

Segmented Bitline Cache: Exploiting Non-Uniform Memory Access Patterns

Ravishankar Rao, Justin Wenck,
Diana Franklin†, Rajeevan Amirtharajah and Venkatesh Akella
University of California, Davis
California Polytechnic State University, San Luis Obispo†

Abstract

On chip caches in modern processors account for a sizable fraction of the dynamic and leakage power. Much of this power is wasted, required only because the memory cells farthest from the sense amplifiers in the cache must discharge a large capacitance on the bitlines. We reduce this capacitance by segmenting the memory cells along the bitlines, and turning off the segmenters to reduce the overall bitline capacitance.

The success of this cache relies on accessing segments near the sense-amps much more often than remote segments. We show that the access pattern to the first level data and instruction cache is extremely skewed. Only a small set of cache lines are accessed frequently. We exploit this non-uniform cache access pattern by mapping the frequently accessed cache lines closer to the sense amp. These lines are isolated by segmenting circuits on the bitlines and hence dissipate lesser power when accessed.

Modifications to the address decoder enable a dynamic re-mapping of cache lines to segments. In this paper, we explore the design-space of segmenting the level one data and instruction caches. Instruction and data caches show potential power savings of 10% and 6% respectively on the subset of benchmarks simulated.

1 Introduction

A fundamental shift is occurring in microprocessor design. The unending quest for performance has led to an unquenchable thirst for power. The focus is shifting from pure performance to power-efficient performance. As this shift occurs, architects take a new look at existing structures. We take another look at the design of a cache.

Caches have traditionally been designed such that each element in the cache is accessed in the same amount of time, requiring the same amount of power. In reality, all elements within the cache are not accessed equally, so a cache should be designed to exploit the patterns that emerge in an application. In essence, this is merely an extension of the original idea of caches. The basic hypothesis is that a subset of the memory has locality, and that a cache can exploit this locality. Temporal and spatial locality are exploited by storing recently used data items, and data in the successive memory locations, in the

cache. Once an item is placed in the cache, however, it becomes equal to all other elements in the cache. This assumes there is no locality within the cache, an assumption that wastes an opportunity for optimization. We propose designing the cache such that the position within the cache determines the power consumption for accessing that set. This offers the opportunity to partition the cache at the circuit level into different power domains and explore micro-architectural techniques to remap the memory accesses into the appropriate power domain to save power. In order to implement a low-overhead segmented bitline cache, we exploit two key characteristics in the design of modern caches. We first observe that caches require more power on the bitlines because every element must drive the signal along the whole length of the bitline, irrespective of its physical location in the cache. We exploit this by placing segmenters on the bitline, requiring an element to drive only the distance from its segment to the output, resulting in a lower power to attain the same performance.

In this paper, we present a segmented bit line cache implementation that minimizes the overhead of segmenting. We develop a power model of the segmented bitline cache using CACTI [8] and HSPICE. We then describe and evaluate two methods to dynamically map sets to bitline segments of the cache to take advantage of different power domains based on the access pattern. Finally, we evaluate these methods in the context of level one instruction cache and data caches on a subset of integer and floating point SPEC2000 benchmarks.

The rest of the paper is organized as follows. We begin by describing related work in power efficient cache design in Section 2. We provide empirical evidence for the non-uniform access patterns in Section 3, which provides motivation for the work. Section 4 describes the implementation of our segmented bit line cache, followed by the clustering and mapping algorithms in Section 5. Evaluation methodology and results are presented in Section 6. Finally, we outline our future work and conclusions in Section 7.

2 Related Work

In this section we discuss several related research projects. In particular, two important areas of research are related to this work: circuit level optimizations of the bitlines for power savings and architectural techniques for power efficient caches.

At the circuit level, bitline isolation has been extensively studied. Kamble and Ghose [4] propose a local bitline for segments which are then connected to a common line across isolating switches. Higher metal layers are used to implement the common bitline. Since these layers have a lower capacitance, it reduces the overall bitline capacitance and hence saves dynamic power.

