Impact of Partitioning Cache Schemes on the Cache Hierarchy of SMT processors

Samantha Kenyon, Sonia López,
Department of Computer Engineering
Rochester Institute of Technology
Rochester, New York
srk1037@rit.edu

Julio Sahuquillo
Dep. de Informática de Sistemas y Computadores
Universidad Politècnica de València, Spain
Camino de Vera s/n Valencia
jsahuqui@disca.upv.es

***Abstract*—Power consumption is becoming an increasingly important component of processor design. As technology node shrinks both static and dynamic power become more relevant. This is particularly critical for the cache hierarchy. Previous implementations mainly focus on reducing only one kind of power in the cache, either static or dynamic. However, for a more robust approach that will remain relevant as technology continues to shrink, both aspects of power need to be addressed.**

Recent processors, e.g. Intel Core or IBM Power8, implement simultaneous multithreading (SMT) cores to hide high memory latencies. In these systems, the dynamic energy in the L1 cache is even more stressed since this cache level is shared by several threads running on the same core. This paper proposes and evaluates the use of phase adaptive caches in all structures of a 3-level cache hierarchy of a SMT cores. Compared to the use of conventional caches, our work results on significant dynamic and leakage energy savings with scarce performance impact.

I. Introduction

As manufacturing technology is improved and transistors shrink, the number of transistors on a chip has increased exponentially. Power consumption has become a major factor in microprocessor design. Performance improvements in current designs are limited by their power consumption and therefore improvements in speed have diminished over the years.

A large percentage of these transistors are used to increase the size of the cache hierarchy. This directly results in the multiple cache structures consuming a large percentage of the overall power budget in the microprocessor. The cache hierarchy can contain around 35%-40% of the total number of transistors on a typical processor [6], making power consumption within the cache critical.

Previous efforts ([1] [4] [5] [8]) have focused on reducing either static power or dynamic power consumption, applying techniques that modify the architecture, the circuit design, or the actual transistor manufacturing. Other research (e.g. [13]) have either focused on increasing performance or reducing energy due to the inherent trade-offs between both of them. Typically, an increase in performance will correspond to an increase in energy consumption. Likewise, decreasing power consumption will correlate to a decrease in performance. This work combines distinct techniques to reduce both static and dynamic power while only slightly affecting the performance.

The drowsy phase-adaptive cache design was first proposed in [6]; however, the original cache design was implemented only in the second level cache of a single threaded processor. In this work, this cache design is applied to all cache structures of a three-level cache hierarchy in a SMT architecture. We evaluate and analyze the effect of combined data access patterns and multithreading on both performance and power consumption when this design applies to the cache hierarchy in its entirety. Our results show up to 80% total energy savings with performance reduction under 5% of the baseline design.

The remainder of this work is organized as follows. Section II presents the drowsy adaptive cache proposed in this work for SMT cores. Section III describes the methodology used to run the experiments and obtain performance and energy metrics. Section IV evaluates the approaches and analyzes the obtained results. Section V presents the related work. Finally, in Section VI, some concluding remarks are drawn.

II. Drowsy Phase Adaptive Cache

The drowsy phase-adaptive cache presented in this work combines the phase-adaptive cache and the drowsy cache. The phase-adaptive cache attempts to save dynamic power while the drowsy approach attempts to save leakage power, both of them while incurring a modest performance penalty.

A. Drowsy Phase Adaptive Cache

In this work we define a cache structure with two partitions. The A partition remains in an active state, while the B partition is placed in a drowsy state. The partitions are phase-adaptive and the partitions' configurations are determined based on cost functions. For our design, L1, L2, and L3 caches implement an associativity degree of 4-, 8-, and 16-way, respectively. The total number of ways in the cache is kept constant however the different configurations allow for each partition to have a varying number of ways. For example, the L1 4-way cache has three possible configurations: 1/3, 2/2 or 4/0 ways in A/B partitions respectively.

