Virtual Exclusion: An Architectural Approach to Reducing Leakage Energy in Caches for Multiprocessor Systems

Mrinmoy Ghosh
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
mrinmoy@ece.gatech.edu

Hsien-Hsin S. Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
leehs@gatech.edu

Abstract

This paper proposes *Virtual Exclusion*, an architectural technique to reduce leakage energy in the L2 caches for cache-coherent multiprocessor systems. This technique leverages two previously proposed circuits techniques — *gated Vdd* and *drowsy cache*, and proposes a low cost, easily implementable scheme for cache-coherent multiprocessor systems. The Virtual Exclusion scheme saves leakage energy by keeping the data portion of repetitive cache lines *off* in the large higher level caches while still manages to maintain Multi-Level Inclusion, an essential property for an efficient implementation of conventional cache coherence protocols. By exploiting the existing state information in the snoop-based cache coherence protocol, there is almost no extra hardware overhead associated with our scheme. In our experiments, the SPLASH-2 multiprocessor benchmark suite was correctly executed under the new Virtual Exclusion policy and showed an up to 72% savings of leakage energy (46% for SMP and 35% for multicore in L2 on average) over a baseline drowsy L2 cache.

1. INTRODUCTION

Owing to the continuing downscaling of CMOS technology, the threshold voltages have become lower and the gate oxides are getting thinner, both resulting in a significant increase in leakage power. For example, a 90nm Pentium 4 processor consumes 110 W, and roughly 40% of it is leakage power[17]. Meanwhile, with technology scaling, the capacity of single-chip processors has exceeded one billion transistors. To consume such an immense amount of transistors, processor architects tend to allocate more cache space and deepen the level of cache hierarchy. While these caches constitute a major portion of a processor's real estate, they are also the least active components and dominate the leakage power among all other architectural modules.

There have been a large number of architectural and circuits techniques proposed to reduce leakage power in caches. Powell *et al.* [16, 20] shows that the leakage current can be substantially reduced by employing sleep transistors to gate off the supply voltage when the corresponding logic blocks are not in use. *Cache Decay* was subsequently proposed in [9] to exploit this circuit technique in the L1 cache by using simple counters to turn off cache lines if they are unlikely to be re-accessed. Although it mentioned implications of applying the decay scheme in large, higher level caches, it provided no further in-depth analysis, in particular, from the cache coherence standpoint, a correctness issue for implementing a multiprocessor (MP) system. One major drawback of the cache decay policy lies in the performance and power trade-offs of the extra misses induced due to the switch-off of decayed lines, which leads to additional accesses to the DRAM. This energy overhead oftentimes outweighs the leakage savings from the technique itself.

Another circuit technique for leakage reduction is using the ABC-MT-CMOS memory cell [15]. This circuit technique uses different supply and ground voltage levels to bias the transistors to increase their effective threshold voltage. It reduces leakage current dramatically while preserving transistor state in a lower supply-voltage, i.e. drowsy mode. Memory cells, however, have to incur a small performance penalty for waking up the drowsy cells. Flautner *et al.* [8] proposed an integrated architectural and circuit technique called drowsy cache that implements a simple circuit to dynamically choose between two different supply voltage modes for leakage reduction. They analyzed different architectural policies for turning L1 lines into drowsy mode. They also showed that they can achieve good leakage power reduction by simply keeping the data portion of all the L2 lines in drowsy mode. A specific data line is reinstated to a normal, high-power mode, only when it is re-accessed with some activation penalty. Since an L2 cache takes tens of cycles to access, adding an extra cycle or two for wake-up will be insignificant to the overall performance. All prior architectural techniques of using Gated-Vdd ignored the implications and correctness issues of maintaining Multi-Level Inclusion (MLI) [5] and cache coherence, voiding their applicability. Switching off an L2 cache line while keeping the same line in the L1 active could either violate the MLI property or complicate the snooping mechanism. With the industry making a paradigm shift to multicores or MPSoC, having a leakage power saving policy for cache-coherent shared-memory MP systems is imperative.

In this paper we propose a simple, low cost, viable architectural technique called *Virtual Exclusion* to reduce leakage energy consumption in the L2 caches. This technique aggressively reduces leakage energy in the L2 or higher level caches while maintaining Multi-Level Inclusion property and cache coherence simultaneously among multiple processors. Virtual Exclusion is achieved by turning off repetitive but infrequently accessed cache lines in the higher level caches, given locality is already present in the lower level L1. For maintaining Multi-Level Inclusion, small modifications to the MOESI snooping bus coherence protocol are proposed to maintain correctness of the protocol when power saving feature is enabled. It does not need any additional hardware support other than the counters for switching off higher-level (e.g. L2) cache lines along with keeping the rest of the lines in the drowsy state. Additionally, Virtual Exclusion reduces the extra misses from the L2 cache that are introduced by the original cache decay scheme, thereby reducing both the performance penalty and dynamic energy consumption incurred by DRAM memory accesses. This ensures that the leakage energy savings is not offset by the much larger energy consumption of DRAM accesses. In addition, Virtual Exclusion can be integrated with a conventional cache decay scheme to obtain more leakage energy reduction. In this work, we provide a comprehensive analysis of the leakage energy reduction for a functioning implementation of a cache coherent multiprocessor system based on Virtual Exclusion technique. The contributions of our paper are summarized as follows.

- We provide a viable, low-overhead solution for maintaining Multi-Level Inclusion and coherence for MP systems in the context of saving leakage energy.
- The technique needs only minor changes to traditional snoop-based cache protocols, e.g. MOESI.
- We apply our techniques to two MP architectures: SMP and the emerging multicore processors, and demonstrate the advantages.