Yang and Kim propose a low power SRAM using a hierarchical bitline and local sense amplifiers (HBLSA-SRAM) [9]. The conventional SRAM bitline is divided into a several sub-bitlines each with its local sense amplifier. The sub-bitlines are connect to the read/write/pre-charge circuits through a common bit line. The HBLSA-SRAM reduces the write power in bitlines without noise

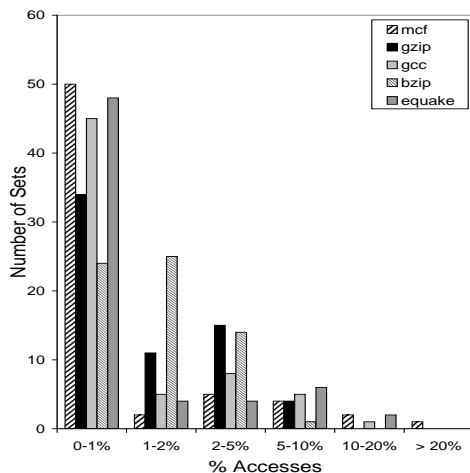


Figure 1: Histogram: L1 Instruction Cache

degradation by applying a low swing signal to the bit line and a full swing signal to the sub-bitline. Yang and Falsafi [10] note that significant energy is wasted in statically pulling up the bitlines in all cache sub-arrays. They show the potential leakage power savings in not pre-charging the unaccessed sub-arrays. The authors study the energy and performance trade-offs of bitline isolation, and propose prediction techniques to exploit its full potential.

3 Motivation

In order for us to gain power savings from our segmented bitline cache, the frequently accessed sets must be mapped to the lower-powered segments. Non-uniform cache access patterns in caches are critical to the success of this design. This section provides preliminary data which demonstrate these patterns.

We simulated an out-of-order processor using SimpleScalar [3]. The cache parameters were taken from the Itanium L1 cache [2]. The instruction and data caches are 4-way set associative, 16KB caches with a 64B set size. A subset of the SPEC2000 benchmark suite was run to completion to generate detailed access distribution data.

Figures 1 and 2 give histograms showing the number of sets and the frequency at which they were accessed for L1 instruction and data caches. There were a total of 64 sets. A uniform access pattern would have resulted in about 1.6% of access to each set.

We can see that for the L1 Instruction cache shown in Figure 1, *mcf*, *gcc*, and *equake* have 1-3 cache sets that account for at least 10% of the accesses each. For *mcf*, about 78% of the sets are accessed less than 1% of the time. These applications are good candidates for our approach. For some applications, like *bzip*, most cache sets are accessed at the mean access rate. They will be

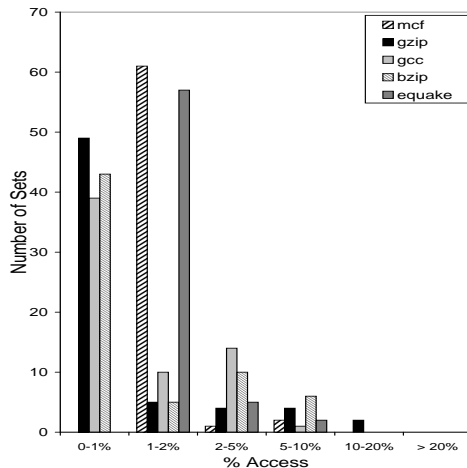


Figure 2: Histogram: L1 Data Cache

poor candidates.

In the data cache shown in Figure 2, the accesses are not quite as skewed. Only one application has sets taking more than 10% of the total accesses. In addition, two applications, *mcf* and *equake*, have the bulk of their accesses near the mean rate.

4 Segmented Bitlines

Data and tag arrays in caches are typically partitioned into banks and further divided into blocks and sub-blocks. A detailed evaluation of various configurations can be found in [1]. The different partitions give different power-performance trade-offs. In this paper, since we primarily explore level 1 caches, we use the lowest access time configuration as given by CACTI [8]. We then add segmenters to the bitline and study two, four and eight segment bitline. Although this approach adds delay to the critical path of the bitline segment furthest from the sense amplifiers, it reduces the length of the bitline for the nearest cache rows thereby reducing the capacitance which has to be discharged, and hence the bitline power.