The access protocol for this cache design takes advantage of this dual partition structure. The cache first looks up the requested data in the A partition. MRU (Most Recently Used) counters are used to control when the cache changes between different configurations. If the data is found, the corresponding MRU counter is updated and the data is delivered to the processor. If the data is not found, then the B partition is searched and on a hit, the MRU counter is updated and the block swapped into the A partition. On a L1 cache miss, the

block is looked up on the L2 cache and if it is found, then the block is brought up to the A partition. The victim block is placed into the B partition, and a block from this partition is evicted into the next cache level. After that, the MRU states are updated to reflect these changes.

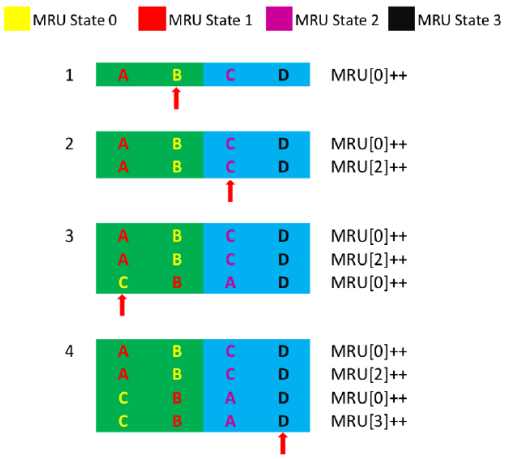

Fig. 1: Example access pattern with MRU counters and states.

Figure 1 shows an example access pattern with updates to the MRU counters and states for a 4-way set-associate cache. A different color has been associated to each MRU state. MRU state 3 refers to the least recently used (LRU) block. First, block B is accessed and its MRU counter is increased. Since the requested data is located in the A partition (green area), this is all that is necessary. After that, block C is accessed and the MRU state counter is incremented; however, block C is in the B partition, therefore C needs to be swapped into partition A with block A (the LRU one). From there, C is accessed again and its MRU state (MRU 0) counter is again increased. Finally, D is accessed and the counter for MRU state 3 is increased.

The phase adaptive portion of this design works as that presented in [6]. The cache design adapts to the phase by dynamically changing the number of ways in the A and B partitions, according to cost functions, explained below, that use the MRU counters as inputs. In this design, the costs functions are re-evaluated every 15K committed instructions. This provides small enough granularity without reducing performance.To ensure that the design is not constantly changing its phase, a warm up period of one phase is given in between each configuration change. When the data is found in the A partition, the logic that gives access to the ways in the B partition remains unaffected, thus saving dynamic power.

In the drowsy phase-adaptive cache, the B partition is placed in a drowsy state. This is done by dropping the voltage from 0.9V to 0.7V in the B partition. This voltage level was determined to be the most optimal by Brendan Fitzgerald *et al.* in [6]. Whenever the B partition is accessed it takes one cycle to raise the voltage back to 0.9V and read the data, meaning that each time the B partition is accessed performance is reduced. Since 92% of cache accesses are to MRU 0 and 98% of cache access are to MRU0 or MRU1 this does not incur a large performance hit [9]. The majority of accesses are to partition A, therefore the majority of the time there will be no performance hit from accessing the drowsy cache. This allows for a large amount of leakage savings when the B partition is large, with a small performance hit.

This design takes advantage of the energy saving techniques used by the accounting cache and phase-adaptive cache to save dynamic power. It also saves leakage power by placing the B partition in a drowsy state. This design, however, does affect performance. One additional cycle is needed to access data in the B partition and swap blocks between partitions. There is also an additional one cycle latency to bring the B partition voltage back up to an active state to access data in the B partition. However this design takes advantage of cache locality and access patterns. When a majority of the data in the cache is accessed multiple times, the energy savings is high and the performance penalties are small making this a good cache design for many applications.

