics					
	Year	Trans (M)	Voltage (V)	Clock (MHz)	Power (W)	Pwr/Clk (W/MHz)
21064 [1, 2]	1992	1.68	3.3	200	30	0.15
21164 [3, 4]	1995	9.3	3.3	300	50	0.17
21264 [5, 6]	1998	15.2	2.2	575	60	0.10
21364 [7]	2000-01	∼100	1.5	∼1000	∼100	∼0.10

Because of the inability of circuit-level techniques to singlehandedly keep power dissipation to reasonable levels, there has been increasing interest in architectural approaches that reduce the switching capacitive power component. These techniques improve *energy* dissipation by reducing the number of signal transitions within the microprocessor for a given workload. One focus area for such work has been the on-chip cache hierarchy, due to the fact that an increasing percentage of die area is being devoted to cache transistors, and that on-chip L1 caches alone can comprise over 40% of the total die power budget [11].

Several approaches have been implemented in commercial microprocessors to reduce cache energy dissipation by limiting switching activity. The SA-110 embedded microprocessor [11] uses 32-way associative 16KB L1 I and Dcaches, each of which is divided into 16 fully associative subarrays. With this scheme, only one-eighth of the cache is enabled for each access which considerably reduces dynamic power. With a 160MHz target clock frequency, the SA-110 designers were able to maintain a one cycle cache latency with this degree of associativity. In a high performance microprocessor with L1 caches several times larger in size and clock rates in the GHz range, such a solution would likely increase L1 cache latencies, and significantly impact the performance of many applications.

A similar approach is used to reduce power consumption in the 21164 microprocessor's 96KB, 3-way set associative L2 cache, whose data section is split into 24 banks. The tag lookup and data access are performed in series rather than in parallel as with a conventional cache. This allows for predecoding and selection of 6 of the 24 data banks in parallel with tag access, and final access of only the 2 banks associated with the selected way[1]. Because only a small fraction of the total L2 cache is enabled on each access, the dynamic power savings is considerable, estimated at 10W [4]. However, when applied to an L1 cache, the serial tag-data access of this technique would significantly increase cache latency as with the SA-110 approach.

Several other approaches have been proposed for reducing the switching activity in on-chip caches. The filter cache [12] attempts to reduce energy dissipation by placing a small (and therefore low-energy) cache in front of the L1 cache. If many L1 cache requests can be serviced by the filter cache, then L1 cache switching activity can be greatly reduced thereby

1. In this paper, we use the term *set* to refer to the cache block(s) pointed to by the index part of the address, and the term *way* to refer to one of the n sections in an n-way set associative cache.

saving energy; indeed, a 65% energy savings is demonstrated with 256 byte instruction and data filter caches backed by a 32KB unified L1 cache. However, the cost is a 29% performance degradation relative to a conventional design due to an increase in the access time on a filter cache miss. The L-Cache [13] similarly reduces switching activity by holding loop-nested basic blocks designated by the compiler and providing these to the pipeline in place of the larger L1 Icache. Because the compiler selectively chooses which basic blocks to include in the L-cache, a much smaller performance overhead (1-5%) is incurred compared with the filter cache. However, the benefits of the L-Cache are limited to large basic blocks within loops that are executed many times; therefore, less than a 10% savings in cache energy dissipation is realized for the four SPECint95 benchmarks that were evaluated.

Each of these techniques significantly alters the cache design in order to improve energy efficiency. An alternative approach is to start with a high performance set associative cache design and to exploit the subarray partitioning that is usually present for performance reasons. With this partitioning in place, only minor hardware modifications are necessary to provide the ability to enable all of the cache ways when required to achieve high performance, but to enable only a subset of the ways when cache demands are more modest. In the latter case, the remainder of the ways can be placed in a quiescient state, thereby reducing cache switching activity. This approach, which we call *selective cache ways*, provides the highest performance solution in an *on-demand* fashion, and opportunistically trades off a small performance penalty for significant energy savings when appropriate. Selective cache ways exploits the fact that cache requirements may vary considerably between applications, as well as during the execution of an individual application. With appropriate software support for determining sufficient L1 cache requirements and enabling the appropriate number of cache ways, a significant savings in cache energy can be realized without degrading the performance of more cache-intensive programs.

The rest of this paper further explores this concept and is organized as follows. In the next section, we examine the subarray partitioning that is necessary to achieve optimal L1 cache performance, and how this partitioning can be exploited to tailor the cache organization to application requirements. We then discuss in Section 3 the selective cache ways approach, and in Section 4, we evaluate the energy savings obtained with different levels of tolerable performance degradation. Software support for selective cache ways is discussed in Section 5, and we conclude and present future work in Section 6.

2. Partitioning Caches for Performance

Selective cache ways exploits the fact that large on-chip caches are often partitioned into multiple subarrays to reduce the long word and/or bitline delays of a single large array. This is especially true for the data array and for set associative caches for which the wordlines can be exceedingly long. For example, each of the two 512KB data banks of the 1MB 4-way set associative L1 Dcache of the HP PA-8500 microprocessor is partitioned into four 128KB subarrays [14]. Our analysis using the Cacti cache cycle time model [15] for smaller caches produces similar results. Table 2 shows the N parameters from Cacti that produce the fastest cycle time for various caches. The parameters *Ndwl* and *Ntwl* refer to the number of times the wordlines are segmented for the data and tag arrays, respectively, while *Ndbl* and *Ntbl* are the corresponding parameters for the bitlines [16]. The parameters *Nspd* and

Table 2: Optimal N parameters for various caches. The block size is 32 bytes.

Cache Org		Optimal N Parameters					
Size	Assoc	Ndwl	Ndbl	Nspd	Ntwl	Ntbl	Ntspd
16KB	1	2	4	1	1	2	2
	2	4	2	1	1	2	2
	4	4	2	1	1	2	1
32KB	1	1	8	1	1	2	4
	2	8	1	1	1	2	2
	4	4	2	1	1	2	1
64KB	1	1	8	1	1	2	4
	2	4	2	1	1	2	2
	4	4	2	1	1	2	1

Ntspd refer to the number of sets that are mapped to the same wordline for the data and tag arrays, respectively [15]. Figure 1 shows an example of a 4-way set associative cache with $Ndwl = 4$ and $Ntbl = 2$ and all other N parameters equal to one. The data array is partitioned into four subarrays, each with its own decoder and one-quarter the sense amps of a single data array. Here, each wordline is roughly one-quarter the length of that in a single array. Two tag subarrays are formed by segmenting the bitlines, resulting in a halving of the decoder width but a doubling of the number of sense amps relative to a single tag array. Only one of the two sets of tag sense amps associated with the same column is activated during each access [16].

Referring to Table 2, we see that in all cases, multiple subarrays are required for optimal performance. In particular, we note that *Ndwl* is at least as large as the associativity in all cases. A key insight for set associative caches is that when *Ndwl* is greater than or equal to the associativity, the resulting organization is already partitioned such that a *subset* of the data ways can operate without changing the basic cache organization or the index-tag address partitioning. Only a small amount of additional gating logic must be added. Thus, the resulting organization can run at the speed of a conventional cache, yet provide the ability to selectively turn off ways when appropriate. However, some overhead is incurred when changing the number of enabled ways, including ensuring that modified data is handled correctly. In the next section, we provide further details on this and other aspects of the selective cache ways approach.