The rest of this paper is organized as follows. Section 2 overviews Multi-Level Inclusion. Section 3 proposes Virtual Exclusion and its integration with a conventional MOESI protocol for leakage power reduction. Section 4 explains our simulation methodology. Section 5 provides a detailed leakage power analysis and Section 6 concludes.

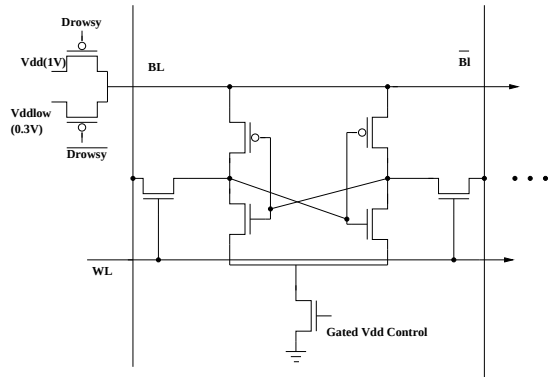


Figure 1: SRAM cell with both Gated-Vdd and DVS control

2. MULTI-LEVEL INCLUSION AND CACHE COHERENCE

In this section we overview the Multi-Level Inclusion property and describe the architectural policy changes required for having a leakage power management policies in the higher level cache (e.g. L2 or L3) while maintaining multi-level cache inclusion and coherence in a multiprocessor system.

2.1 Multi-Level Inclusion

A multi-level cache hierarchy consists of a number of levels of caches between the CPU and the main memory, with the lower level caches being closer to the CPU. Multi-Level Inclusion (MLI), proposed by Baer and Wang in [5], is a property in a cache hierarchy which requires that if a cache line is present in a lower level cache (e.g. L1), it should also be present in all the higher levels (e.g. L2 and beyond). MLI is an important property for facilitating an efficient implementation of cache coherence. Using this property the higher level cache effectively shields the lower level cache from I/O and the snooping bus. Without MLI, the lower-level caches will encounter a large number of queries from the snooping bus. This could lead to substantial performance degradation due to the limited number of ports in small, highly accessed L1 caches.

The baseline cache hierarchy we use to demonstrate MLI in this paper contains multiple cores, each with two-level caches communicating via a snooping bus. Each processor features a small L1 data cache, backed by a larger L2 cache, which is connected to the memory through the snooping bus. MOESI protocol is employed in this work to maintain cache coherence across processor cores.

The detailed algorithm and architectural support for maintaining inclusion in such a cache architecture is detailed in [5]. The second level cache needs to have an inclusion bit for every cache line to indicate whether the line is at the previous level. The following are the cache policy changes required to maintain MLI.

- For any line-fill in the L1 cache, the L2 cache sets the inclusion (I) bit for the corresponding line.
- For all evictions (Clean and Dirty) in the L1 cache, the line address is given to the L2 cache and the L2 cache resets the I bit of the corresponding line.
- All invalidation requests for cache lines in the snooping bus are propagated from the L2 cache to the L1. Both the L2 line and the L1 line are invalidated, ensuing necessary writebacks for dirty lines.
- For any write to a line at L1, the line is also marked dirty and written to the L2 cache and the L2 cache sends invalidation requests to the bus.

Each of the aforementioned changes to the protocol mentioned is an overhead in terms of cache bandwidth. They consume power and can affect performance due to cache contention. That is the reason why a snooping port is typically dedicated to caches [3, 18]. However, these enhancements are required for guaranteeing the correctness of cache coherency protocol and hence are assumed to be part of the baseline cache hierarchy in our work.

2.2 Leakage Energy Reduction Schemes for Coherent Caches

In a multi-level cache hierarchy, leakage energy reduction schemes are more appealing and profitable for higher level caches for several reasons. One reason is that higher level caches are much larger using most of the on-chip transistors, thereby consuming more leakage power that makes them good candidates for leakage power reduction. Moreover, given the high hit rates of the L1 caches, L2 or higher level caches are not frequently accessed, suggesting that they can stay idle for long periods of time. However, classic power saving schemes in caches [4] and more recent leakage management techniques like [14] do not consider the fact that a cache may be part of a multiprocessor system. Thus some assumptions as that are true in a uniprocessor system, like a low enough L2 Cache activity so that a way can be completely switched off, are not true in systems where the cache is shared by multiple processors.

Hu *et al.* [9] discussed briefly the effect of applying their policy on a cache-coherent multiprocessor system. Note that the first design priority of applying leakage power schemes to such systems is *correctness*. For their cache decay technique, even if higher level cache lines can be turned off, it is imperative that the tags and the state of a turned off line must be kept active to maintain MLI and to shield the lower level caches (e.g. L1) from snooping traffic. Another issue not addressed and evaluated in their work is the potential yet serious performance degradation and additional power consumption by extra misses going to main memory due to the L2 decayed lines.

The drowsy cache [8] does not suffer from the correctness or MLI issues as it keeps the entire state in drowsy state. As such, the drowsy cache will not introduce additional misses as the cache decay scheme. However, the issue is the increase in the latency for waking up a drowsy cache line. The performance degradation due to this is expected to be negligible since the L2 latency is typically in the order of tens of cycles. Nonetheless, the leakage power savings by using a drowsy cache is expected to be lower than the cache decay scheme as all the drowsy lines still consume some leakage power. In this work we assume a cache line circuitry where we can control the Vdd reaching the circuit as in [8] as well as gate off the supply voltage as shown in [16]. The schematic of such an SRAM cell is shown in Figure 1. Our proposed architecture technique will exploit this circuit to reduce leakage energy in caches. We discuss this technique in the next section.

3. APPLYING VIRTUAL EXCLUSION

In this section we explain why cache decay fails to work with an MLI environment. We then describe the concept of Virtual Exclusion and explain how it can be used to save leakage energy. Finally, we apply the Virtual Exclusion concept to cache decay and explain how can it help and improve cache performance over simple cache decay and still saves more leakage energy.