1) Cost Functions: The ideal cache configuration in this phase adaptive cache is determined using different cost functions. The generic cost function shown below was derived previously by Fitzgerald [6]. This function is based off of the work presented by Dropsho et al. [4]. The cost is based either on delay or energy. The general cost function is shown in equation 1.

$$Cost = hits_A * cost_A + hits_B * cost_B + misses * cost_{misses} \quad (1)$$

In this equation $hitsA$ and $hitsB$ are the number of hits in the A and B partitions. These are calculated by summing the MRU counters for the ways that correspond to each partition. The variable $misses$ is the number of misses in the cache level and $cost_{misses}$ is the cost of accessing the next cache level. The cost variables represent the cost to either power, latency, or both, of accessing the partition or incurring a miss in the partition. Since this work is primarily looking at energy equation 2 represents the energy cost.

$$\begin{aligned} EnergyCost_i = {} & hits_A * DynamicEnergy_A + hits_B \\ & * DynamicEnergy_B + misses * cost_{misses} \\ & + LeakageEnergy_A + LeakageEnergy_B \\ & + swaps * (DynamicEnergy_A \\ & + DynamicEnergy_B) \end{aligned} \quad (2)$$

The delay cost function defined in [6] is shown below in equation 3.

$$\begin{aligned} DelayCost = {} & hits_A * Latency_A + hits_B \\ & * Latency_B + misses * Latency_{misses} \end{aligned} \quad (3)$$

The total cost can be described as either the total energy cost or the total delay cost. These equations have been derived for the L1, L2, and L3 cache configurations. In the work presented int his paper, all the ways in the A and B partitions of each cache level are shared among the threads with no restrictions. The MRU counters collect the number of accesses to the different blocks without tracking the thread ID for each access. This approach simplifies the data collection and logic necessary for the phase adaptive implementation with in [4] was estimated to be below 5,000 gates.

III. Methodology

To perform the proposed work multiple tools were used. SPICE was used to determine the drowsy voltage used in this work. CACTI was used to perform hardware simulations for a 32nm technology node to determine the appropriate latency and power values for each configuration's cost function described bellow. Finally, detailed performance and timing results were obtained using Multi2sim, which was modified to implement the drowsy phase-adaptive cache hierarchy in all three levels of our architecture.

A. CACTI

As shown in the cost functions in Section II-A1, latency and power numbers are required to be able to evaluate each of the configuration options. These hardware parameters were determined using CACTI 6.5. For each cache level Uniformed Cache Access (UCA) was used ensuring that accessed to each block of the cache would be the same. Also, for each cache there are two exclusive read ports and two exclusive write ports. Sequential access mode is used, meaning that the tag array is accessed first and then the data array. This reduces energy consumption in comparison to other access modes. The drowsy simulations require the voltage used in the CACTI model to be set to 0.7 volts instead of the normal 0.9 volts by modifying the CACTI source and recompiling for all drowsy parameters.

1) Latency: CACTI only allows for associativities in powers of two. Linear interpolation has been applied to determine the latency values for configurations where the B partition is not a power of two. All of these simulations assume a 3 GHz processor. These simulations are done for both the Drowsy case, where the voltage is reduced, and the normal case for all cache levels. From there, this data is plotted and linear interpolation is used to determine an equation for the latency in relation to the associativity. These latencies are used in the cost equations discussed above, to determine overall performance of the system. Table I shows the latency values for the different partitions of the studied L1, L2, and L3 caches. The latency values used for the baseline case, when all caches are held un-partitioned, are the values shown in I when the B partition has 0 ways.

	L1 partition		L2 partition		L3 partition	
Name	A	B	A	B	A	B
C0	1-way 2-cycle	3-way 3-cycle	1-way 3-cycle	7-way 5-cycle	1-way 7-cycle	15-way 14-cycle
C1	2-way 3-cycle	2-way 3-cycle	2-way 4-cycle	6-way 4-cycle	2-way 9-cycle	14-way 14-cycle
C2	4-way 4-cycle	0-way 0-cycle	4-way 5-cycle	4-way 4-cycle	4-way 12-cycle	12-way 13-cycle
C3	– –	– –	8-way 5-cycle	0-way 0-cycle	8-way 14-cycle	8-way 12-cycle
C4	– –	– –	– –	– –	16-way 16-cycle	0-way 0-cycle

TABLE I: Configurations and latencies of the studied caches. NA: not applicable.