3. Selective Cache Ways

Selective cache ways combines the following hardware and software elements:

- A partitioning of the data and possibly the tag arrays into one or more subarrays for each cache way;
- Decision logic and gating hardware for disabling the operation of particular ways;

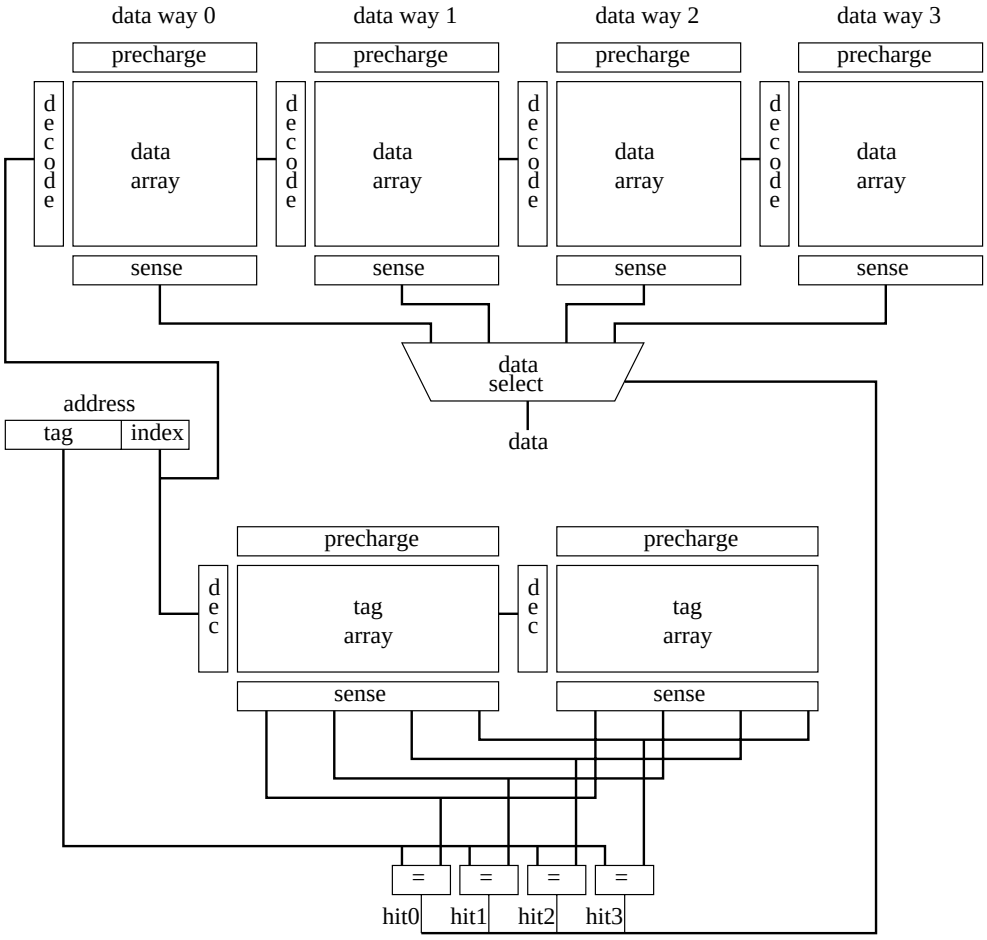

Figure 1: A 4-way set associative cache with $Ndwl = 4$, $Ntbl = 2$ and all other N parameters equal to one.

- A software-visible register, called the *Cache Way Select Register (CWSR)*, that signals the hardware to enable/disable particular ways;
- Special instructions, called WRCWSR and RDCWSR, for writing and reading, respectively, the CWSR;
- Software support for analyzing application cache requirements, enabling cache ways appropriately, and saving the CWSR as part of the process state;
- Support for sharing data in disabled ways and maintaining its proper coherency state.

In the following subsections, we describe the hardware aspects of selective cache ways; software support for this approach is discussed in Section 5.

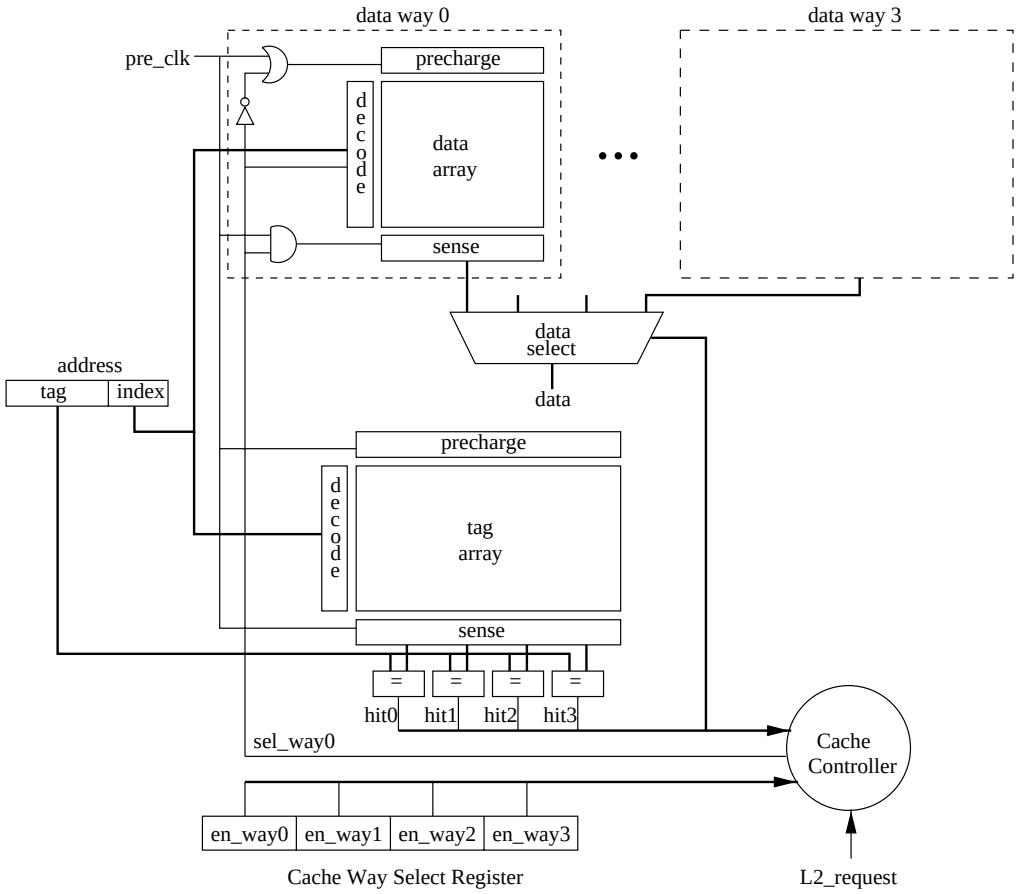

Figure 2: A 4-way set associative cache using selective cache ways. The details for data ways 1-3 are identical to way 0 but are not shown.

3.1 Hardware Organization

Figure 2 is an overall diagram of a 4-way set associative cache using selective cache ways. The wordlines of the data array are segmented four times per the Cacti results of Table 2, creating four separate data way elements. The bitlines of each data way may be segmented as well. Note that the tag section of the cache (which includes the status bits) is identical to that of a conventional cache. Cacti-based timing estimates indicate that for the cache organizations we have studied, segmenting the tag wordlines will result in a significant cache cycle time degradation relative to the optimal tag N parameters of Table 2. Thus, our approach only saves energy in the data section of the caches we have studied. However, for the benchmarks that we investigated, we found that the the data section comprises roughly 90% of the total energy dissipation for these organizations.

The Cache Way Select Register (CWSR) is written and read via the WRCWSR and RDCWSR instructions and contains four bits in this example (*en_way0*, *en_way1*, *en_way2*, and *en_way3*), each of which signals the Cache Controller to enable a particular way, and

thereby allow it to operate. If a particular way enable bit is zero, the *sel_way* signal for that way is also zero (except in limited circumstances that are discussed in Section 3.3). Therefore, the data way is not precharged, no word lines are selected, and the sense amps are prevented from firing. Thus, with this scheme, no data is selected from a disabled way and its data array dissipates essentially no dynamic power. Note that the replacement decision logic within the Cache Controller must also ensure that no data is allocated to a disabled way.

The degree to which ways are disabled is a function of two factors. The first is the relative energy dissipations of different memory hierarchy levels and how each is affected by disabling ways. The second is the amount of performance degradation that can be tolerated, which is discussed in the next section.