3.1 Generic Virtual Exclusion Policy

The drowsy cache paper [8] shows that for L2 or higher level caches the best and complexity-effective architectural strategy for leakage power control is to keep them in drowsy mode. A specific data line is activated, or woken up only when it is accessed. Since the access latency for L2 caches is large, keeping the whole cache drowsy would not incur a large performance penalty as it just adds one or two cycles to the L2 access latency. Our entire L2 cache is initially assumed to be in the drowsy mode before the Virtual Exclusion algorithm is applied.

The Virtual Exclusion scheme is added on top of the drowsy cache scheme to allow more data lines to be turned off in the cache hierarchy for saving more leakage energy. To make drowsy higher level caches work with a cache-coherent MP system, it is important to note that the tag arrays of these higher level caches (i.e. L2 in our example) must be kept *on* all the time for supporting a functional cache coherence protocol. The schematic of a cache hierarchy using Virtual Exclusion is depicted in Figure 2. Each entry in the Tag RAM of the L2 cache contains a physical address Tag (T), a Valid bit (V), a Dirty bit (D), an an Inclusion (I) bit. The state of the I bit indicates the presence of a line in the L1 so as to determine whether the data portion of the line in the L2 should be kept on in drowsy state or be Vdd-gated off. The first simple change for the Virtual Exclusion scheme is the following. Whenever there is a line-fill into the L1 cache due to an L1 miss, the same line in the L2 cache (or the missed line brought back into the L2 from main memory) will have its corresponding I bit set. This I bit precisely indicates that the data is now present in the L1 cache as well. Subsequently, the corresponding data portion of the line in the L2 is V_{dd} gated off immediately. This turn-off of the data lines in the L2 that are present in the L1 gives our technique its name. We illustrate this mechanism in Figure 2(a).

The L2 lines are turned back on under the following scenarios. Whenever a line is being displaced from the L1 due to a conflict miss, in order to explicitly maintain MLI, the L1 will inform the L2 and forward the line to the L2 for every single L1 line eviction regardless of whether the state of the evicted line is clean or dirty. Note that, since the Virtually Exclusive L2 cache does not have the data portion of a line when the line is present in the L1. Therefore, for each eviction, the L1 cache always supplies the cached, (un)modified data portion along with its address to the L2. Upon such an eviction, the corresponding L2 cache line is turned back *on* to drowsy state and the I bit is reset to zero simultaneously as depicted in Figure 2(b). Such eviction is completely off the critical path.[1] Also, it takes place only when there is an L1 miss, thus is unlikely to clobber L2 accesses and impact the performance. Another potential scenario to have data lines in L2 in drowsy state is for architecture that support instructions that prefetch data only into L2, e.g. prefetcht2 in Intel's SSE instruction set. This type of instructions bring data from main memory into L2, but not in the L1, and the data will be kept in drowsy mode for saving leakage energy until the processor makes requests for them.

In addition to the simple changes in the cache line fill and line eviction policy in the caches, some minor changes in the cache coherence protocol are also needed to maintain data consistency. First, any remote request for a cache line with an local L2 hit and its associated I bit set will cause the L2 to pass the request to the L1 cache. This is shown in Figure 3(a). Second, when a line in the L1 cache is being written, the address is provided to the L2 cache for marking the same line as *dirty* without changing any other state. Meanwhile, the L2 also needs to broadcast an invalidation signal for the address on the snooping bus. These operations are illustrated in Figure 3(b). In this way, the states of the same line in the L1 and L2 are kept consistent.

For the scenario depicted in Figure 3(a), considering a remote request in the MOESI protocol, our policy will increase latency if the tag of a line is present in the L2 cache and its I bit is set. In order to maintain correctness, we need a slight change in the protocol to handle this special case. If the requested line has its I bit set, then the only correct copy of the cache line must be present in the L1. Now any remote request pertaining to the line needs the line to be supplied to the bus. This can be done in two ways depending on the cache architecture. If the L1 cache has a direct path to the bus, then the data may be directly supplied by bypassing the L2. Otherwise, the data may be written back to the L2 cache, which in turn writes it to the bus. In our simulations, we assume that the line needs to be written to L2 and then to the bus.

While discussing Virtual Exclusion we should consider the situation where a line being replaced from the L1 Cache has already been replaced from the L2 Cache. Given the basis of our assumptions of an inclusive cache hierarchy, this situation should never take place. This is because for an inclusive cache the lines with their I bits set will not take part in cache replacement and hence will never be replaced while the line is in the L1.

It is noteworthy that apart from keeping the L1 lines in L2 *turned off*, we have other opportunities to V_{dd} gate off a few more cache lines. One obvious candidate for turn-off are the invalid lines. A line may become invalid if it is not being allocated or if it has been invalidated by remote snooping activity. These lines, including the tag arrays and data portion, can be safely turned off without incurring any performance loss. Once turned off, they will be turned back when a cache miss to the same locations occurs. Since this event involves an access to either main memory or remote caches, it could take some hundreds of cycles or more. Thus, an additional delay of a few cycles to turn a line on will incur minute impact on performance.