2) Energy and Power: Just like with the latency values, linear interpolation is used to find the energy and power values for cache configurations that are not a power of two. Both leakage power and dynamic power are found for each level of the cache, for each of the possible configurations that are powers of two. This is done for both the Drowsy case, where the voltage is reduced, and the normal case.

B. Multi2sim

A modified version of Multi2sim 3.2 [14] was used for the work described here. These modifications required adding MRU counters and cost functions as described in [6] in order to determine the next cache configuration based on the statistics for the just finished phase. From there, it sets the latency for the B partition and records the energy usage of the previous phase. The configuration costs are determined using Equation 2 and Equation 3. The next configuration is determined by comparing the energy usage from all of the possible configurations of the just finished phase. The counters and variables are reset for the next execution interval.

1) Simulation Configurations: To evaluate the proposal we used the cache hierarchy and processor parameters shown in Table II.

Cache hierarchy	
L1-Data Cache	32KB, 4 way, 4 cycle
L2 Cache	256KB, 8 way, 5 cycle
L3 Cache	8MB, 16 way, 16 cycle
All caches	LRU, 64B line, 2 Read ports, 2 Write Ports
Processor	
Fetch Queue	64 bytes
Branch Predictor	Combined, 1024 entry Biomodal, Two level 8K history table
Decode, Issue & Commit Width	4 instructions
Reorder Buffer Size	129

TABLE II: Cache-hierarchy and processor configuration

We assume that the core is deployed in a multicore processor as that shown in Figure 2, which depicts the cache hierarchy. Local caches include private L1 and L2 caches, and shared last level cache (SLLC) refers to the L3 cache, where it is important to keep low the SLLC cache interferences. The main processor and cache hierarchy parameters used in the experiments are shown in Table II.

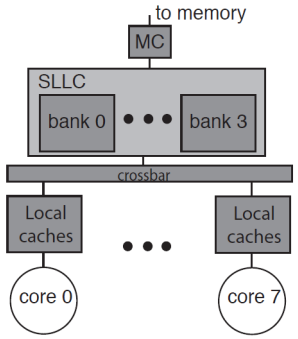

Fig. 2: Multicore Processor block diagram [2]

2) Benchmarks: The SPEC2006 benchmark suite [3] was used for all of the simulations. For the multithreaded and multicore simulations different mixes of specific benchmarks were chosen. These combinations were chosen based on the

Benchmark	IPC	$MPKI_{L1}$	$MPKI_{L2}$
mcf	0.58	102	5
hmmer	1.19	4	0.001
milc	0.66	19	9
dealII	1.6	4	0.1
lbm	0.43	57	26

TABLE III: Spec2006 benchmark characteristics

Configuration name	Description
Baseline	The cache configuration was held unpartitioned
Phase	The cache configuration is determined on the MRU statistics, energy and leakage.
Drowsy	The cache configuration is determined on the MRU statistics, energy and leakage with the B partition being put into the drowsy state.
PhaseED	The cache configuration is determined on the MRU statistics and the energy-delay product.
DrowsyED	The cache configuration is determined on the MRU statistics, and the energy-delay product with the B partition being put into the drowsy state .

TABLE IV: Compared Cache Schemes and Descriptions

performance (i.e. IPC) of each of the different benchmarks shown in Table III. The combinations of these benchmarks simulate both mixes of high performance benchmarks as well as benchmarks with low IPC. Mixes are also created considering the *Misses Per Kilo Instruction* (MPKI) for both cache level 1 ($MPKI_{L1}$) and level 2 ($MPKI_{L2}$). Each of these benchmarks will be run for a minimum number of cycles to obtain representative results.

IV. Results

Different variants of drowsy phase-adaptive cache design have been devised to improve the tradeoff between performance and energy consumption. Table IV shows the compared cache schemes. These caches have been compared working in SMT cores.