3.2 Performance Degradation Threshold

The *Performance Degradation Threshold (PDT)* signifies how much performance degradation relative to a cache with all ways enabled is allowable. Thus, if the PDT is 2% for a given period of execution, and performance is projected to degrade by 1% with three ways of a 4-way cache enabled, and 4% with two ways enabled, then three ways are enabled, so long as the total energy is less than that with four ways enabled. Thus, the minimum energy dissipation is not necessarily that with all but one way disabled, as the increase in L2 cache energy dissipation due to additional L1 misses may override the decrease in L1 energy obtained with disabling ways.

One advantage of selective cache ways over static solutions to energy reduction is that the tradeoff between performance and energy dissipation can be made variable via different PDT values. Thus, a different tradeoff can be made among machines that use the same microprocessor, and between different applications running on the same machine. Different PDT values can even be used for different instantiations of the same application on the same machine. For example, a server application may be invoked to run in a "low energy cache mode" during low-activity periods, and run in a "high performance cache mode" with more cache ways enabled during high-load periods. The operating system or a continuous profiling and optimization system [17, 18] could effectively control the PDT by changing the number of ways enabled under different loading conditions. In Section 4, we evaluate the energy savings that is achievable with different PDT values. In the next subsection, we address the issue of how to properly handle access to data stored in a disabled way.

3.3 Maintaining Data Accessibility

When an L1 Dcache way is disabled, modified cache blocks must be made accessible to the same process and to sharing processes (on the same processor or other processors) and to the I/O subsystem. Also, data coherency must be maintained in case the way is reenabled. In this section, we discuss two approaches for providing this capability, and discuss the performance overhead and energy dissipation of each solution.

3.3.1 Flushing Cache Ways

The most straightforward approach is to flush all blocks from the disabled cache ways before they are actually disabled. This may be performed via either software or dedicated

hardware. In a writethrough cache, flushing requires setting all blocks to the invalid state, while in a writeback cache, modified cache blocks must also be written to the L2 cache.

Some processors provide architectural support for cache flushing. In the Intel Pentium Pro processor, the WBINVD instruction causes modified blocks in the L1 Dcache and the L2 cache to be flushed to memory and all status bits set to Invalid. On the Intel TFLOPS supercomputer, a WBINVD writes back data at a rate of 250 MB/sec over the 66MHz 64-bit memory bus [19]. Assuming a 64KB L1 Dcache with 50% modified blocks and three ways to flush, a cache flush at this rate would take about 100 μs. However, this assumes that all writebacks must be sent to main memory. Because we only need to flush the L1 Dcache, writebacks may hit to a block in a non-shared state in the L2 cache, which would permit the writes to be done without requiring a memory bus transaction. If this is always the case, and we assume that the L1-L2 writeback bandwidth is a factor of 10 higher than the Pentium Pro memory bus, then a flush would require about 10 μs. If a flush overhead of less than 1% is desirable, then the number of enabled ways can be changed no faster than every 1 ms on average, a high degree of granularity relative to the rate at which changes in cache requirements may occur within an application [20]. Furthermore, Agarwal et al. [21] have shown that cache flushing in a multiprogramming environment can significantly impact cache miss rate, and therefore impose a significant performance penalty. This penalty would be imposed regardless of the write policy. Finally, considerable energy is expended during a cache flush, mitigating some of the benefit of disabling ways. For these reasons, we explore an alternative approach in the next subsection.

3.3.2 Limited Cache Way Accessibility

An alternative to flushing the cache is to make all ways accessible at all times for coherency requests from the L2 cache. In this scheme, the L2 cache behaves as if the entire L1 Dcache is enabled, initiating L1 Dcache coherence transactions when appropriate. When the L1 Dcache Cache Controller receives such coherency requests, it performs operations on the tag and status array and the data array as in a conventional cache. Thus, coherency transactions are carried out for all ways, regardless of the state of the CWSR, and these infrequently occurring operations expend no more energy than in a conventional cache.

With a slight modification to the cache datapaths, we can also ensure that a CPU request for a block that resides in a disabled way in its own L1 Dcache is handled properly. This can be achieved with additional states in the Cache Controller for acting on a hit to a disabled way, and with the addition of a multiplexor connecting the writeback datapath to the fill datapath as shown in Figure 3. This additional hardware allows the Cache Controller to read the block from the disabled way and send it to an enabled way via the fill datapath, to set the status bits to the appropriate coherence state, and to set the status of the block in the disabled way to Invalid. Depending on the size of a cache block and the datapath widths, it may be necessary to buffer part of the block during this transfer operation. If the CPU request was a load, this operation can be partially overlapped with sending the desired word to the CPU.

Although this approach allows for rapid changing of the enabled ways (as no flushing is required), a performance penalty on the order of 5-10 cycles (assuming the datapath widths are no less than one-quarter the width of a cache block) is incurred for each transfer

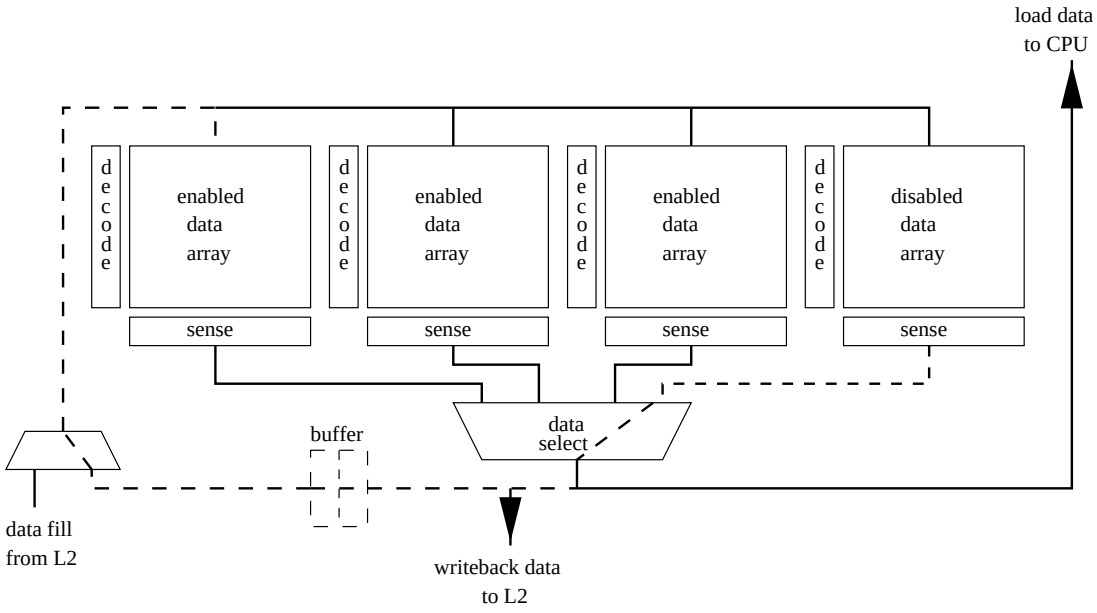

Figure 3: A 4-way set associative cache using selective cache ways showing the path for transferring data between a disabled and an enabled way.

of data from a disabled to an enabled way. In addition, a significant amount of energy is dissipated in transferring the cache block, offsetting the energy savings of disabling ways if these operations frequently occur. This may be the case in programs in which the CWSR is loaded many times during execution. In this case, the algorithm for changing the enabling of ways must consider the program boundaries within which variables and data structures are referenced. These operations occur rarely if the CWSR is loaded at the granularity of an individual process (*i.e.*, at context switch time), which we assume in the performance evaluation in Section 4.

3.4 Banking and Selective Cache Ways

Banking is a common method to reduce power in memories such as caches. With this approach, the subarrays are organized such that only one is selected for each access using a subset of the address bits while the others are gated off. This can be achieved for set associative caches by including all ways in each subarray.

