AN ARCHITECTURAL AND CIRCUIT-LEVEL APPROACH TO IMPROVING THE ENERGY EFFICIENCY OF MICROPROCESSOR MEMORY STRUCTURES

David H. Albonesi
Dept. of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627-0231 USA[*]
albonesi@ece.rochester.edu

Abstract We present a combined architectural and circuit technique for reducing the energy dissipation of microprocessor memory structures. This approach exploits the subarray partitioning of high speed memories and varying application requirements to dynamically disable partitions during appropriate execution periods. When applied to 4-way set associative caches, trading off a 2% performance degradation yields a combined 40% reduction in L1 Dcache and L2 cache energy dissipation.

1. INTRODUCTION

The continuing microprocessor performance gains afforded by advances in semiconductor technology have come at the cost of increased power consumption. Each new high performance microprocessor generation brings additional on-chip functionality, and thus an increase in switching capacitance, as well as increased clock speeds over the previous generation. For example, both transistor count and clock speed have roughly doubled in the three years separating the Alpha 21164 microprocessor [6, 11] and the recently introduced Alpha 21264 [14, 15]. Because the dynamic power, which is currently the dominant contributor to power consumption in high-speed CMOS circuits, is linearly related to each of these factors, it is extremely difficult for circuit-level techniques (such as voltage scaling) to singlehandedly keep power consumption from increasing under these circumstances. Indeed, the power consumption of the 21264 is 1.5-2 times that of the most recent 21164 version despite the fact that the voltage has been reduced from 3.3V to 2.2V [8, 9]. Similarly, while the UltraSparc I microprocessor [20], which was introduced in 1995, dissipated 28W at 167MHz, the forthcoming UltraSparc III design [12] is estimated to

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

dissipate 70W at 600MHz. For these reasons, in order to prevent microprocessor designers from being limited by power and energy dissipation, especially in desktop and portable environments where heat dissipation and battery life are critical constraints, it is necessary to devise architectural techniques for low power that complement circuit-level approaches.

Because of the increasing usage of microprocessor die area for on-chip caches, several architectural-level approaches to reducing *energy dissipation* in these structures have been devised. These techniques seek to reduce the amount of switching activity within the hardware for a given workload. The SA-110 embedded microprocessor [19] 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. However, with larger on-chip caches and frequencies in the GHz range, such a solution would likely increase L1 cache latencies. The resulting increase in the branch mispredict penalty and in load latency would significantly degrade performance for 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 portion 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] (as determined by the tag hit logic). Because only a small fraction of the total L2 cache is enabled on each access, the dynamic power savings is considerable, estimated at 10W [11]. However, the serial tag-data access of this technique increases cache latency as with the SA-110 approach, and thus this approach is limited to on-chip memory structures where overall performance is relatively insensitive to the latency of the structure.

Several other approaches, such as the filter cache [16] and the L-Cache [5] have been proposed for reducing the switching activity. However, each of these techniques significantly alters the on-chip memory design in order to improve energy efficiency. This ultimately results in a non-trivial performance degradation or other limitations. In addition, these schemes only address caches whereas non-trivial amounts of energy may also be dissipated in other memory structures such as Translation Lookaside Buffers (TLBs), register files, branch predictors, and instruction queues. In this paper, we introduce an alternative and more general approach: that of leveraging the subarray partitioning that is

[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.

often present in large on-chip memories for speed reasons in order to provide the ability to *selectively enable* particular partitions of the memory. With appropriate architectural support, these partitions can be *dynamically* enabled in an *on-demand* fashion. That is, the full unmodified memory structure is enabled when necessary to obtain good application performance, but only a subset is enabled during periods where application requirements are more modest. Such a *performance on demand* approach exploits that fact that hardware demands vary from application to application, and may also vary during the execution of an individual application [2, 23]. With the ability to enable only the precise amount of on-chip memory needed to meet performance requirements, a significant amount of energy savings can be realized in the on-chip memories, and thus in the overall microprocessor. However, unlike many other previous approaches to energy savings, this technique delivers identical performance to a conventional memory design when required by the application.