A. SMT Core Results

This section evaluates the studied cache schemes working in a two-threaded and in a four-threaded SMT core. For comparison purposes, we built four-benchmark mixes by replicating some two-benchmark mixes (i.e. lbm+dealII and lbm+milc). In other words, in the four-thread mixes we have the interference of the two-thread mixes plus the interference of the added benchmarks (see Table V).

Name	Thread 1	Thread 2	Thread 3	Thread 4
fourA	lbm	dealII	lbm	dealII
fourB	lbm	lbm	milc	milc

TABLE V: Experiments with four threads: combined benchmarks in *fourA* and *fourB*

1) Performance: Figures 3, 4, and 5 show the percentage of time spent in each possible configuration for the studied approaches across the different cache levels. Results vary across the different caches according to the data locality and how often each cache is accessed.

Remember that in SMT cores, the L1 cache is private to the core but shared among all the threads in the core. This means that intra-core interferences among the cache accesses of the threads sharing the core appear in the L1 cache. Due to the high data locality, most of the accesses concentrate on the MRU block when running a given application in isolated execution [9]. However, because of interferences this percentage is expected to decrease in SMT processors. This is the reason why in the results for the dual-threaded architecture, some applications spent a significant amount of time in C1 instead of concentrating almost all the time in C0 as it can be appreciated in Figure 3. Of course, interferences increase with the number of threads in the core, thus lowering the percentage of time in the C0 configuration. This can be appreciated in the results for the four-threaded processor.

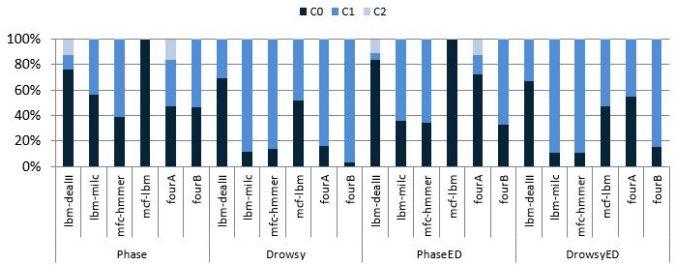

Fig. 3: L1 configuration distributions

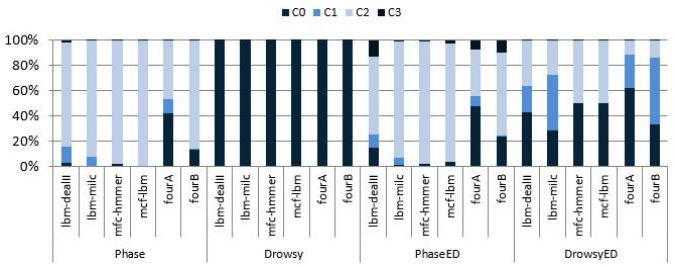

Fig. 4: L2 configuration distributions

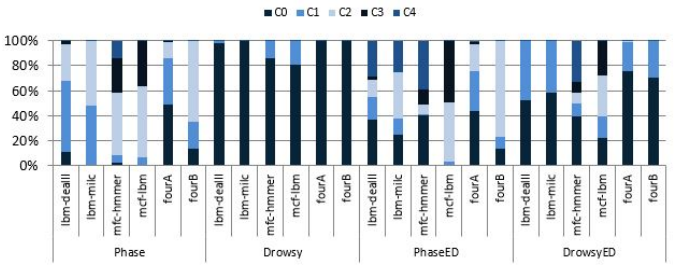

Fig. 5: L3 Configuration distributions

Regarding the L2 cache, it can be appreciated (see Figure 4) that, compared to the L1 behavior, there is a high configuration variation for the *Phase* and *PhaseED* caches. The main reason is due to the fact that L1 caches capture much of the memory requests issued by the processor, thus L1 caches hide most of the data locality to L2 caches. Consequently, unlike what happens in first-level caches, most of the accesses to the L2 caches do not hit the MRU block. The *Drowsy* cache spends most of its time in C0 to optimize for the most energy savings and the *DrowsyED* approach spends time in the first few cache