The use of banked caches does not preclude the use of selective cache ways. Rather, the two can be combined to affect an even greater reduction in power consumption than can be achieved with either technique alone. In a banked cache each wordline is organized as a *global wordline* driven to the center of each subarray from which the *local wordline* is driven to the RAM cells in each subarray. In a banked cache, only one of the subarrays has its local wordlines enabled. With a small amount of additional gating logic, selective cache ways can be implemented by providing the option of sending the local wordline to either one or the other half of the subarray. Thus, a four way set associative cache may be configured

as only a two way cache. If additional wordline switches are placed between ways, support for three or one way may also be provided. However, this latter solution requires extra area and may unduly compromise access time. Because of the limitations of our timing and power models, we only evaluate the use of selective cache ways in non-banked caches.

4. Evaluation of Selective Cache Ways

4.1 Methodology

Our evaluation methodology combines detailed processor simulation for performance analysis and for gathering event counts, and analytical modeling for estimating the energy dissipation of both conventional caches and caches employing selective cache ways.

We use the SimpleScalar toolset [22] to model an out-of-order speculative processor with a two-level cache hierarchy. The simulation parameters, listed in Table 3, roughly

Table 3: Simulated system parameters.

Parameter	Value
fetch width	8 instrs
fetch queue	16 instrs
fetch speed	2X
decode width	8 instrs
branch pred	combined, 4K 2-bit chooser, 4K-entry bimodal, 12-bit, 4K-entry global, 2-cycle penalty
BTB	8192 entries, 4-way set assoc
return address stack	64 entries
RUU size	64 entries
LSQ size	8 or 16 entries
issue width	4
integer ALUs	2
integer mult/div	2
flt. pt. ALUs	2
flt. pt. mult/div	2
commit width	4
L1 Icache	64KB, 4-way set assoc, 32B block, random, 1 cycle latency
L1 Dcache	32 or 64KB, 2 or 4-way set assoc, 2 ports, 32B block, random, 1 cycle latency
L2 cache	512KB, 1MB, or 2MB, 4-way set assoc, 32B block, LRU, 15 cycle latency, 16 partitions
main memory	16B bus width, 75 cycle initial latency, 2 cycles thereafter

correspond to those in a current high-end microprocessor such as the HP PA-8000 [23] and Alpha 21264 [6]. We explore the performance of four different L1 Dcaches, with and without selective cache ways: 32KB and 64KB 2-way and 4-way set associative caches. All four L1 Dcache configurations are simulated as dual-ported caches, but are assumed to be single-ported caches which are double-pumped as in the Alpha 21264. The data array of the L2 cache is implemented as 16 partitions, only one of which is selected for each access, similar to the Alpha 21164 on-chip L2 cache [3]. Because performance is not as sensitive to L2 cache latency as it is to L1 Dcache latency, this approach was used for the L2 cache instead of selective cache ways. Note also that we vary the Load-Store Queue (LSQ) size in order to investigate the impact of different levels of memory operation concurrency on the performance and energy dissipation of selective cache ways.

We estimate L1 Dcache and L2 cache energy dissipations using a modified version of the analytical model of Kamble and Ghose [24]. This model takes as inputs technology and layout parameters and counts of various cache events and calculates the energy dissipated in each part of the cache in a similar manner as Cacti calculates access time. The event counts, in addition to performance results, are gathered from SimpleScalar simulations (each 400 million instructions long) of eight benchmarks which vary in their cache requirements: the SPEC95 benchmarks *compress*, *ijpeg*, *li*, *turb3d*, *mgrid*, *fpppp*, and *wave5*, as well as *stereo*, a multibaseline stereo benchmark from the CMU benchmark suite [25] that operates on three 256 by 240 integer arrays of image data. Only L1 Dcache and L2 cache energy dissipations are calculated as the L1 Icache and main memory energy dissipations do not change significantly with the number of enabled L1 Dcache ways.

4.2 Performance Degradation

Figure 4 shows the performance degradation incurred for each benchmark as the number of enabled ways is varied. For the 4-way set associative caches, performance degrades by 0-4% when only three ways are enabled, and more sharply thereafter. The degradation is particularly significant (up to 17%) when moving from two ways to only one way enabled. This is also the case for the 2-way set associative caches, in which performance degrades by 2-11% when only one of the two ways is enabled.

Figure 5 shows the same performance information but with the Load-Store Queue doubled to 16 entries. Due to the ability to overlap more memory activity[2], the relative impact of cache misses on IPC lessens with an increase in the Load-Store Queue depth, and therefore the performance degradation incurred with fewer data ways enabled is less severe. The change is especially striking for *stereo*, whose performance improves greatly with more concurrent load miss activity. The result is that performance degrades by only about 0-2% when only three ways are enabled for the 4-way set associative caches, and by about 2-6% for the 2-way caches with one enabled way.

The relative impact of conflict versus capacity misses can be assessed by observing that the 4-way caches are half the size of the corresponding 2-way caches for the same number of enabled ways. Cases where the 2-way associative curves parallel the 4-way curves between

2. Another effect is cache accesses being eliminated due to load address matches with previous stores in the queue, where these are too far apart in program order to be captured in the smaller queue. However, this effect has a very minor impact on performance and energy for the Load-Store Queue sizes and benchmarks that we studied.

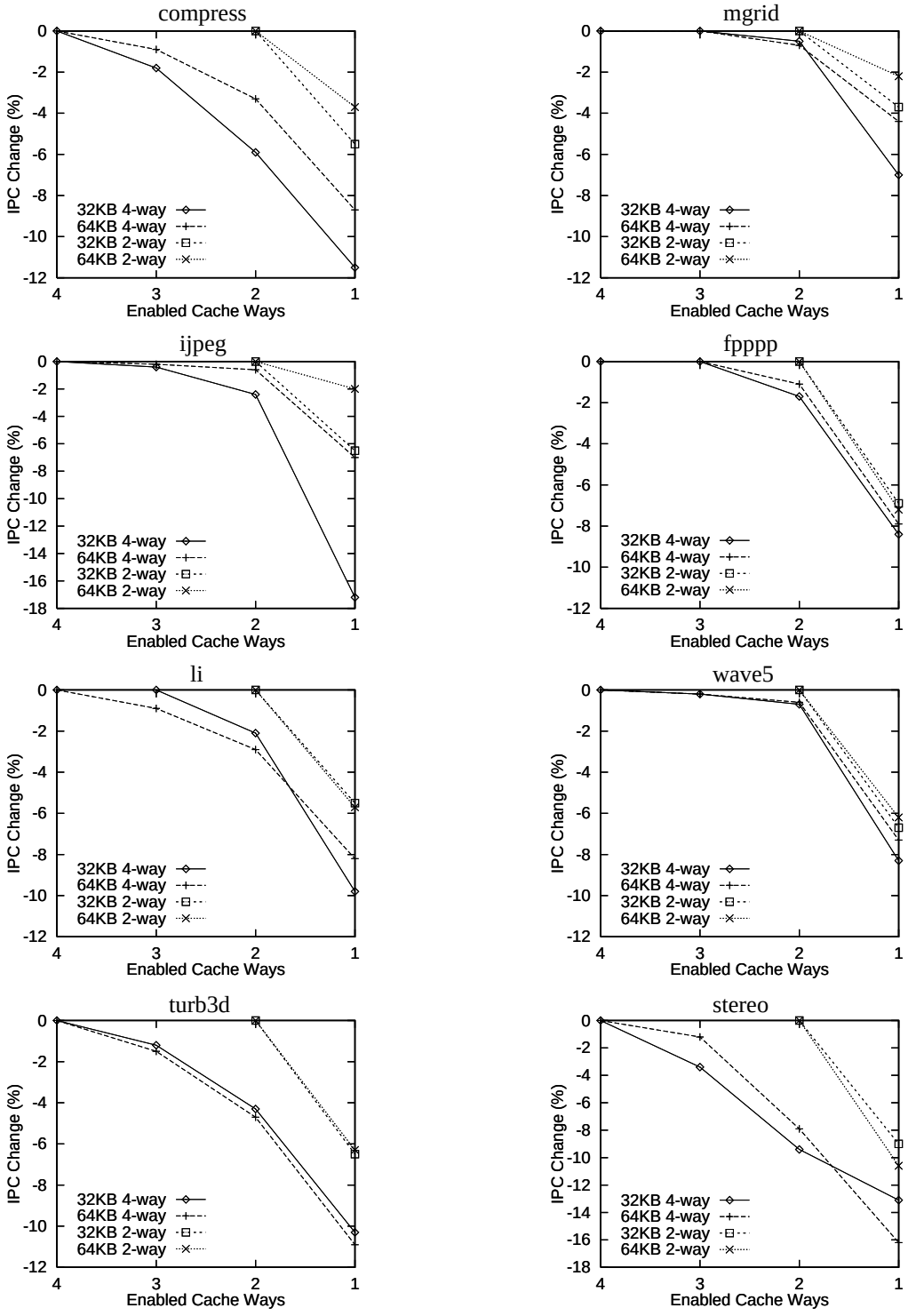

Figure 4: Performance degradation as a function of the number of ways enabled with a Load-Store Queue depth of 8 and a 1MB L2 cache. Note the different scales used for *ijpeg* and *stereo*.