3.2 Cache Decay and Hybrid Virtual Exclusion Policies

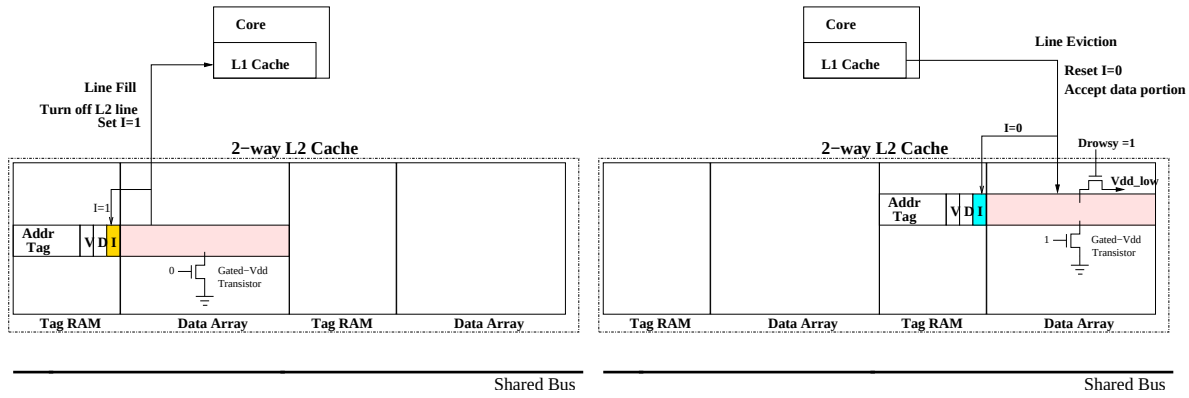
3.2.1 Generic Cache Decay in L2

The original cache decay scheme proposed in [9] does not address the correctness issue for a cache coherent multiprocessor system where Multi-Level Inclusion property needs to be enforced. The decay scheme turns off cache lines that are not used for a specified number of cycles based on the size of the decay counter employed. Since data will be lost when V_{dd} gating is applied, if a line is allowed to decay in a higher level cache when having a copy in the lower level cache, it will violate the Multi-Level Inclusion property and cause the cache coherency protocol to fail. Here we first discuss a minor change to the cache decay policy to enable Multi Level Inclusion.

To maintain the Multi Level Inclusion when cache decay is applied, the tags of lines that have their I bit set need to be always turned on even if the decay counter indicates that the line can be decayed for not being used for cycles.[2] This policy potentially de-

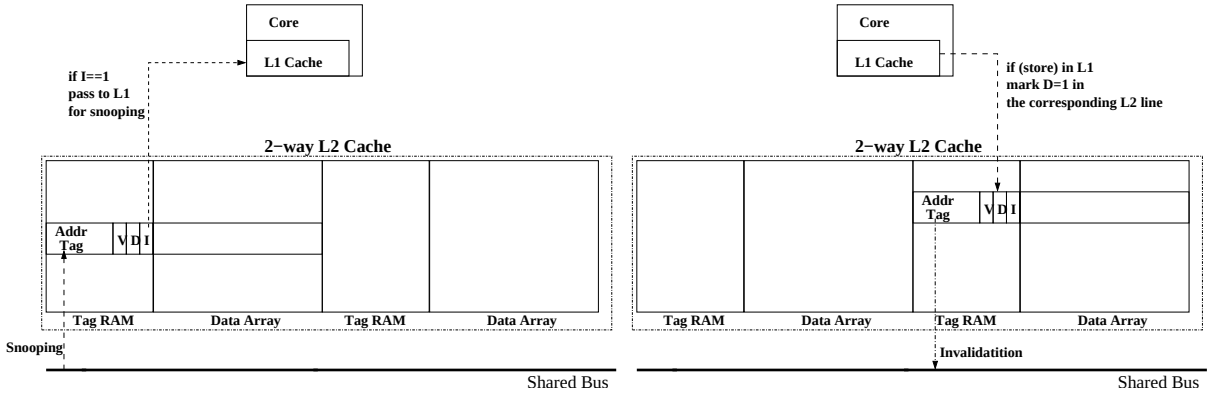
[1] The only overhead is an increase in the dynamic power consumption. This extra dynamic power overhead is accounted for as an overhead component when calculating leakage power savings.

[2] In our experiments, we always charge up the tag arrays to the



(a) L1 Line Fill (b) L1 Line Eviction

Figure 2: Cache Line Allocation for Virtual Exclusion



(a) Incoming Snooping (b) Invalidation upon Write

Figure 3: Protocol Changes for Virtual Exclusion

creases the power savings compared to the originally proposed decay policy but is indispensable to preserve the requirement for the MLI and guarantees the correct functioning of the coherence protocol. Figure 4 shows the decaying mechanism. When a line is first brought into the L2 and L1, the corresponding L2 decay counter (DC) is reset to the maximum value, e.g., 4 million [3] as shown in Figure 4(a). Similar to a normal decaying scheme, the DC starts down-counting also shown in Figure 4(a) when the corresponding L2 line is idle. Illustrated in Figure 4(c), when there is a conflict miss causing an eviction of the line, no matter it is clean or dirty, the L2 will keep down-counting. Typically, there is no action if the line was not updated during its lifetime in the L1, otherwise, it needs to be written back to the L2 if it is dirty. Nonetheless, the DC will be untouched in either scenarios. The DC will only be reset back to the maximum value when there is a request that generates an L2 hit. Figure 4(c) shows such a case.