In the rest of this paper, we explore this concept in further detail. In the next section, we examine the subarray partitioning that is often necessary to minimize the access time of memory structures, and how this partitioning can be exploited to tailor the memory organization to application requirements. We then discuss in Section 3 our approach for selectively enabling partitions, including schemes for properly handling information in a disabled partition. In Section 4 we explore the application of this technique to the L1 Dcache, and evaluate the energy savings that can be realized when some performance degradation can be traded off for reduced switching activity. Finally, we conclude and present future work in Section 5.

2. THE ORGANIZATION OF ON-CHIP MEMORIES

In this section, we use a modified version of the Cacti cache cycle time model [22] to explore the partitioning of on-chip memories that is necessary to optimize access time. Cacti is an analytical delay model that evaluates in detail each component of the data and tag delay paths. In addition, Cacti includes six layout parameters that allow for partitioning of single tag and data arrays into multiple subarrays. 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 bitline parameters [21]. The parameters *Nspd* and *Ntspd* refer to the number of sets that are mapped to the same wordline for the data and tag arrays, respectively [22].

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

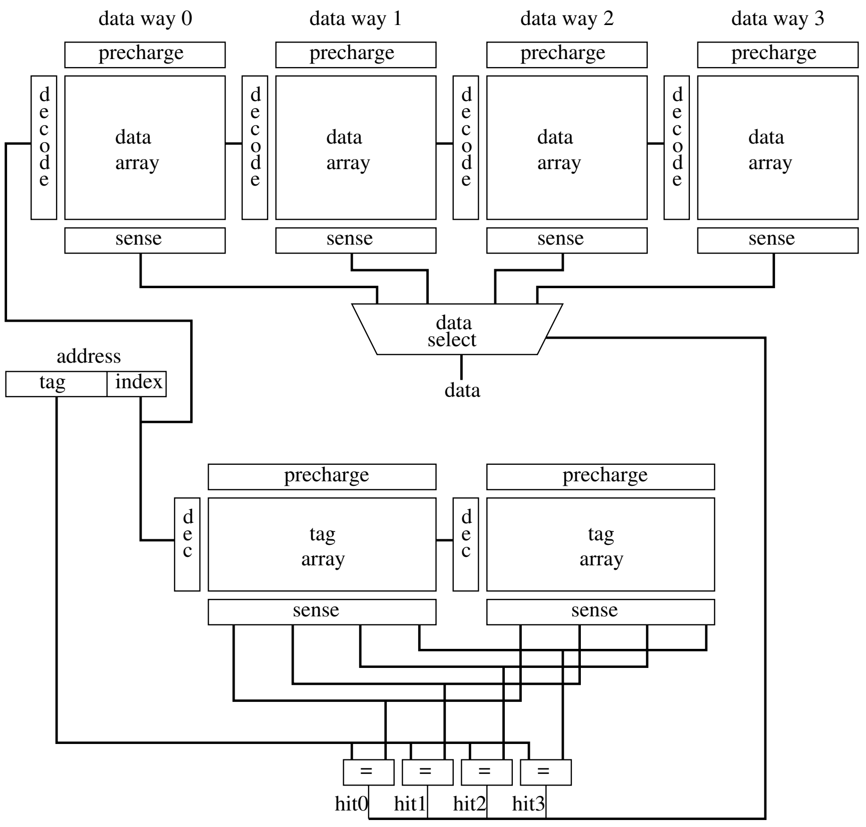


Figure 1 A 4-way set associative cache with *Ndwl* = 4, *Ntbl* = 2 and all other *N* parameters equal to one.

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. Note that by segmenting the tag array, an extra output selector delay is incurred. This can be implemented by activating only one of the two sets of tag sense amps associated with the same column during each access [21].

We used Cacti to explore the partitioning necessary to minimize the access time of two different on-chip memories: TLBs and caches. The results are shown in Table 1, which displays the *N* parameters that produce the fastest access time for various organizations. Each *N* parameter is limited to a maximum value of eight to avoid unreasonable aspect ratios. Note that in all cases, a partitioning of both the data and tag arrays into multiple subarrays is necessary to minimize access time. For the TLBs, the data wordlines need to be segmented

Table 1 Optimal *N* parameters for on-chip TLB and cache structures. The block size is 16 bytes for TLBs and 32 bytes for caches.