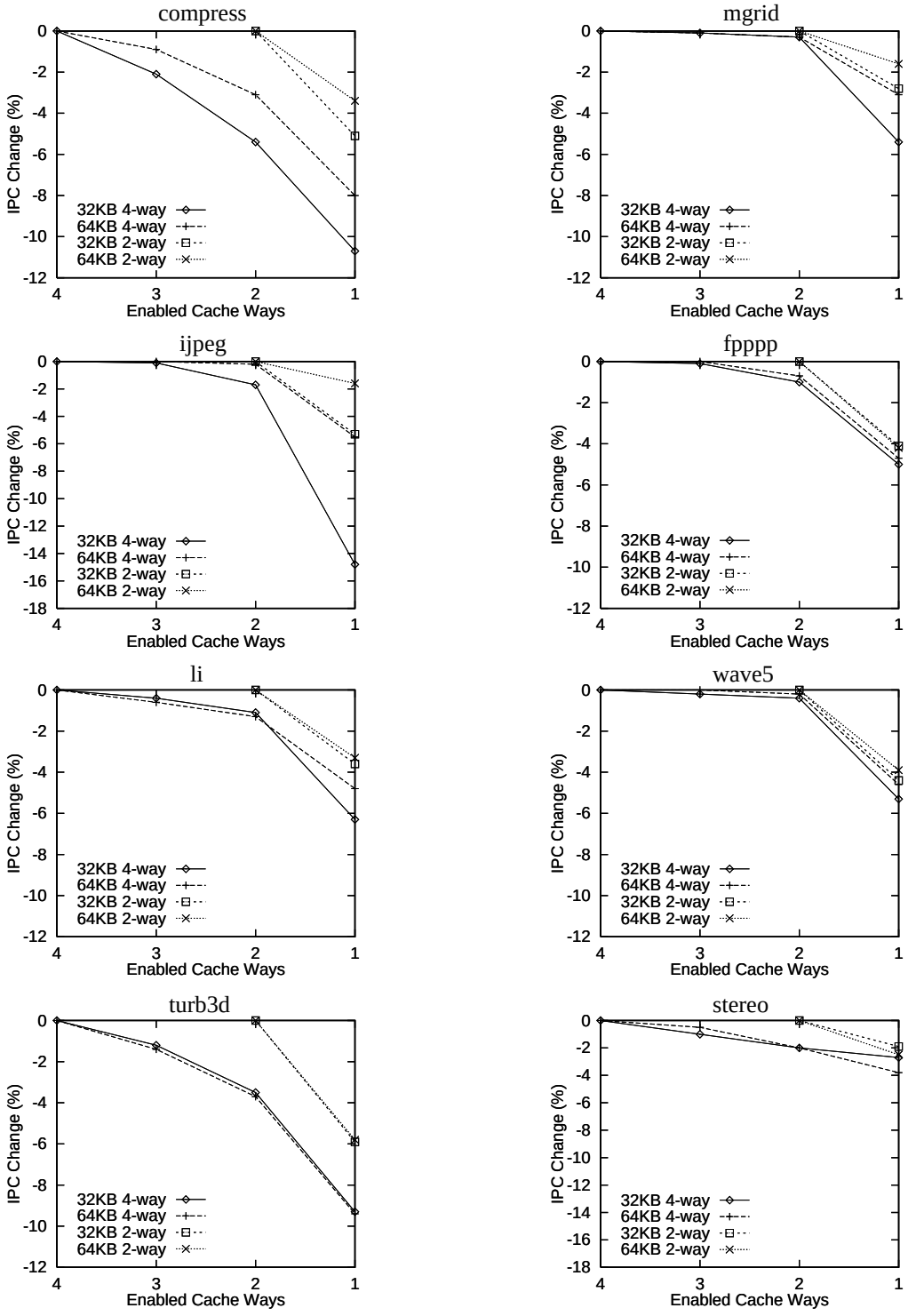

Figure 5: Performance degradation as a function of the number of ways enabled with a Load-Store Queue depth of 16 and a 1MB L2 cache. Again, the scales are different for *ijpeg* and *stereo*.

two and one enabled way, as with *turb3d*, indicate a high degree of conflict misses, since the percent performance change remains roughly the same even with twice the amount of cache. On the other hand, the domination of capacity misses is indicated by a faster drop-off for the 4-way curves relative to the same size 2-way curves, as with *ijpeg*.

4.3 Energy Impact

The impact on L1 Dcache and L2 cache energy dissipation depends on how disabling ways impacts the energy dissipations of each cache. Significant savings can be realized for benchmarks in which few additional L1 Dcache misses are generated when disabling ways. However, disabling ways may *increase* energy dissipation if there is a significant increase in L2 cache activity. Figure 6 shows this tradeoff. *Li*, *turb3d*, and *mgrid* show significant reductions in combined L1 Dcache and L2 cache energy dissipation for all L1 Dcache organizations. For *ijpeg*, *fpppp*, and *wave5*, energy dissipation is reduced when one or two of the ways of a 4-way set associative cache are disabled. When additional ways are disabled, L2 cache accesses increase substantially, increasing L2 cache energy, which may override the reduction in L1 Dcache energy. The results are mixed for the 2-way caches, with improvements for both *ijpeg* and *wave5* with one of the ways disabled, but a degradation for the 32KB 2-way cache for *fpppp*.

The results for *compress* and *stereo* demonstrate the critical need to accurately assess the impact of disabling ways on L1 and L2 cache energy dissipation. Although roughly a 10% energy savings is realized for these benchmarks when disabling one of the ways of a 64KB 4-way cache, energy dissipation may significantly increase when otherwise disabling ways. For *compress*, disabling three of the cache ways in a 32KB 4-way cache more than doubles energy dissipation with a 1MB L2 cache; with a 2MB L2 cache, the energy increases by 180%. Thus, system software must accurately determine the optimal number of ways as is discussed in Section 5. In the next section, we discuss the overall energy savings and performance impact of selective cache ways assuming that this support is in place.

4.4 Overall Energy and Performance Results

Figure 7 shows the energy savings realized and performance degradation incurred across all benchmarks for different PDT values. The energy savings is calculated from the average L1 Dcache and L2 cache energy dissipation of all benchmarks with all ways enabled, and the average with the best number of disabled ways for each benchmark. The best number is that which meets the PDT criterion and minimizes combined L1 Dcache and L2 cache energy dissipation. The performance degradation is similarly calculated from the corresponding IPC values. Note that the actual performance degradation incurred is significantly less than the PDT value. Observe also that for a 64KB 4-way cache and a PDT of 4%, which incurs an actual performance loss of less than 2%, a 20-35% average cache energy savings can be obtained. The results are more modest, yet still significant (10-25% energy reduction), for a 32KB 4-way cache. The lesser performance degradation observed with a larger L2 cache is primarily due to the more limited opportunities to disable L1 Dcache ways with the larger L2 cache, due to the higher energy dissipation incurred on an L1 cache miss.