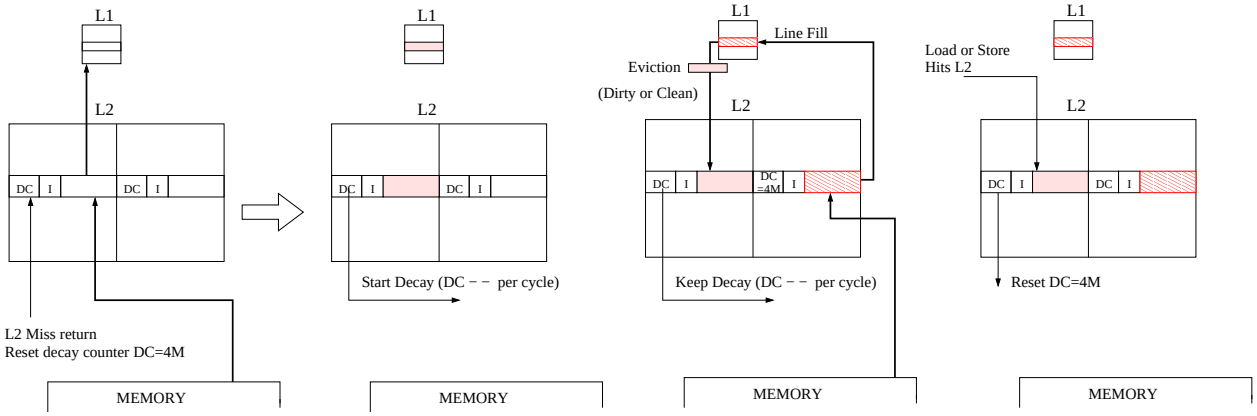
3.2.2 Hybrid Virtual Exclusion Policy

Now we discuss how to further improve the energy efficiency of the cache decay scheme using our Virtually Exclusive cache architecture. To exploit the advantages of both Cache Decay and Virtual Exclusion schemes, we will be able to further reduce the leakage energy consumption. We call our new scheme *Hybrid Virtual Exclusion* policy. There is a subtle caveat in the generic cache decay scheme. The intuition behind cache decay is that due to temporal locality a line not being used for a long time is unlikely to be used again any time soon. Based on the above, when applying decay technique to higher level L2 cache, lines in L2 is likely to decay when L1 is effective and exhibits high temporal locality. Additionally, to maintain Multi-Level Exclusion in such scenarios, the tag arrays of the L2 cannot be completely gated off for snooping reasons even if the decay counter is already counted down to zero. In other words, so long as the I bit in the L2 is set, decaying (e.g. V_{dd} gate-off) will be disabled for the L2 address tags. According to our Virtual Exclusion mechanism discussed in Section 3.1, when a line is evicted from the L1, the I bit of the same line in the L2 will be reset and the line from the L1 will be copied to its data portion. Upon this point, the decay counter will start counting down.

It is noteworthy to point out that the L2 lines with I bit set do not decay as shown in Figure 5(a), they only start decaying when the I bit is reset in Figure 5(b). Namely, the L2 lines decaying only starts after they are evicted from the L1, the difference between the hybrid and the decay scheme in the previous section. Note that, for any L2 hit, similar to the generic decay scheme, the decay counter will be reset back to the maximum value. From the above discussion, the

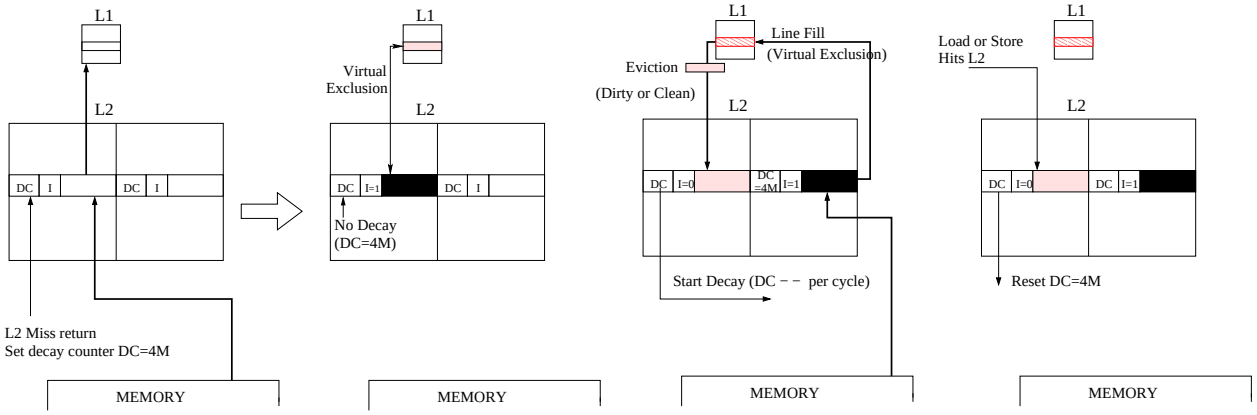
normal, high supply voltage even for the invalid lines.

[3] The 4-million cycle decay interval for the L2 Cache has been chosen using analysis of decay intervals done by Hu *et al.* in [9].



(a) An L2 Miss and reset DC; DC counts down (b) A Conflict Miss (c) An L2 Hit Before Replacement

Figure 4: Cache Decay Countdown Mechanism



(a) An L2 miss and reset DC; Virtual Exclusion turns off L2 line (b) A Conflict Miss (c) An L2 Hit before Replacement

Figure 5: Hybrid Virtual Exclusion Countdown Mechanism

decay counter, when starting to count, will always (re)start from the maximum value because (1) the decaying only starts after an eviction due to replacement; (2) any prior L1 line fill, either hit in L2 or miss the first time, must have the decay counter reset back to the maximum value.

Having our Virtual Exclusion policy applied to the Cache Decay has the following advantages.

- It reduces leakage consumption by turning off data portion of lines in the L2 that are in the L1. It does not wait until the line start to decay. In prior work, there was unnecessary leakage current consumption during the 4 million cycles the decay counter is counting down.
- For inclusive lines, the decay counters start counting only when the corresponding L1 line is evicted. This gives us a decaying victim cache, reducing the possibility of decay-induced L2 misses. Reducing a few L2 Cache misses is extremely important, because a L2 cache miss causes a memory access, which in turn consumes more energy in the DRAM and suffer from additional latency of some hundreds of cycles.