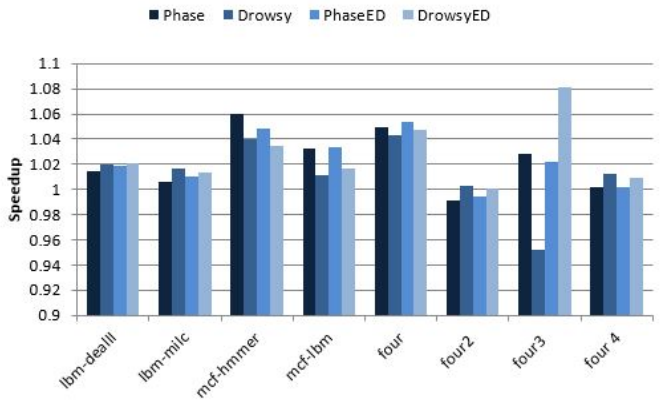

Fig. 6: Overall speedup results over the static baseline —as the harmonic mean of the individual per-thread speedups— for each one of the four schemes *Phase, Drowsy, PhaseED* and *DrowsyED*

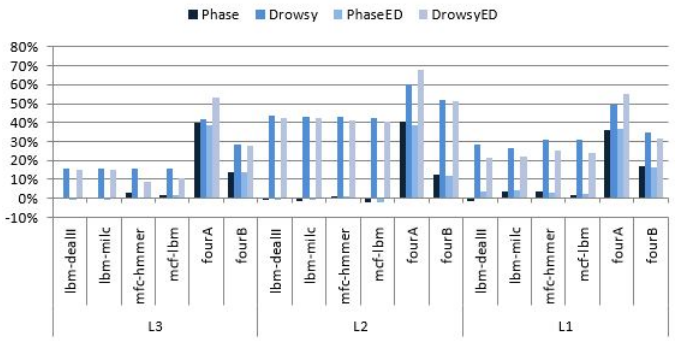

Fig. 7: Total energy savings of each cache level for the single SMT core

schemes to optimize for both energy and delay. Since the L3 cache is much less accessed than the L2 cache there is again a lot of configuration variation for the *Phase* and *PhaseED* simulations (see Figure 5). The *Drowsy* scheme spends most of its time in C0 to optimize for the most energy savings and the *DrowsyED* simulation spends time in the first few configurations to optimize for both energy and delay.

Figure 6 shows the speedup for each of the benchmarks run with a drowsy phase-adaptive approach across all the cache hierarchy with respect to the baseline machine as shown in Table I. It can be seen that the performance improves in some cases and incurs a small penalty in others. The speedup ranges from 95.2% to 108.1%. Since the cache access time varies with the size of the A partition, in some cases the access can be faster than in the baseline. In other cases, accesses to the B partition can contribute to a larger overall access latency, hence the various contributions to speedup depend on the benchmark locality. When performance is hurt, the damage stays below 5% of the baseline performance with our proposed architecture. The *Drowsy* approach shows the worst performance. Due to the energy benefits of having a large B drowsy partition, a small A partition is selected more often; which can incur in more misses in the A partition, more hits in the B partition, and hurt overall performance even when delay is taken into account in the cost function.

2) Energy and Power: The average energy savings for all the cache levels is shown in figure 7. In our experimental results, the dynamic energy savings dominate the overall energy savings. The *Drowsy* and *DrowsyED* approaches hold the most dynamic energy savings. One of the simulations has negative energy savings for the *Phase* and *PhaseED* cases. This can be explained by looking at the L1 miss-cost (i.e. the cost of an L2 access). This cost in the *Phase* and *PhaseED* cases is higher for all phase configurations.This allows for the possibility of higher energy consumption in some cache schemes than that of the baseline.