Cache Design and Operation In a standard SRAM cache a physical bitline is one continuous wire segment that connects many SRAM cells to the sense amplifier (SA), precharge and write circuitry. Every time the bitline voltage levels need to be changed, either for a read or a write, the entire bitline must be charged or discharged. As CMOS processes continue to scale down, the parasitic interconnect capacitance begins to dominate overall cache energy. Bitlines could consume up to 50% of the total cache power [4]. When accessing an SRAM cell that is physically close to the read/write/precharge circuitry, energy is wasted

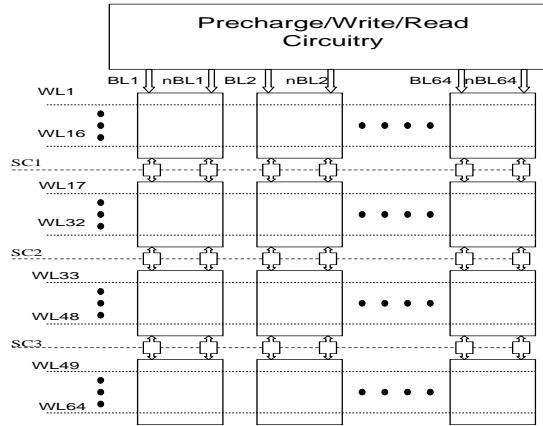


Figure 3: Segmented Bitlines

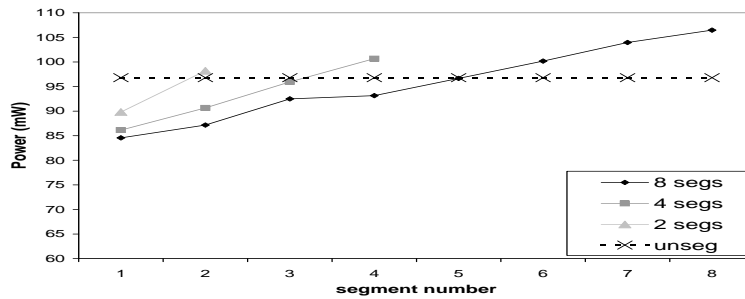


Figure 4: Cache Power for accessing different segments

by charging/discharging the other end of the bitline. To prevent this waste of energy we use a segmented bitline shown in Figure 3. BL is the bitline and nBL is its complement. WL refers to a wordline. Each box has 16 SRAM cells connected between the bitline and its complement. The block diagram shows the bitline broken into 4 segments. We define a segment to be a portion of the bitline that can be isolated from any other bitline segment. The bitline can easily be divided into 2, 8, or any other combination of segments. Each segment is separated by a segmenter, which consists of a full CMOS transmission gate. Each transmission gate has a control signal (SC), which is operated each cycle according to the address of the data being accessed. When the clock is low all segmenters are turned on to enable the precharge of the entire bitline. Although it would take less power to pre-charge only part of the bitline up to the segment being accessed, in high performance caches, pre-charging overlaps address decoding and hence the segment address may not be decoded during pre-charge.

Power Model The power model was developed using of CACTI and HSPICE. The only difference between a conventional cache and a segmented bitline cache are the additional segmenters along the bitline. Hence, the power consumption of the rest of the cache is assumed to be as given by CACTI. A column with 64 SRAM cells was simulated in HSPICE using TSMC 0.18 μ m technology to obtain the bitline power dissipation. The bitlines with two, four and eight segments were simulated to account for the delay and power consumption of the segmenters. Figure 4 plots the total cache power dissipation for accessing the different segments for a two, four and eight segmented bitline cache at 100MHz and 1.7V voltage. As seen from the figure, for the two segment case, the power dissipated to access the second segment is slightly higher than the unsegmented cache power. Similarly, in the four segment case, the third and fourth segments require higher power. Thus, our gains will be constrained by the percentage of accesses to the higher-power segments.

5 Clustering and Mapping

In a segmented bitline cache, the cache is divided into two, four or eight bitline segments. We define clustering as determining which cache sets can be partitioned together into a cluster. These clusters are then mapped to the bitline segments. Clustering and mapping introduce several challenges.