Structure	Organization		Optimal N Parameters					
	Size	Associativity	Ndwl	Ndbl	Nspd	Ntwl	Ntbl	Ntspd
TLB	1KB	4	8	1	1	2	1	1
	2KB	4	8	1	1	2	1	1
	4KB	4	8	1	1	2	1	1
Cache	8KB	1	2	4	1	1	2	2
		2	4	2	1	1	2	1
		4	8	1	1	2	1	1
	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

eight times, while a combination of data wordline and bitline segmentation is often required for optimal cache performance.

Although there are several aspects to Cacti that limit the applicability of these results, on-chip caches are often partitioned in practice. For example, the 32KB 2-way set associative and 2-way banked L1 Dcache in the R10000 microprocessor is partitioned into four subarrays [1, 24], as is each of the two 512KB data banks of the 1MB 4-way set associative L1 Dcache of the HP PA-8500 microprocessor [18].

Note also that although we did not evaluate other on-chip memory structures, many of these need to be similarly partitioned for speed-optimality. For example, branch prediction tables are typically tagless structures whose size has increased significantly in recent microprocessors. For example, while the Alpha 21164 employed a single 2K$\times$2 branch direction predictor table, the Alpha 21264 implements four tables of sizes 1K$\times$10, 1K$\times$3, 4K$\times$2, and 4K$\times$2. The latter predictor is 3.625KB in size, over seven times that in the 21164, and may consume non-trivial amounts of energy. Each of the predictor tables when implemented as a single array would be a very long and narrow memory structure, and therefore, the bitline delay would be proportionally much longer than the other delay components. Thus, segmentation of the bitlines would likely reduce the access time of these large tables. In the next section, we discuss the mechanisms necessary to exploit this partitioning such that the enabling and disabling of memory partitions can be manipulated under software control.

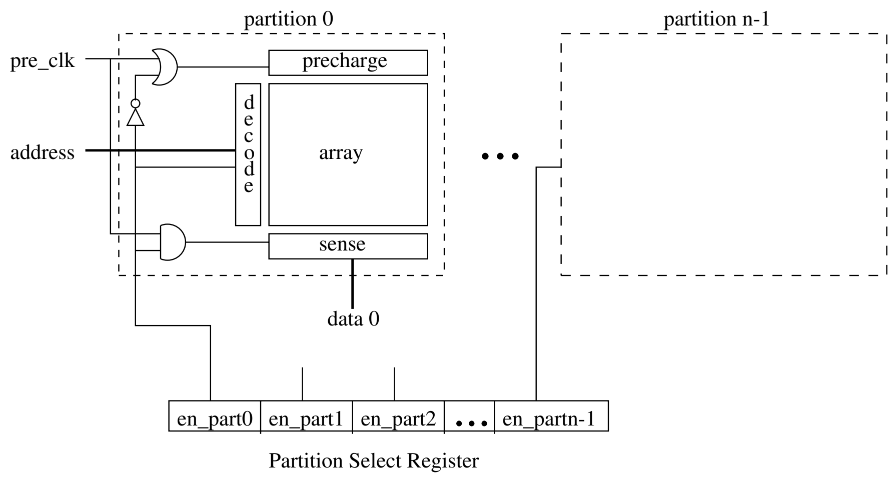


Figure 2 An on-chip organized as *n* partitions, each of which can be enabled/disabled via the Partition Select Register.

3. MECHANISMS FOR DISABLING MEMORY PARTITIONS

Figure 2 is an overall diagram showing a memory structure which has been partitioned into subarrays for speed purposes, and the small amount of additional circuitry needed to selectively disable each memory partition, which is comprised of one or more subarrays. The gating logic in this diagram is based on that used to selectively activate data subarrays in the Alpha 21164 L2 cache [6]. Each bit in the Partition Select Register (PSR) controls the enabling of one of the *n* memory partitions. If a particular bit is set to zero, then that partition is not precharged, no word lines are selected, and its sense amps are prevented from firing. Thus, no switching activity ensues and thus this partition dissipates essentially no dynamic power.