The energy savings for 2-way set associative caches is significantly less than that of 4-way caches of the same size for two reasons. First, a large jump in conflict misses often

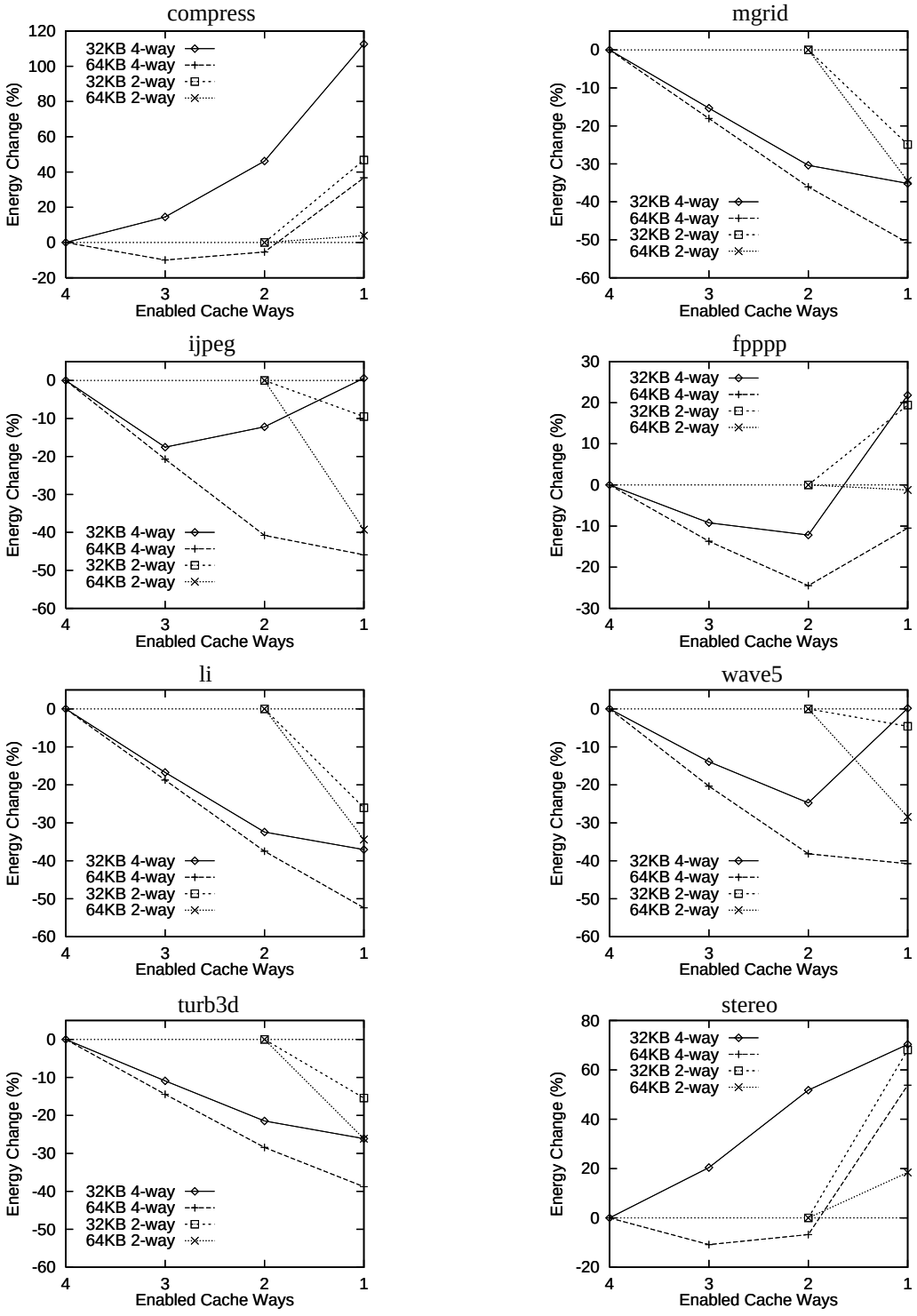

Figure 6: Relative change in combined L1 Dcache and L2 cache energy as a function of the number of ways enabled with a Load-Store Queue depth of 8 and a 1MB L2 cache. The results for a Load-Store Queue depth of 16 are virtually identical.

accompanies a move from two ways to only one way enabled. Second, the higher granularity of partitioning in the 4-way caches allows for the potential of disabling a greater percentage of the cache. Thus, in general, the effectiveness of selective cache ways increases with the cache associativity.

However, with more concurrent memory activity supported in the hardware, many applications become much less performance sensitive to the L1 cache organization. Figure 8 shows energy savings and performance degradation for a Load-Store Queue of 16 entries. For many applications, the performance degradation of enabling a subset of the ways is considerably less with the larger queue. Thus, a greater energy savings (both in relative and absolute terms) can be obtained for a given threshold value with more concurrent memory hierarchy activity. For example, for a PDT of 4%, which incurs an actual performance loss of less than 2%, a 25-40% average cache energy savings can be obtained for a 64KB 4-way set associative cache. The maximum energy savings is almost 45% for a 512KB L2 cache, but the performance degradation approaches 4%. Even the 64KB 2-way set associative cache achieves up to a 30% energy savings. Still, it is difficult to justify selective cache ways for the 32KB 2-way L1 Dcache considering that the energy savings barely exceeds 15% for a 512KB L2 cache, and is even less for the larger L2 caches.

Table 4, which displays the number of disabled ways for each benchmark as a function of PDT value, provides additional insight as to the effectiveness of selective cache ways. For the largest and most associative cache (64KB 4-way), at least one way is disabled for all benchmarks even with a PDT of only 2%. For the 32KB 4-way L1 Dcache, no ways are disabled for *compress* and *stereo* due to the resulting increase in cache energy (Figure 6). Fewer ways can be disabled compared with the 64KB cache for the other benchmarks, especially for larger PDT values. This limits the effectiveness of selective cache ways for a 32KB 4-way cache under this workload. For the 2-way caches, the PDT must be at least 4% in order for one way to be disabled for most of the benchmarks.

We conclude that a significant savings in combined L1 Dcache and L2 cache energy dissipation can be realized with selective cache ways, but this is primarily during periods in which applications with modest cache requirements dominate the workload. During periods when programs like *compress* and *stereo* predominate, very little energy savings may be realized for the simple reason that the full cache is needed. In addition, the change in performance and energy dissipation as the number of enabled cache ways is varied must be predictable with high accuracy; otherwise, a significant performance degradation and/or an increase in energy dissipation may result. We next describe software systems for application profiling and optimization that can be leveraged for selective cache ways.

5. Software Support for Selective Cache Ways

Several software profiling tools have been developed to identify portions of code that exhibit poor cache behavior, to provide insight to programmers for restructuring code to improve memory performance. Given certain high-level cache circuit characteristics, these same tools can also estimate the energy dissipation of different levels of the memory hierarchy. Cprof [26] provides cache miss rate information at a fine-grain source code level, and further classifies misses as compulsory, capacity, and conflict types. More recent tools, such as the profiling tool developed by Ammons et al. [27] and Compaq's ProfileMe [28]