First, we do not have perfect knowledge of which sets will be accessed. Second, even if we had perfect knowledge, it would not necessarily be feasible to allow arbitrary permutation of sets to clusters. This would require a more complex logic to decode a set in the cache, increasing access times eroding some of our gains due to the shorter bitlines. We need a balance between flexibility in which cache lines may be clustered together and the speed of finding elements in the cache. Finally, access patterns change with time requiring re-mapping at regular intervals. Performing a re-mapping can be an extremely high-overhead operation. Either the sets that need to be re-mapped must be copied to their new locations, or those sets must be flushed from the cache. Swapping cache lines can be very expensive in terms of power consumption, and flushing the sets from the cache would result in an increased miss-rate. In this paper, we propose to limit the re-mapping to context switches. We assume that during a context switch, much of the state in the cache is lost, so invalidating the cache will incur a negligible performance penalty.

Clustering can be done either statically or dynamically. With static clustering, a single clustering is used across all applications and throughout the run-time of the application. Alternately, the clustering could be changed at run time, which is called dynamic clustering.

The simplest implementation of static clustering is to cluster the cache lines based on most significant address bits. With 64 cache lines, for instance, 6 bits are required to determine the row address. Now, a subset of these bits determine the segments and the others the line in the segment. For example, with 4 segments 2 MSBs determine the segment and 4 bits determine which

line within the segment is being accessed. A configuration register stores the mapping of the clusters to the segments. The address bits control the mux (in this case, two 4:1 muxes are required.), which selects the new segment address. The only increase in the critical path of the decoder is the multiplexer delay.

Once we have a clustering scheme, we need to map the cluster into bitline segments. The basic idea is to map the most frequently accessed clusters into the lowest power segments. We associate a counter with every cluster and increment it each time the cluster is accessed. We study two versions of mapping: static mapping and dynamic re-mapping. In static mapping, a single mapping of clusters to segments, obtained by profiling, is used throughout the application.

The dynamic re-mapping uses the access counters to re-map at regular intervals. Two versions of dynamic re-mapping are explored in this paper. In the first version, the segment access information of only the previous interval is used for re-mapping. We refer to this case as 'dynamic counter flush' or **dcf** because the counters used for re-mapping are flushed each time a re-mapping occurs. In this second case called 'dynamic no counter flush' or **dncf**, the cumulative access counts to the cluster from the beginning of the application to the current interval is used to decide re-mapping.

6 Results

Simulations were run with SimpleScalar. Counters were added to generate cache line access statistics and re-mapping was implemented on every 1 million cycles. The experiments were run on a subset of the SPEC2000 benchmarks, drawn from both the floating-point and integer suites, using a 16KB 4-way set associative cache for integer and data caches. We used SimPoint3.0 [7] to reduce the simulation time. SimPoint provides representative intervals and weights for each application. Multiple standard simulation points each 100 million instructions long were simulated. For each benchmark, all the intervals were simulated, and the results were weighted using the interval weights given by SimPoint. The weighted percentage of access to each segment was then used as an activity factor to scale the power of the corresponding segment.

We perform a series of experiments to evaluate the benefits of these techniques. The graphs show the total cache power savings relative to an unsegmented cache. First, we compare static mapping based on a profile and dynamic re-mapping, with that of a dynamic clustering. We then vary the number of segments from 1 to 8 segments. The operation of a segmented bitline cache at higher frequency is discussed.

6.1 Static vs Dynamic Re-mapping

Figures 5 and 6 compare three choices of mapping: static, dynamic with cumulative counters, and dynamic with counters that reset each interval. We present the results for eight segments. and compare with that of a best case dynamic clustering labeled as oracle.