A PSR may concatenate partition control bits for several on-chip memory structures, and it may be necessary to include several PSRs in the processor. Each PSR is software readable and writable through special instructions. Several possible sources are possible for control of the PSRs including the compiler, the runtime system, the core operating system (which also needs to save the PSRs as part of the process state on context switches), special priviledged routines such as those that manage the TLB on several processors, or a continuous profiling and optimization system [4, 25]. Another approach is to have hardware that dynamically detects when partitions can be disabled. But this hardware would be complex and dissipate energy, thereby mitigating some of the benefits of our approach. These architectural issues are beyond the scope of this paper and are further discussed in [3].

Note that our ability to exploit subarray partitioning is dependent on how the partitioning divides the information stored in the memory. For example, if only the wordlines of an instruction queue are segmented, then disabling a partition would eliminate access to part of the instruction information in every queue entry, and thus the processor would not correctly operate. For memory structures that consist of a tag array in addition to a data array, different subarray partitioning may be employed. In this case, the disabling of tag and data partitions must be such that hits are masked for disabled data partitions, and that hits are detectable for enabled data partitions. In some cases, this may mean that we cannot take advantage of the tag partitioning in terms of disabling partitions and saving power. (We discuss this further in Section 4.) Thus, our scheme cannot be used for every possible partitioning of a memory structure, and in these cases, the partitioning needs to be changed from that which is speed-optimal in order to provide the ability to disable partitions. The resulting increase in access time may critically impact performance as we describe in Section 4. Therefore, in this paper we assume that we do not deviate from the speed-optimal partitioning of Table 1 in applying this approach to on-chip memory structures.

3.1 PRESERVING INFORMATION TO ENSURE CORRECT OPERATION

For some on-chip memory structures, partitions can be disabled without regard to the accessibility of the information in the disabled array. This is the case for instruction caches, branch predictors, TLBs, and register files. Both instruction caches and TLBs create no new information but rather provide faster access than main memory, from which the same information can be extracted. By ensuring that instruction TLB entries are invalidated before they are disabled, we can ensure correct operation of both the instruction TLB and instruction cache whenever entries are re-enabled. Similarly, the absence of branch prediction information may simply cause more mispredictions, but these do not result in incorrect execution. Any performance penalty resulting from having to reload instruction caches and TLBs or reconstruct branch prediction information can usually be mitigated by limiting the rate at which partitions are disabled, *e.g.*, only during context switches.

Similarly, if the register file is appropriately partitioned, no action is required by the hardware to preserve data in a disabled partition. Because the register file is compiler-managed, the compiler can precisely determine application register usage and insert instructions to disable register file partitions when requirements are modest. When more registers are required, or when the values in these disabled registers are to be used, the compiler re-enables the required partitions. The full register file must also be enabled by the operating system when saving process state to memory. The issues associated with register renaming are more complex and are not discussed in this paper.

For other structures, incorrect operation can result unless the information in a disabled partition is preserved or made accessible. For example, the partitions of instruction queues and reorder buffers cannot be disabled until the instructions associated with the disabled entries have completed execution and committed their results. Thus, some time must elapse between the execution of an instruction that writes the PSR and the actual disabling of these structures. During this period (which may last 20-40 cycles on a modern microprocessor that issues two instructions per cycle on average), no new instructions must be placed in the to-be-disabled partitions. To speed up this process, the instruction scheduling hardware can raise the priority of the instructions in these partitions.

The disabling of partitions in the L1 data cache (Dcache) and the L2 cache is not as straightforward. For these structures, modified cache blocks must be made accessible to sharing processes (on the same processor or other processors), to the same process (either on the same or another CPU), and to the I/O subsystem. In addition, the coherence state of data in a disabled partition must be properly maintained in case the partition is later re-enabled. These issues are addressed in [3] in which we discuss the performance on-demand approach of a set associative cache in which ways can be dynamically enabled to meet performance demands. In the next section, we briefly describe this approach as an example of saving energy in on-chip memory structures.

4. A PERFORMANCE ON-DEMAND L1 DCACHE

In this section, we describe the application of disabling memory partitions to on-chip caches. Specifically, we quantify the energy savings obtained, and performance degradation incurred, with set associative L1 Dcaches in which the data ways of the cache can be selectively disabled using the methods described earlier and in the rest of this section. The hardware organization of this approach, which we call *selective cache ways*, is described in the next section.