Our technique is extremely simple and it only uses the state bits, the dirty bit and the inclusion bit to determine whether to switch a cache line off. Since these state bits are already present in the cache for the purpose of maintaining coherence and inclusion, the only major area overhead will be to maintain different voltage power supplies and the simple cache line driver circuitry. There is also an extra overhead of decay counters that are also present in the original cache decay scheme. Another notable feature of our technique is that we do not inherently change the MOESI protocol in terms of state changes to a cache line or signals transmitted to the snoopy bus. This ensures that correctness of the protocol and hence coherence between the SMP caches.

3.3 Virtual Exclusion in Multicore Processors

In addition to a traditional multiprocessor system, the Virtual Exclusion technique can also be applied to the emerging multicore architectures. A modern multicore processor consists of a number of processors sharing a large L2 cache. This L2 cache may be simply a single monolithic structure or may be non-uniformly distributed among processor cores with some type of interconnection network that guarantees coherency [7, 11, 10]. Similarly, in a multicore architecture, the Inclusion bit will be set if any of the L1 caches has a copy of the line. The concept of Virtual Exclusion is the same as it is in the case of an SMP architecture explained earlier; any line with its “I” bit set will have its data array part off. Since the “I” bit being set guarantees that the line is present in some L1 Cache, any other cache requesting the data may get it through a cache-to-cache transfer. As explained previously, we also apply the decay scheme on top of our Virtual Exclusion scheme to obtain energy benefits. In multicore type structures, a large number of cores can share the L2 cache. Therefore, more L2 lines will be inclusive in several,

distinct L1 caches, thus a greater leakage saving opportunity for Virtual Exclusion — leaving a larger number of L2 data portions to be V_{dd} gated off. Also, due to a larger number of processor cores, the number of accesses to the L2 cache will be greater, too, making decaying lines by a conventional cache decay mechanism more difficult. Again, we do not change the inherent snooping bus protocol to implement our technique. Our results show that using decay with Virtual Exclusion in a multicore lead to up to 72% savings in L2 cache leakage power over a baseline drowsy L2 Cache.

4. EVALUATION METHODOLOGY

Our experiments were based on the M5 simulator system developed by the University of Michigan [6]. M5 is capable of performing a system level simulation for a snooping bus multiprocessor system. Our baseline architectural parameters along with various cache sizes are listed in Table 1. The processor is chosen to be in-order to be in tune with the some latest trend in Multicore processors that have multiple cores of simple in-order processors on a chip with an on-chip L2 Cache. The aim of these Multicore architectures is to increase throughput through TLP. An example is the Ultra SPARC T1 (Niagara) processor that contains 8 in-order processor cores on the die [2]. The power estimation tool used for estimating leakage power is based on ECacti [12]. We integrated the ECacti leakage power model into the M5 simulator to analyze both dynamic and leakage power consumptions in caches. The DRAM access energy is estimated from the data sheets of commercial DRAMs offered by Micron [1].A typical write to a 256MB DRAM costs 9.72 nJ of energy and a typical read uses 11.52nJ. All the simulations are performed on the SPLASH-2 benchmark suite [19] and SPEC CPU2000 Integer benchmark programs. To evaluate the dynamic cost of using the same counters in [9] and the modified bitline and wordline driver circuitry in [8, 9], we use the energy overhead numbers supplied in these papers and scaled down to 70nm technology using conventional technology scaling rules [13].

The simulations were carried out on two types of architectures: multicore processors and SMPs. In the multicore architecture, we simulate using six configurations. These configurations consist of an L2 cache either 256KB or 512KB, being shared by 2, 4 and 8 processor cores, respectively. For the SMP architecture, each processor contains their own L1 and L2 caches. We simulate the above two L2 cache sizes, with 2, 4, and 8 processors running on a shared bus. We implement and evaluate three distinct energy management policies for the L2 caches. The policies are:

- **Decay**: Cache Decay policy implemented to work for Multi-Level Inclusion. The decaying policy was detailed in Section 3.2.1.
- **Virtual Exclusion**: Generic Virtual Exclusion policy described in Section 3.1 for the L2 cache.
- **Hybrid**: Virtual Exclusion implemented on top of the cache decay. The policy was discussed in Section 3.2.2 with illustration.

For all the above configurations we ran the SPLASH-2 benchmark to completion. We also simulated a set of simulations for the multicore architecture that involves running heterogeneous SPEC benchmark programs on different processor cores in a multicore system. These simulations were aimed to analyze the effect of heterogeneous applications running on multiple cores that contain no data sharing. All the SPEC2000 INT benchmark programs were run for 1 billion instructions. The exact SPEC2000int programs used in our simulations are given in Table 2. The reason we did not show all the results is that not all the SPEC2000int programs were successfully ported to M5 simulation framework due to various issues such as unimplemented system calls.

Table 1: Architectural Parameters

Processor Core	In-order, stalls on cache misses
L1 D Cache Size	16KB 2-way 64-byte line
L1 I Cache Size	16KB 2-way 64-byte line
L1 Access Time	1 cycle
L2 Cache Sizes	256KB 8-way and 512KB 8-way
L2 Access Time	10 cycles Normal, 12 cycles Drowsy
Memory Access Time	200 cycles

Table 2: Spec2000 Benchmark used for simulations

2-way Multicore	bzip and gzip
4-way Multicore	bzip, gzip, crafty and gap
8-way Multicore	2 copies each of bzip, gzip, crafty and gap

5. EXPERIMENTAL RESULTS ANALYSIS

In this section we evaluate our techniques by running SPLASH-2 benchmarks for both SMP and multicore architectures. All results shown are relative savings in the leakage energy over a baseline — *drowsy cache*. All savings numbers take into account the energy consumption overhead. The overhead is different for different cache policies, this discussion encompasses all overheads considered in our analysis. We consider the overhead for the extra circuitry required for maintaining Gated-V_{dd} scheme, the energy consumed for the extra misses by DRAM memory accesses and finally for Virtual Exclusion, the overhead of bringing a line from L1 on a bus read request, and also writing clean values from L1 to L2 during evictions.