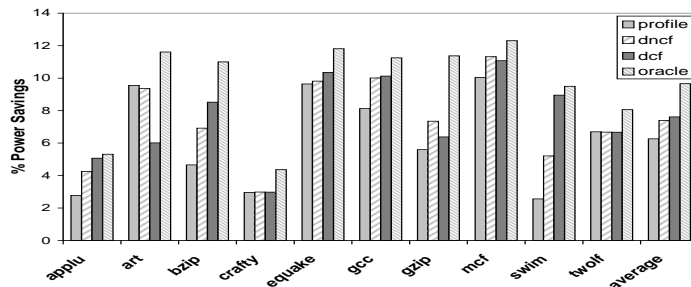


Figure 5: Static vs. Dynamic: Instruction Cache

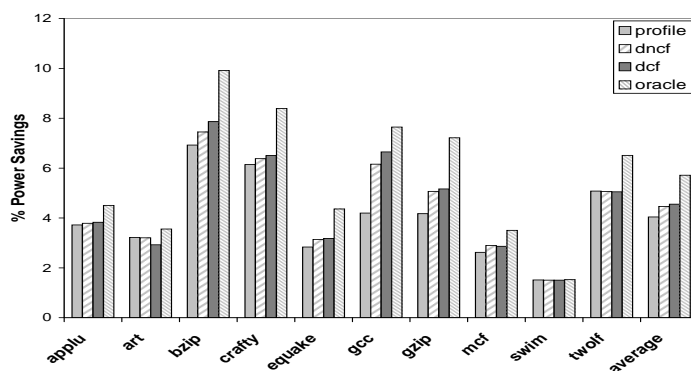


Figure 6: Static vs. Dynamic: Data Cache

For the instruction cache, the average power savings from a static mapping is 6.3% while the two dynamic re-mapping techniques provide approximately 7.6% average power savings. The dynamic clustering indicates that there is a potential to save up to 10% power on average. For some applications like *applu*, the dynamic techniques achieve close to maximum power savings, while others like *gzip* indicate that the potential is much more than what is obtained by the current dynamic re-mapping techniques. Also, note that maximum power savings are obtained for applications like *mcf* and *gcc* for which access patterns are most-skewed. The data caches have less power savings. Static mappings provide about 4% savings, whereas dynamic re-mapping performs slightly better at 4.5% power savings. The dynamic clustering results indicate a potential power savings of about 5.7% on average. This is consistent with the more uniform access distribution we noted earlier for data caches. Again, as noted earlier, we see a direct relation between power savings and non-uniformity of the access distribution. Applications like *mcf* and *equake* give least power savings because most of their cache sets are accessed at about the mean rate.

6.2 Number of Segments

The number of segments is an interesting design trade-off. With more segments, you can save more from the low-power segment, but you lose more from the high-power segment. We vary the number of segments from 2 through 8. Since the

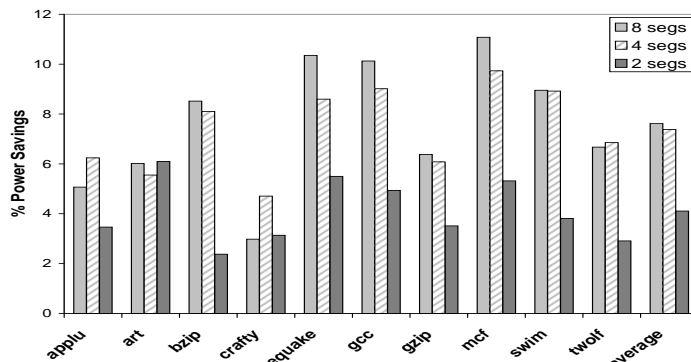


Figure 7: Number of Segments: Instruction Cache

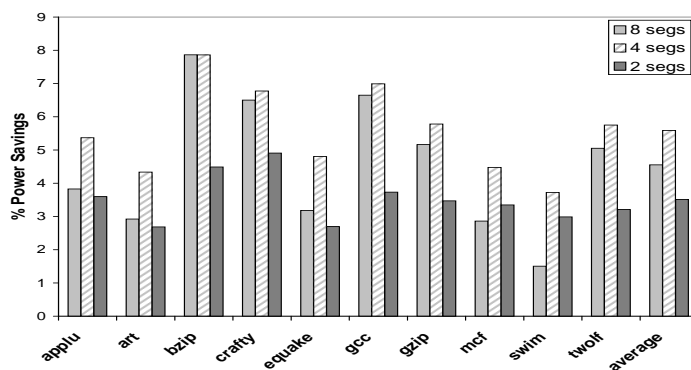


Figure 8: Number of Segments: Data Cache

dynamic remapping with counter resetting performs better than static mapping and does not require prior knowledge, we present only the **dcf** results. Figures 7 and 8 show the results for the instruction cache and the data cache respectively.