4.1 HARDWARE ORGANIZATION

Figure 3 is an overall diagram of a 4-way set associative cache using selective cache ways. The wordlines of the data array are segmented either four or eight times according to the Cacti results of Table 1, creating four separate data way partitions. The bitlines of each data way may be segmented as well, although this is not shown in the diagram. Note however, that the tag portion of the cache (which also includes the status bits) is identical to that

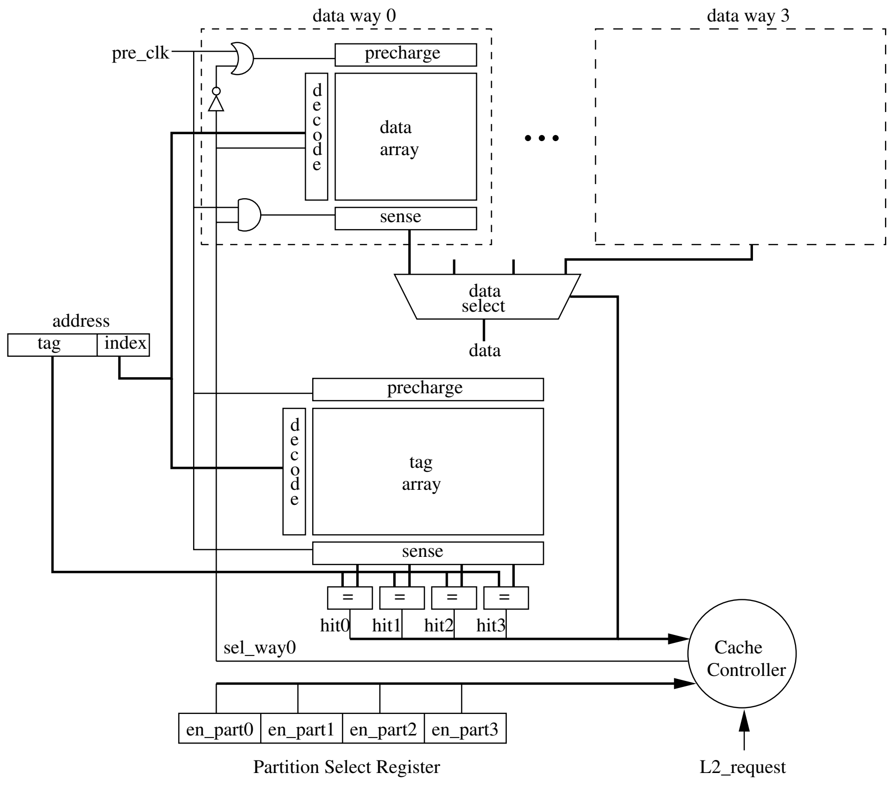


Figure 3 A 4-way set associative cache using selective cache ways. The details for data ways 1-3 are identical to way 0 but are not shown for simplicity.

Table 2 Cache cycle time degradation of tag wordline partitioning relative to the speed-optimal cache partitioning.

Cache Org	Degradation
32KB 2-way	3.7%
64KB 2-way	7.8%
32KB 4-way	4.3%
64KB 4-way	7.3%

of a conventional cache. Our Cacti-based timing estimates indicate that for the cache organizations we have studied, segmenting the tag wordlines can in some cases result in a significant cache cycle time degradation relative to the optimal tag *N* parameters of Table 1. For example, Table 2 shows that the cycle time degradation incurred when the tag wordlines are segmented is roughly 4-8% for 32KB and 64KB set associative caches. Because the L1 Dcache is

Table 3 Simulated memory hierarchy parameters.

Mem Level	Organization
L1 Icache	64KB, 4-way set assoc, 32B block, random, 1 cycle latency
L1 Dcache	64KB, 4-way set assoc, selective cache ways, 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

typically a critical path, especially for caches as large as those in this table, this degradation may result in an overall cycle time increase. For these reasons, we used the *N* parameters of Table 1 and therefore only save energy in the data portion of the cache. However, the data portion comprises roughly 90% of the total energy dissipation for the cache organizations we have studied.