5.1 SMP Analysis

Figure 6 illustrates the energy savings for a dual processor SMP system and each processor has a 256KB L2 cache. The percentage reduction calculation is based on the baseline leakage energy (the denominator). The numerator considers both the leakage energy by each scheme and the dynamic energy overhead caused by extra trips to the DRAM memory. The rationale is to evaluate how much energy can be saved with these architectural leakage-reduction techniques. We observe that in almost all the benchmark programs the hybrid scheme shows the best saving results. The reason is that the cache decay scheme incurs a lot of overhead for the extra L2

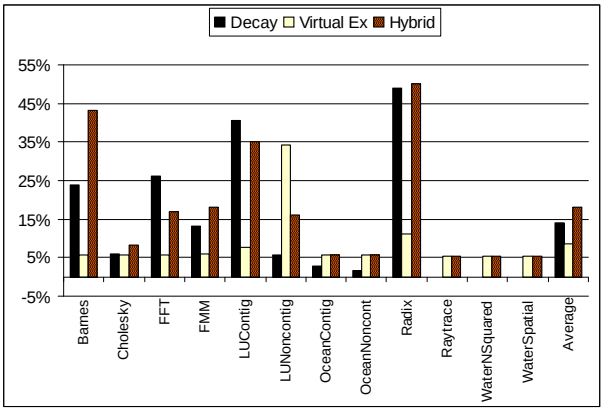


Figure 6: Leakage Energy Reduction for 2-way SMP (256KB L2, Baseline: Drowsy L2 Cache)

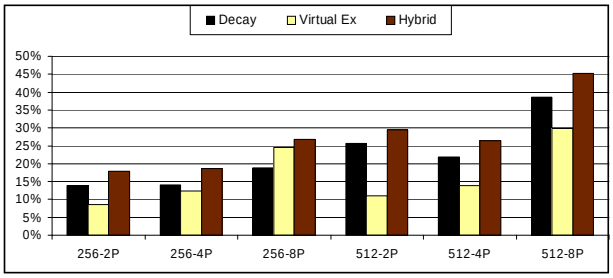


Figure 7: Average Leakage Energy Reduction over Drowsy Cache for Different SMP Configurations

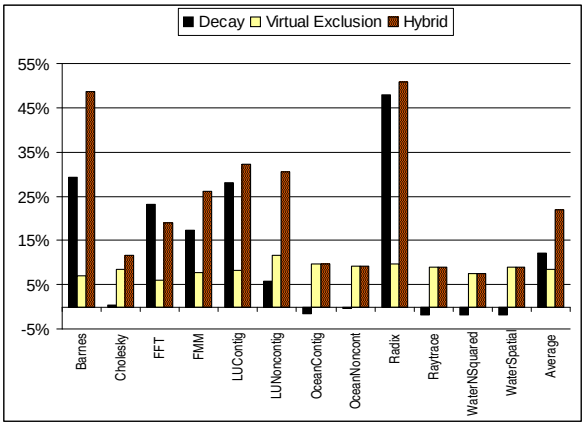
misses that consume additional DRAM memory energy. This overhead is effectively eliminated by the hybrid scheme because the hybrid scheme transforms a portion of the L2 cache into a decaying victim cache. Also note that the decay scheme for some programs failed to save energy. This happens for the same reason — the large memory access overheads caused by L2 misses. Our simple Virtual Exclusion scheme does not suffer from such overheads. But since the L1 cache size is a small percentage of the L2, Virtual Exclusion alone gives an average of 8% leakage energy savings. By combining with decay in our hybrid scheme, the average saving is increased to 20%. Also there is never a case where the hybrid scheme actually encounters energy loss. For FFT and Lu-Contig from SPLASH-2, the pure decay scheme does better than the hybrid scheme. This happens because the Virtual Exclusion scheme turns "on" the lines that are evicted from the L1 cache. On the other hand, in the decay scheme, the line might have been decayed in the L2 already, therefore, some benchmark programs show better energy savings for the decay scheme. However, as seen from the results, this policy of having a decaying victim cache is useful in reducing L2 misses and its ensuing memory access overheads if the replacement is transient.

Figure 7 plots the average L2 leakage energy savings for 2, 4, and 8 processor systems for the entire SPLASH-2 benchmark suite. The average across all the applications clearly reveals that the hybrid method is the best among all techniques. Another obvious trend from the graph is that the leakage energy savings increase with increased cache size. This is because for a given data working set, the larger the cache, the higher likelihood of decaying a line. In overall, the hybrid scheme saves from 19% to as much as 45% of leakage energy consumption of an L2.

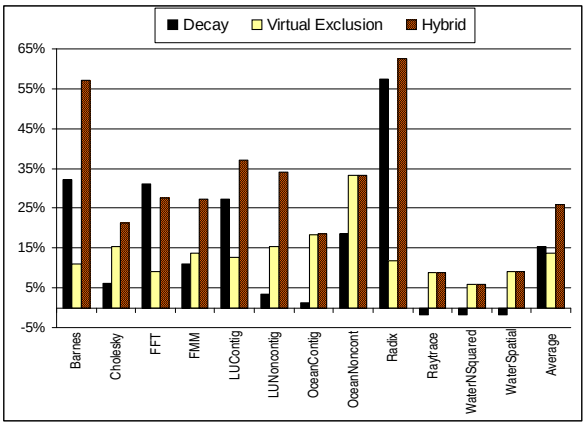
5.2 Multicore Processors Analysis

Now we show the energy results for multicore processors in Figure 8. Using the same metric, we compare the leakage energy savings for each of the three techniques, *Decay*, *Virtual Exclusion* and *Hybrid* in each figure. Figure 8(a) shows the savings for different SPLASH-2 benchmark programs for a 2-way multicore system with a 16KB L1 data and instruction cache in each processor and a 256KB L2 cache shared by the two processor cores. We can see that the leakage energy savings highly depend on the benchmark characteristics. Similar to the observations made in the SMP analysis, we find that the Cache Decay technique sometimes led to energy loss for more DRAM accesses. The Virtual Exclusion technique provides around 10% savings across all the benchmark programs. The hybrid technique obtains the best savings of L2 leakage energy, up to 52% in Radix. Neither the Virtual Exclusion nor the Hybrid technique ever shows any negative savings results.