For instruction caches, increasing the number of segments increases the power savings. While eight segments gives about 7.6% power savings on average, the savings reduce slightly to 7.4% and 4.1% for 4 and 2 segments respectively. This is because instruction caches showed greater non-uniformity in their access patterns, allowing them to greatly benefit from the extra segments. The data caches show a good inflection point giving best power savings at four segments. A more uniform access pattern of the data cache leads to more accesses to the high-power segments, decreasing power savings for eight segments.

6.3 Increasing Frequency

The results presented in the previous sections are at 100 MHz where the segmenter delay was small compared to the clock cycle. As the frequency is increased, the voltage to which the bitline (or its complement) discharges is reduced. Adding segmenters to this bitline, further increases the bitline delay,

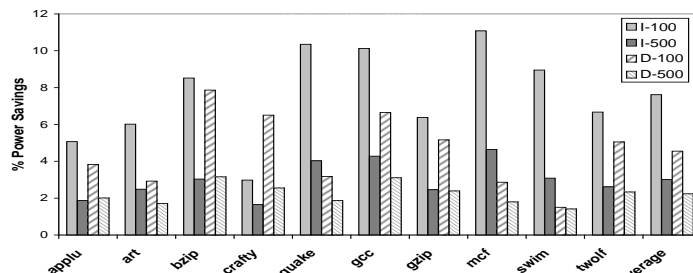


Figure 9: Increased frequency: Both Caches

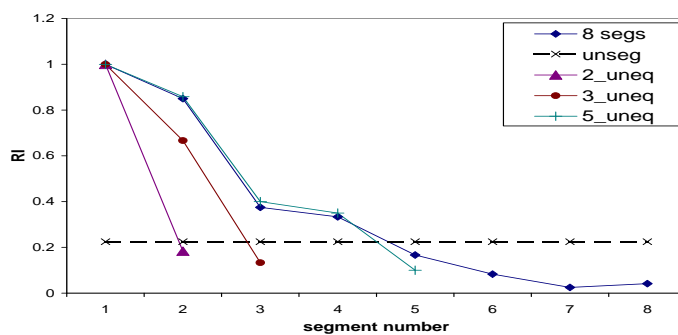


Figure 10: Cache Reliability

thereby reducing the discharge voltage of segments even more.

This reduces the power savings obtained by segmentation because the power dissipated on the bitlines is directly proportional to the voltage to which the bitline discharges. Its effect on the applications is shown in Figure 9 for 8 segments for the **dcf**.

The decrease in bitline discharge also potentially reduces the reliability of segments further away from the sense-amp circuitry since reduced swing increases the likelihood of sense-amp failure. We assume that reliability is inversely proportional to the discharge voltage of the bitline.

So using our remapping techniques, if most accesses are limited to these segments, an overall higher reliability is possible. Even so, in the eight segment case, half the cache has a lower reliability than an unsegmented bitline. To reduce the low reliable cache area we simulated unequal segments in our bitline. Three cases of unequal segments are considered in this paper: First, a two segment bitline, with eight rows in the first segment and 56 in the second segment. Second, a three segment bitline, with eight rows in the first segment, 16 in the second and 40 rows in the third segment. Third, a five segment bitline, with eight rows each in the first four segments, and a fifth segment with the remaining 32 rows.