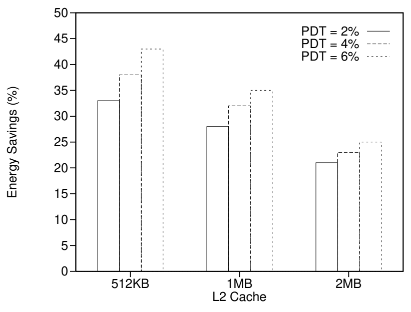
Note from Figure 3 that the outputs of the PSR do not directly control the enabling of the arrays (as was the case in Figure 2), but rather are sent to the Cache Controller. This is to allow the Cache Controller the ability to access modified data in disabled ways as well as to properly handle coherence transactions as is discussed in [3].

4.2 ENERGY AND PERFORMANCE EVALUATION

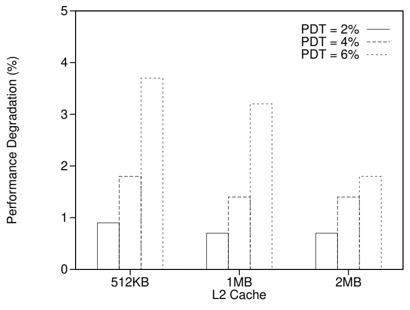
In this section, we quantify the performance degradation incurred, and the energy savings obtained, with an L1 Dcache using selective cache ways. Our evaluation methodology combines detailed processor simulation for performance analysis and for gathering event counts, and analytical modeling for estimating the energy dissipation of both conventional caches and caches employing selective cache ways.

We use the SimpleScalar toolset [7] to model a modern 4-way out-of-order speculative processor with a two-level cache hierarchy that roughly corresponds to a current high-end microprocessor such as the HP PA-8000 [17] and Alpha 21264 [15]. Table 2 shows the simulator parameters for the memory hierarchy. Selective cache ways is implemented for only the L1 Dcache. The data array of the L2 cache is implemented as 16 partitions, only one of which is selected for each access. This power-saving technique is used in the Alpha 21164 on-chip L2 cache [6].

We estimate L1 Dcache and L2 cache energy dissipations using a modified version of the analytical model of Kamble and Ghose [13]. This model calculates in detail cache energy dissipation using technology and layout parameters



(a)



(b)

Figure 4 (a) Combined L1 Dcache and L2 cache energy savings and (b) actual performance degradation as a function of the performance degradation threshold.

from Cacti and counts of various cache events (hits, writebacks, *etc.*) as inputs. These event counts, in addition to performance results, are gathered from SimpleScalar simulations (each 400 million instructions long) of eight benchmarks: the SPEC95 benchmarks *compress*, *ijpeg*, *li*, *turb3d*, *mgrid*, *fpppp*, and *wave5*, as well as *stereo*, a multibaseline stereo benchmark from the CMU benchmark suite [10] that operates on three 256 by 240 integer arrays of image data. The number of enabled ways is determined based on overall application cache characteristics, and therefore the number of enabled cache ways is only changed during context switches. 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.

The energy savings of selective cache ways depends on the amount of performance that can be traded off for energy. The *Performance Degradation Threshold (PDT)* signifies the average performance degradation relative to a cache with all ways enabled that is allowable for a given period of execution. If the PDT is 2%, and, for a given period of execution, performance is projected to degrade by 1% with three ways enabled, and 4% with two ways enabled, then three ways are enabled for that period of execution, so long as the total energy is less than that with all four ways enabled. This would not be the case if the extra misses with three ways enabled increase L2 cache energy more than the energy savings obtained with disabling one of the L1 Dcache ways. In this case, all four ways are enabled. In this study, the optimum number of enabled ways for each benchmark is determined from comparing performance and energy dissipation results. In an actual system, a runtime system such as Compaq's DCPI [4] can read cache hierarchy performance counters and make changes based on knowledge of relative L1 and L2 cache energy dissipations.