The leakage energy savings dominates the overall energy savings for L2 since this cache is much less accessed than the L1 cache. In our experimental results, there is large amount of leakage energy savings for L2, ranging from 44.5% to 69.7%.The total energy savings for L3 is more balanced between both dynamic and leakage energy.

V. Related Work

Srikantaiah *et al.* propose a reconfigurable adaptive multi-level cache design called a *MorphCache* [13]. This cache dynamically changes the cache hierarchy to improve performance. From there the MorphCache either merges slices or splits them, dynamically, to change the slices and configurations available to each core. The MorphCache will merge slices when either one slice is highly-utilized while the other is under-utilized, or both are highly-utilized and shared by threads sharing the same address space. This implementation is focused on improving performance. The work proposed here differs from this because it is focused on saving power with a small performance hit. Also the MorphCache requires modifications to the interconnection network where as the proposed design does not.

Keramindas *et al.* proposes a new framework for efficient cache resizing in terms of both power and performance [7]. The cache proposed here dynamically reconfigures memory based on the behavior of the running application. In a typical resizing cache, when the cache size is decreased the discarded part of the cache is immediately deactivated. Instead, here the discarded part is kept active and they gradually deactivate. When all of these parts are deactivated the transition has officially completed. Overall this method results in cache size reduction of anywhere from 13% to 30% with a very low performance impact of 4% to 10%. This work focuses on reducing the area of a cache in a way that is efficient in terms of both energy and performance. The implementation proposed here differs from this because it focuses saving leakage and dynamic energy, by modifying the cache configuration, while keeping the cache area the same.

Agrwal *et al.* proposes a Data Retention Gated-Ground Cache, or a DRG-Cache. This cache uses gated-ground techniques to reduce leakage power consumption in cache memory systems [1]. The gated-ground technique in [10] inserts an NMOS transistor between the ground line and the SRAM cell. This allows for the supply voltage to be effectively off, substantially reducing leakage energy dissipation. This technique allows for a large reduction in leakage energy, however

it increases dynamic energy for read and write operations. Also turning off the supply voltage results in the possible of destroying data stored in the SRAM cell. Another component to this technique is determining the optimal size of this Gated-Ground transistor. Increasing the size of this transistor improves performance as well as data retention but lowers the overall power savings. The work proposed here differs from this because it is state-preserving. Also the proposed work aims to save both dynamic and leakage power whereas this work only saves leakage power while increasing dynamic energy.

Jason Nemeth *et al.* proposes a location cache that has a low power second level cache using gated-ground techniques [8]. Combining these two ideas the goal is to save on both static and dynamic power. This second level using the gated-ground techniques discussed in [10] similarly to the DRG-Cache in [1]. This implementation combines this low power structure with a location cache. This location cache is a direct-mapped cache that provides the second level cache with accurate access way location information. This additional cache runs in parallel with the L1 cache. This reduces L2 power consumption more than the typical set-associative cache. This implementation is shown to save between 2% and 43% of total power, however it has the same limitations as the DRG-Cache [1] because it uses the same gated-ground technique.

Farahani *et al.* proposes a modification to the general drowsy cache configuration [5]. This work proposes two different methods for saving leakage in a cache. In the first design cache words are placed in a drowsy mode at the end of a period of time called an update window (UW). When a word is needed only that word is brought back up to an active voltage level and the rest of the cache line remains in a drowsy state. The second approach identifies actives words and does not place them into drowsy mode at the end of a UW. This allows for a performance savings as well as leakage savings with the words that are placed in the drowsy state. The first implementation reduces leakage power by an average of 88% with an average performance loss of just 0.7%. The second implementation reduces leakage power by an average of 89% with an average performance loss of 0.5%. Although this implementation reduces leakage power significantly with a low performance penalty it does not reduce dynamic power at all.

Other approaches try to partition the cache in two independent structures, each one devoted to exploit the localities of the data it stores [11], [12]. This way tries to improve performance by making shorter the cache access time.