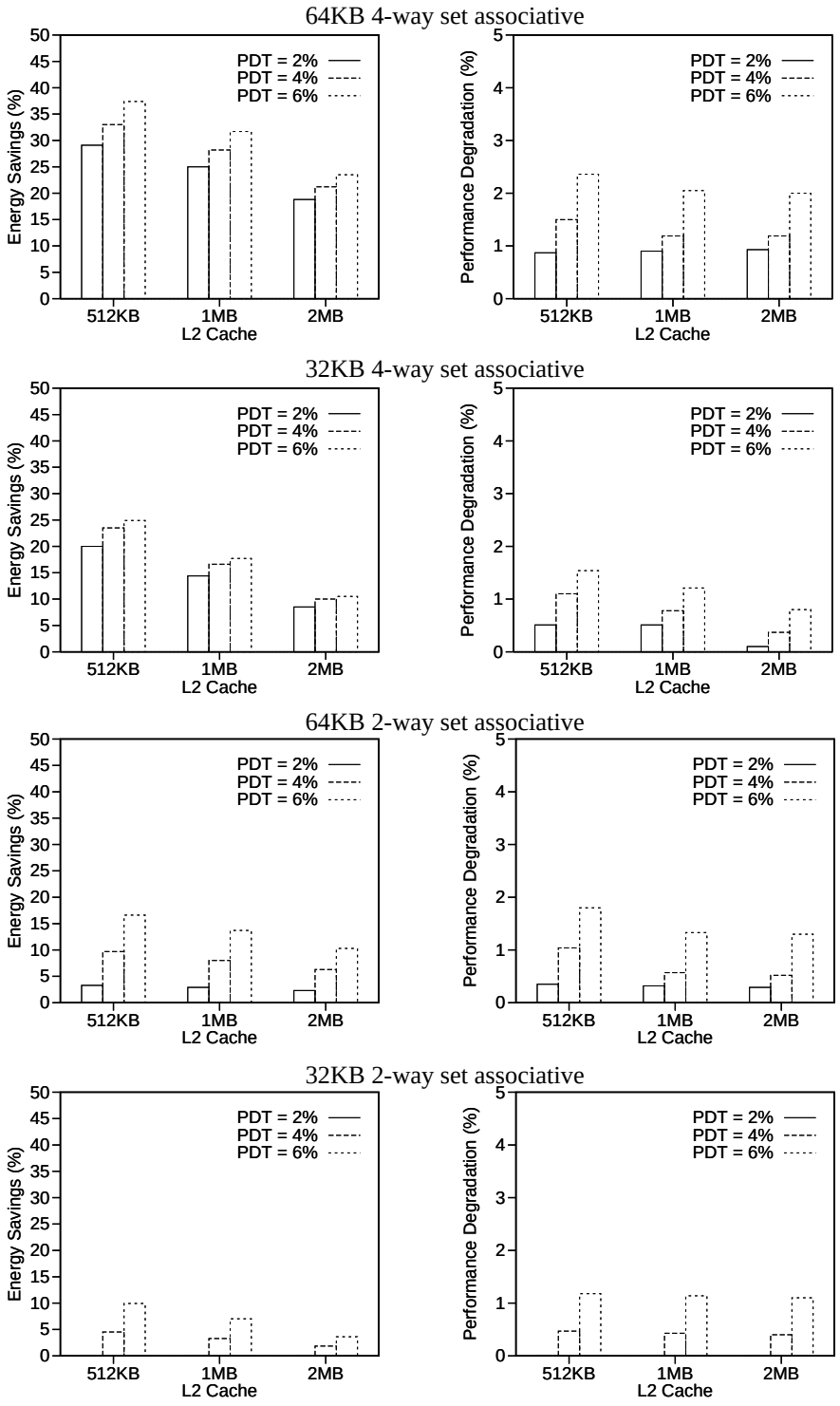

Figure 7: Combined L1 Dcache and L2 cache energy savings and actual performance degradation as a function of performance degradation threshold with a Load-Store Queue depth of 8.

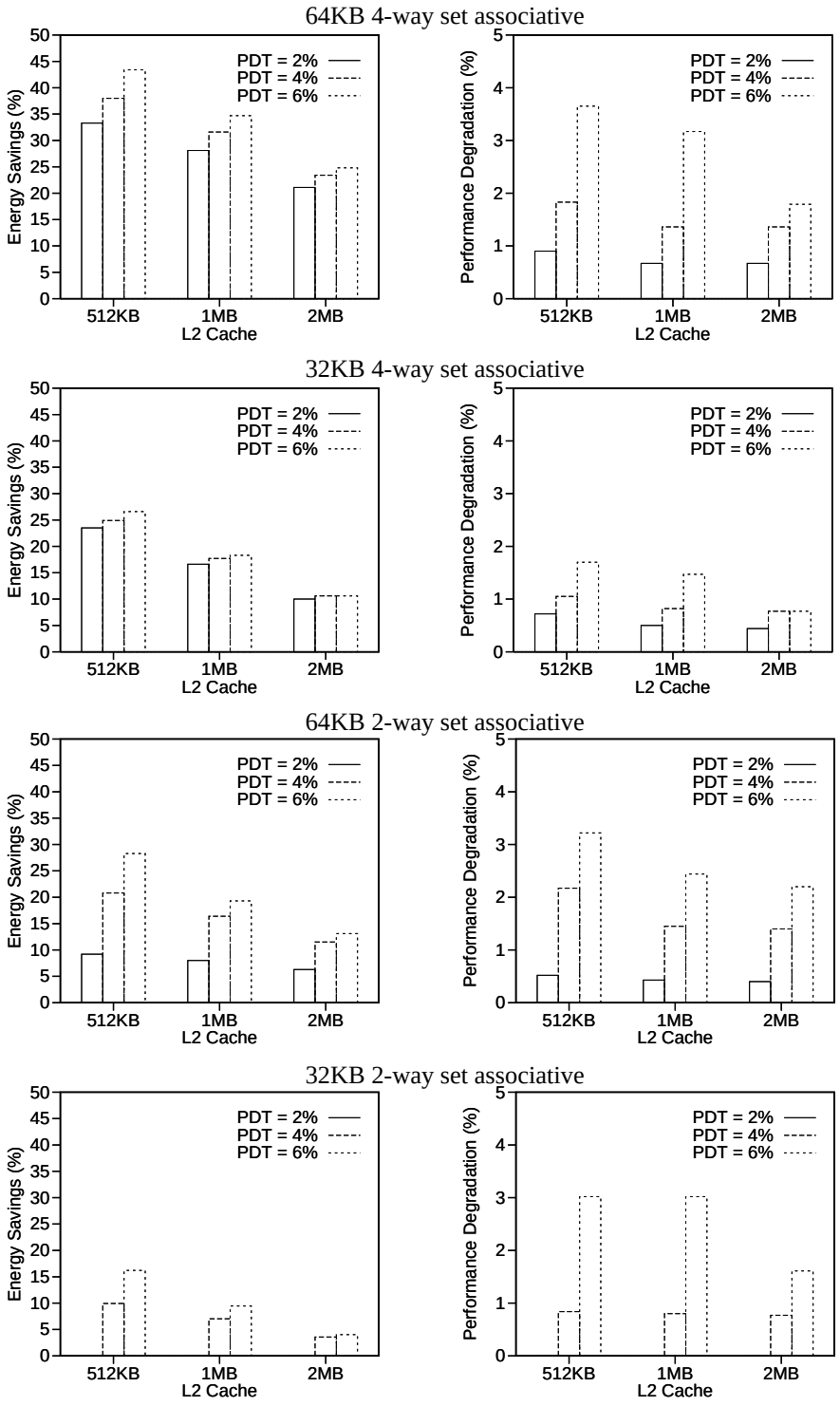

Figure 8: Combined L1 Dcache and L2 cache energy savings and actual performance degradation as a function of performance degradation threshold with a Load-Store Queue depth of 16.

Table 4: The number of disabled ways for each benchmark as a function of L1 Dcache organization and PDT value with a 1MB L2 cache and a Load-Store Queue depth of 16.

Cache Org	Benchmark	Performance Degradation Threshold		
		2%	4%	6%
		Disabled Ways	Disabled Ways	Disabled Ways
32KB 4-way	compress	0	0	0
	ijpeg	1	1	1
	li	2	2	2
	turb3d	1	2	2
	mgrid	2	2	3
	fpppp	2	2	2
	wave5	2	2	2
	stereo	0	0	0
64KB 4-way	compress	1	1	1
	ijpeg	2	2	3
	li	2	2	3
	turb3d	1	2	2
	mgrid	2	3	3
	fpppp	2	2	2
	wave5	2	2	3
	stereo	1	1	1
32KB 2-way	compress	0	0	0
	ijpeg	0	0	1
	li	0	1	1
	turb3d	0	0	1
	mgrid	0	1	1
	fpppp	0	0	0
	wave5	0	0	1
	stereo	0	0	0
64KB 2-way	compress	0	0	0
	ijpeg	1	1	1
	li	0	1	1
	turb3d	0	0	1
	mgrid	1	1	1
	fpppp	0	0	1
	wave5	0	1	1
	stereo	0	0	0

and DCPI [17] leverage the on-chip performance counters found on modern processors to provide additional insight on cache behavior. Rather than simply count events, ProfileMe and DCPI can be used to gather detailed information on a per-instruction basis, allowing for a more accurate assessment of cache performance and energy dissipation. By gathering such detailed information when all ways are enabled, it may be possible to accurately assess the appropriate number of cache ways to enable for different phases of application

execution without additional profiling runs for each way configuration. Another alternative with ProfileMe and DCPI is to provide low-cost continuous profiling of the application.