Figure 4 shows the energy savings and actual performance degradation incurred across all benchmarks as a function of the PDT. The energy savings is calculated from the average energy dissipation of all benchmarks with all ways enabled, and the average with the number of disabled ways allowable for a given PDT value. The performance degradation is similarly calculated from the corresponding Instructions Per Cycle results. The actual performance degradation incurred is significantly less than the PDT value. Overall, roughly a 40% cache hierarchy energy savings is realized with less than a 2% performance degradation for a 512KB L2 cache with a PDT of 4%. The benefits are less, yet still significant, for larger L2 caches due to the higher energy dissipated servicing an L1 Dcache miss. Even with a large 2MB on-chip L2 cache, a 25% energy savings is obtained with less than a 2% performance degradation using this technique.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have described techniques for leveraging the subarray partitioning of high-speed memory structures in order to dynamically tailor the organization of these structures to application requirements. We have discussed the mechanisms necessary to allow software to dynamically modify which partitions are disabled, as well as to ensure the proper handling of information in disabled partitions. Through detailed simulation and analytical modelling, we demonstrated that a 40% reduction in overall cache energy dissipation can be achieved for 4-way set associative L1 Dcaches with only a 1-2% overall performance degradation.

Our future work includes applying this technique in concert to multiple on-chip memory structures, as well as exploring how these structures can be dynamically tailored to changing requirements during individual application execution. Finally, we plan to combine these efforts with our previous work on dynamic speed-complexity performance tradeoffs [2] in order to dynamically optimize the energy-delay product of future high performance microprocessors.

Acknowledgments

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."

References

[1] A. Ahi et al. The R10000 superscalar microprocessor. *Hot Chips VII Symposium*, August 1995.

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

[3] D.H. Albonesi. Selective cache ways: On-demand cache resource allocation. *Proceedings of the 32nd International Symposium on Microarchitecture*, November 1999.

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

[5] 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*, pages 70–75, August 1998.

[6] W.J. Bowhill et al. Circuit implementation of a 300-MHz 64-bit second-generation CMOS Alpha CPU. *Digital Technical Journal*, 7(1):100–118, Special Issue 1995.

[7] D. Burger and T.M. Austin. The SimpleScalar toolset, version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[8] Digital Equipment Corporation. Alpha 21164 microprocessor data sheet. August 1998.

[9] Digital Equipment Corporation. Alpha 21264 microprocessor data sheet. February 1999.

[10] P. Dinda et al. The CMU task parallel program suite. Technical Report CMU-CS-94-131, Carnegie Mellon University, March 1994.

[11] J.H. Edmondson et al. Internal organization of the Alpha 21164, a 300MHz 64-bit quad-issue CMOS RISC microprocessor. *Digital Technical Journal*, 7(1):119–135, Special Issue 1995.

[12] T. Horel and G. Lauterbach. UltraSPARC III: Designing third-generation 64-bit performance. *IEEE Micro*, 19(3):73–85, May/June 1999.

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

[14] R. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

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

[16] J. Kin, M. Gupta, and W.H. Mangione-Smith. The filter cache: An energy efficient memory structure. *Proceedings of the 29th International Symposium on Microarchitecture*, pages 184–193, December 1997.

[17] A. Kumar. The HP PA-8000 RISC CPU. *IEEE Computer*, 17(2):27–32, March 1997.

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

[19] J. Montanaro et al. A 160-MHz, 32-b, 0.5W CMOS RISC microprocessor. *Digital Technical Journal*, 9(1):49–62, 1997.

[20] M. Tremblay and J.M. O'Connor. UltraSparc I: A four-issue processor supporting multimedia. *IEEE Micro*, 16(2):42–50, April 1996.

[21] T. Wada, S. Rajan, and S.A. Przybylski. An analytical access time model for on-chip cache memories. *IEEE Journal of Solid-State Circuits*, 27(8):1147–1156, August 1992.

[22] S.J.E. Wilton and N.P. Jouppi. An enhanced access and cycle time model for on-chip caches. Technical Report 93/5, Digital Western Research Laboratory, July 1994.

[23] B. Xu and D.H. Albonesi. A methodology for the analysis of dynamic application parallelism and its application to reconfigurable computing. *Proceedings of the SPIE International Symposium on Voice, Video, and Data Communications (Track: Reconfigurable Technology: FPGAs for Computing and Applications)*, September 1999.

[24] K.C. Yeager. The Mips R10000 superscalar microprocessor. *IEEE Micro*, 16(2):28–41, April 1996.

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