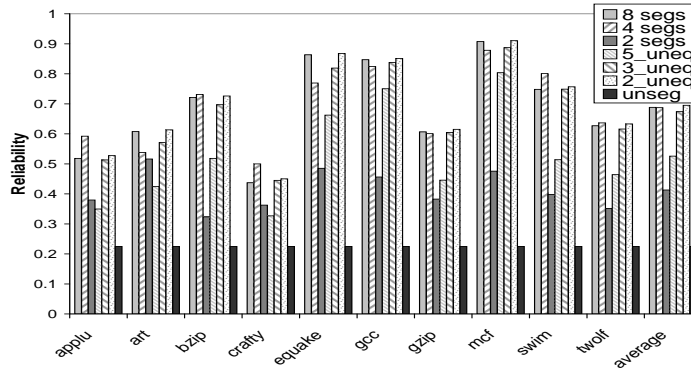


Figure 11: Application Reliability

We quantify reliability in terms of 'Reliability Indicator' (RI) which is inversely proportional to the final voltage to which the bitline discharges. A typical sense-amp requires a differential voltage of 0.5V [6] between the bitline and its complement. The reliability of a segment is thus inversely proportional to the lowest voltage on the bitline. For instance, if the bitline discharges to 1.2 V (with $V_{dd} = 1.7V$), the RI is 0, and if the bitline discharge is to 0V, the RI indicator is 1. Figure 10 compares the reliability of unequal segment configurations with that of an unsegmented cache and an eight segment bitline cache. First, we observe that an unsegmented cache itself has a reduced RI at higher frequency. The eight segment bitline has segments with very low RI. This is improved with unequal segments, where only one segment has a RI lower than an unsegmented bitline. We compute the reliability of an application as the sum of, product of segment RI and its percentage access for each configuration. The reliabilities thus obtained for all segment configurations is plotted in Figure 11 for the instruction cache with dcf mapping. It is clear from the figure that all cache configurations provide much better reliabilities than an unsegmented cache. Clearly, this is due to the effectiveness of the re-mapping techniques. Similar results were obtained for the data cache.

The power savings due to these unequal segment configurations is compared to that of eight and four segment bitlines, for the **dcf** mapping for an instruction cache, in Figure 12. As seen from the figure in both the data and instruction caches, for most applications, power savings are between those of an eight and a four segment bitline. Similar results were obtained for the data cache.

7 Conclusions

In this paper, we present architectural techniques that are combined with simple circuit modifications of a cache. We exploit the non-uniform access patterns of

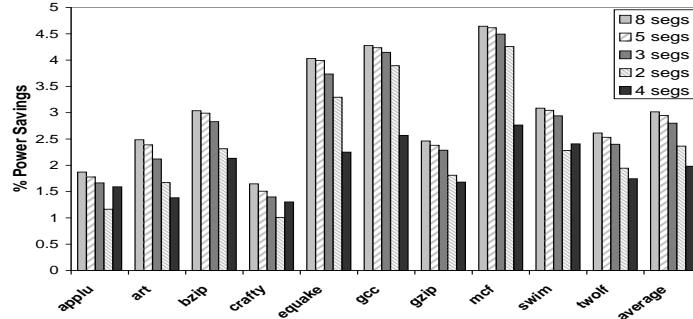


Figure 12: Unequal Segments: Instruction Cache

different applications to obtain dynamic power savings. We explore partitioning the cache lines into clusters and mapping them to segments dissipating different power. A re-mapping occurs on a context switch to limit the performance and power overhead. Instead, the re-mapping could be guided by the changing program phases. SimPoint has tools to detect phase changes in applications at runtime [5]. By integrating these tools clusters could be re-mapped only during the phase changes.

References

- [1] B. S. Amrutur and M. A. Horowitz. Speed and power scaling of srams. *IEEE Journal of Solid-State Circuits*, 35(2):175–185, February 2000.
- [2] D. Bradley, P. Mahoney, and B. Stackhouse. The 16kb single-cycle read access cache on a next-generation 64b titanium microprocessor. In *International Solid State Circuits Conference*, 2002.
- [3] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-1997-1342, University of Wisconsin, Madison, June 1997.
- [4] K. Ghose and M. B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *International Symposium on Low Power Electronics and Design*, pages 70–75, 1999.
- [5] J. Lau, S. Schoenmackers, and B. Calder. Transition phase classification and prediction. In *11th International Symposium on High Performance Computer Architecture*, Feb 2005.
- [6] J. M. Rabaey. Digital integrated circuits: A design perspective. 1996.
- [7] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct 2002.
- [8] S. J. Wilton and N. P. Jouppi. Cacti: An enhanced cache access and cycle time model. In *IEEE Journal of Solid-State Circuits*, May 1996.
- [9] B.-D. Yang and L.-S. Kim. A low-power sram using hierarchical bit line and local sense amplifiers. In *IEEE Journal of Solid-State Circuits*, June 2005.
- [10] S.-H. Yang and B. Falsafi. Near-optimal precharging in high-performance nanoscale cmos caches. In *36th International Symposium on Microarchitecture*, 2003.