The Morph system [18] is designed to provide automatic program optimization via similar performance sampling techniques as well as on-the-fly modification of the application, *e.g.*, via a binary rewriting tool. With this type of software support, an application would be initially run with all ways enabled. Based on gathered cache information, estimates can be made on the performance degradation and energy dissipation with disabling ways for subsequent runs. In addition, the PDT (and thus, the number of disabled ways) could be adjusted according to machine conditions (loading, remaining battery life, *etc.*). The coupling of selective cache ways hardware with these emerging software techniques provides the opportunity for the tradeoff between performance and energy to be automatically made by the system without the need for extensive user involvement.

6. Conclusions and Future Work

Selective cache ways exploits the subarray partitioning of set associative caches in order to provide the capability to disable ways of the cache during periods where full cache functionality is not required to achieve good performance. Because only minor changes to a conventional cache are required to implement this technique, full-speed cache operation can be maintained. We have described the hardware and software aspects of this approach and demonstrated that a 40% reduction in overall cache energy dissipation can be achieved for 4-way set associative caches with less than a 2% overall performance degradation.

This *performance on-demand* approach of partitioning hardware resources and tailoring them to application requirements can be applied to the instruction cache and to other processor structures as well. For example, it is well known that branch prediction performance is very application-dependent, and therefore all of the elements of a combined predictor may not be needed for every application, nor may the full tables of the enabled predictors be necessary. As with selective cache ways, these requirements can be gleaned via the compiler or continuous software-based profiling, and the precise amount of branch prediction hardware needed so as not to exceed the performance threshold can be enabled, and the rest disabled to save energy. A similar approach can be applied to the other large, RAM-based structures that constitute a large fraction of the functionality in a modern microprocessor, and are often partitioned for performance reasons. In addition, partitioning by sets in addition to ways can be explored by manipulating the division between tag and index bits.

Like complexity-adaptive processors [29], the approach described in this paper exploits the fact that applications may drastically differ in their hardware requirements. In our future work, we plan to combine these complementary approaches in order to improve both the performance and energy dissipation, and thus the energy-delay product, of future high performance microprocessors.

Acknowledgements

John Strasser coded the cache energy dissipation model, and to the author's knowledge, George Cai of Intel first coined the term "performance on demand." This research is

supported by the National Science Foundation under CAREER Award CCR-9701915 and grant CCR-9811929.

References

[1] D. Dobberpuhl *et al.*, "A 200MHz, 64-bit, dual-issue CMOS microprocessor," *Digital Technical Journal*, vol. 4, pp. 35–50, Special Issue 1992.

[2] E. McLellan, "The Alpha AXP architecture and 21064 processor," *IEEE Micro*, vol. 13, pp. 36–47, June 1993.

[3] W. Bowhill *et al.*, "Circuit implementation of a 300-MHz 64-bit second-generation CMOS Alpha CPU," *Digital Technical Journal*, vol. 7, pp. 100–118, Special Issue 1995.

[4] J. Edmondson *et al.*, "Internal organization of the Alpha 21164, a 300MHz 64-bit quad-issue CMOS RISC microprocessor," *Digital Technical Journal*, vol. 7, pp. 119–135, Special Issue 1995.

[5] R. Kessler, "The Alpha 21264 microprocessor," *IEEE Micro*, vol. 19, pp. 24–36, March/April 1999.

[6] R. Kessler, E. McLellan, and D. Webb, "The Alpha 21264 microprocessor architecture," *International Conference on Computer Design*, October 1998.

[7] P. Bannon, "Alpha 21364: A scalable single-chip SMP," *Microprocessor Forum*, October 1998.

[8] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473–484, April 1992.

[9] M. Tremblay and J. O'Connor, "UltraSparc I: A four-issue processor supporting multimedia," *IEEE Micro*, vol. 16, pp. 42–50, April 1996.

[10] T. Horel and G. Lauterbach, "UltraSPARC III: Designing third-generation 64-bit performance," *IEEE Micro*, vol. 19, pp. 73–85, May/June 1999.

[11] J. Montanaro *et al.*, "A 160-MHz, 32-b, 0.5W CMOS RISC microprocessor," *Digital Technical Journal*, vol. 9, no. 1, pp. 49–62, 1997.

[12] J. Kin, M. Gupta, and W. Mangione-Smith, "The filter cache: An energy efficient memory structure," *Proceedings of the 30th International Symposium on Microarchitecture*, pp. 184–193, December 1997.

[13] N. Bellas *et al.*, "Architectural and compiler support for energy reduction in the memory hierarchy of high performance microprocessors," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 70–75, August 1998.

[14] G. Lesartre and D. Hunt, "PA-8500: The continuing evolution of the PA-8000 family," *Proceedings of Compcon*, 1997.

[15] S. Wilton and N. Jouppi, "An enhanced access and cycle time model for on-chip caches," Tech. Rep. 93/5, Digital Western Research Laboratory, July 1994.

[16] T. Wada, S. Rajan, and S. Przybylski, "An analytical access time model for on-chip cache memories," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 1147–1156, August 1992.

[17] J. Anderson *et al.*, "Continuous profiling: Where have all the cycles gone?," *Proceedings of the 16th Symposium on Operating Systems Principles*, October 1997.

[18] X. Zhang *et al.*, "System support for automatic profiling and optimization," *Proceedings of the 16th Symposium on Operating Systems Principles*, October 1997.

[19] G. Henry, P. Fay, B. Cole, and T. Mattson, "The performance of the Intel TFLOPS supercomputer," *Intel Technology Journal*, Q1 1998.

[20] B. Xu and D. Albonesi, "A methodology for the analysis of dynamic application parallelism and its application to reconfigurable computing," *Proceedings of the SPIE International Symposium on Reconfigurable Technology: FPGAs for Computing and Applications*, pp. 78–86, September 1999.

[21] A. Argawal, J. Hennessy, and M. Horowitz, "Cache performance of operating system and multiprogramming workloads," *ACM Transactions on Computer Systems*, vol. 6, pp. 393–431, November 1988.

[22] D. Burger and T. Austin, "The SimpleScalar toolset, version 2.0," Tech. Rep. TR-97-1342, University of Wisconsin-Madison, June 1997.

[23] A. Kumar, "The HP PA-8000 RISC CPU," *IEEE Computer*, vol. 17, pp. 27–32, March 1997.

[24] M. Kamble and K. Ghose, "Analytical energy dissipation models for low power caches," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 143–148, August 1997.

[25] P. Dinda *et al.*, "The CMU task parallel program suite," Tech. Rep. CMU-CS-94-131, Carnegie Mellon University, March 1994.

[26] A. Lebeck and D. Wood, "Cache profiling and the SPEC benchmarks: A case study," *IEEE Computer*, vol. 27, pp. 15–26, October 1994.

[27] G. Ammons, T. Ball, and J. Larus, "Exploiting hardware performance counters with flow and context sensitive profiling," *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 1997.

[28] J. Dean *et al.*, "ProfileMe: Hardware support for instruction-level profiling in out-of-order processors," *Proceedings of the 30th International Symposium on Microarchitecture*, pp. 292–302, December 1997.

[29] D. Albonesi, "Dynamic IPC/clock rate optimization," *Proceedings of the 25th International Symposium on Computer Architecture*, pp. 282–292, June 1998.