VI. Conclusion

This paper presented a drowsy phase-adaptive cache design that saves both static and dynamic power while having a scarce impact on performance. While exploiting the temporal locality of memory accesses and using two-partition caches these goals were met for all cache levels in a three-level cache hierarchy of a processor with SMT cores. In most cases, energy savings occur in the *Drowsy* cache approach. In these cases the applications generally spend most of their time in the C0 configuration, which is the one with the smallest A partition and largest B partition. Since these simulations have the highest energy savings, while incurring a low and acceptable performance hit, it can be concluded that for many applications the C0 configuration is optimal for the *Drowsy* cache approach.

There are many opportunities to expand on this work and investigate this design further. First, since this work only focuses SMT architectures, it would be useful to explore the affect that multicore systems have on the overall performance. In addition, limits to the available number of ways in each cache level can be set per thread or core to further explore intelligent optimization of the resources and avoid thread starvation.

Acknowledgments

This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and by FEDER funds under Grant TIN2012–38341–C04–01.

References

[1] A. Agarwal, H. Li, and K. Roy. Drg-cache: a data retention gated-ground cache for low power. In *Design Automation Conference, 2002. Proceedings. 39th*, pages 473–478, 2002.

[2] J. Albericio Latorre, P. Ibez Marin, and J. M. Llaberia Grino. *Improving the SLLC Efficiency by exploiting reuse locality and adjusting prefetch*. PhD thesis, Zaragoza, Universidad de Zaragoza, Zaragoza, Ago 2013. Presentado: 20 05 2013.

[3] S. P. E. Corporation. Spec cpu2006 benchmark suite, June 2008.

[4] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. Scottt. Integrating adaptive on-chip storage structures for reduced dynamic power. In *Parallel Architectures and Compilation Techniques, 2002. Proceedings. 2002 International Conference on*, pages 141–152, 2002.

[5] M. Farahani, F. Eslami, and A. Baniasadi. Application specific low leakage data cache for embedded processors. In *Green Computing Conference (IGCC), 2013 International*, pages 1–6, June 2013.

[6] B. Fitzgerald. Drowsy cache partitioning for reduced static and dynamic energy in the cache hierarchy, 2012. Copyright - Copyright ProQuest, UMI Dissertations Publishing 2012; Last updated - 2014-01-19; First page - n/a; M3: M.S.

[7] G. Keramidas, C. Datsios, and S. Kaxiras. A framework for efficient cache resizing. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pages 76–85, July 2012.

[8] J. Nemeth, R. Min, W.-B. Jone, and Y. Hu. Location cache design and performance analysis for chip multiprocessors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(1):104–117, Jan 2011.

[9] S. Petit, J. Sahuquillo, J. M. Such, and D. Kaeli. Exploiting temporal locality in drowsy cache policies. In *Proceedings of the 2Nd Conference on Computing Frontiers*, CF '05, pages 371–377, New York, NY, USA, 2005. ACM.

[10] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-vdd: a circuit technique to reduce leakage in deep-submicron cache memories. In *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on*, pages 90–95, 2000.

[11] J. Sahuquillo and A. Pont. The filter cache: A run-time cache management approach1. In *25th EUROMICRO '99 Conference, Informatics: Theory and Practice for the New Millenium, 8-10 September 1999, Milan, Italy*, pages 1424–1431, 1999.

[12] J. Sahuquillo and A. Pont. The split data cache in multiprocessor systems: an initial hit ratio analysis. In *Proceedings of the Seventh Euromicro Workshop on Parallel and Distributed Processing. PDP'99, University of Madeira, Funchal, Portugal, February 3-5, 1999*, pages 27–34, 1999.

[13] S. Srikantaiah, E. Kultursay, T. Zhang, M. Kandemir, M. Irwin, and Y. Xie. Morphcache: A reconfigurable adaptive multi-level cache hierarchy. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 231–242, Feb 2011.

[14] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez. Multi2sim: A simulation framework to evaluate multicore-multithreaded processors. In *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, pages 62–68, Oct 2007.