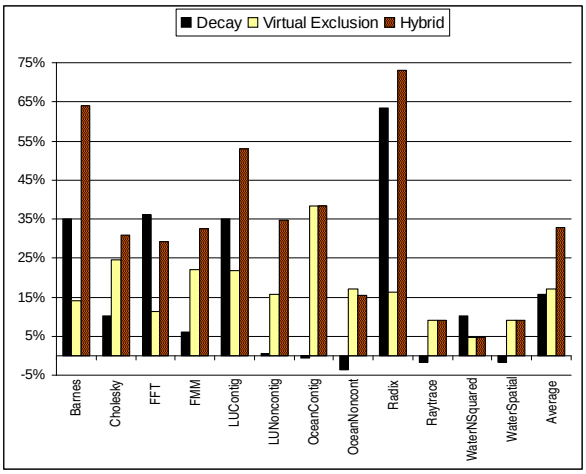
We further studied the leakage energy savings for a 4- and 8-core system using SPLASH-2. The results in Figure 8(b) and Figure 8(c)



(a) 2-way Multicore Processor



(b) 4-way Multicore Processor



(c) 8-way Multicore Processor

Figure 8: Leakage Energy Reduction for Multicore Processors (256KB L2, Baseline: Drowsy L2 Cache)

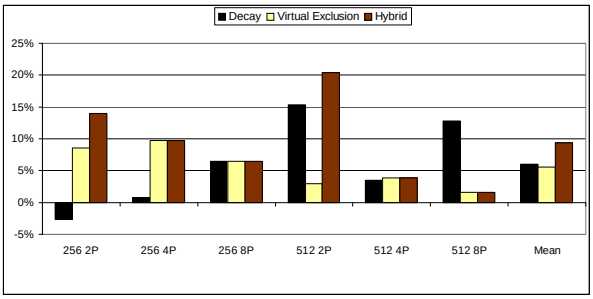


Figure 9: Leakage Energy Reduction for Multicore Systems (SPECint2000: 256KB and 512KB L2)

demonstrated similar trends to a 2-core system. In fact, as the number of processor cores increases in a multicore system, the relative leakage power savings using the hybrid scheme also increases compared to the decay scheme. This is because as the number of processors increases, the occupancy and activity in the L2 by different workloads also increases. This reduces the opportunity for the generic decay scheme to decay lines.

Figure 9 shows leakage energy savings for systems where different SPEC benchmark programs run on different processor cores. The purpose of this experiment is aimed to study the energy impact for heterogeneous applications running on a multicore system, a more realistic scenario for multiple independent single-threaded applications are concurrently executing. Note that, these applications have their respective address spaces. The combinations of SPEC2000int programs for different cores on a 2-, 4- and 8-core system are detailed in Table 2. As mentioned earlier, we subset the results simply because some SPECint programs have not been successfully ported to the M5 simulator yet. We observe that the hybrid scheme provides the best average savings (9%) for all the benchmark programs and configurations we simulated. Unlike the decay scheme, neither the Virtual Exclusion nor the Hybrid scheme ever consumes more energy (i.e. negative savings) than the baseline. As the number of processor cores keeps increasing in future generations of multicore processors, our scheme will become more effective in addressing the leakage issues.

5.3 Performance Impact

Compared to the baseline MP system with drowsy L2 caches, the performance of our Virtual Exclusion will mostly be on par. Note that during a snoop hit, the baseline system requires extra cycles to wake up the drowsy lines. On the other hand, the Virtual Exclusion needs to perform an L1 lookup for retrieving the most up-to-date data if the snoop-hit line in the L2 is turned off. In other words, both schemes suffer similar performance overheads. According to our simulation results, the performance differences between our scheme and the baseline are within the noise range (below 0.00001%) — almost negligible. Therefore, we do not report the performance results in this paper.

6. CONCLUSIONS

Multiprocessor or multicore systems are the current design trend in all processor market segments. All these designs use multiple levels of large on-chip caches, in which leakage control in caches will become highly critical for several looming issues — power management, thermal control, and circuit reliability. However, existing leakage energy saving techniques in multiprocessor systems are limited in scope because cache coherency maintenance for correctness is often neglected in these previously proposed low-power architectural designs. In this paper we present a new, low overhead architectural technique called Virtual Exclusion to save leakage energy in higher level caches that simultaneously provides guaranteed Multi-Level Inclusion property for correct operations of cache coherence protocols and saves leakage energy more effectively. Our technique shows that a significant leakage energy savings of up to 46% in an 8-processor SMP and 35% for an 8-way multicore architecture can be achieved. We envision that such a practical, easy-to-implement technique will be very useful in saving leakage energy for the cache-coherent multicore, multiprocessor systems.

7. ACKNOWLEDGMENTS

This research was supported by NSF Grants CCF-0326396, CNS-0325536, a Department of Energy CAREER Award and an NSF CAREER Award (CNS-0644096).

8. REFERENCES

[1] http://www.micron.com/products/dram/sdram/.
[2] Suns Niagara Pours on the Cores. www.mdronline.com, 2004.
[3] Advanced Micro Devices, Inc. *AMD Athlon Processor Architecture*, 2000.
[4] D. H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *International Symposium on Microarchitecture*, pages 248–, 1999.
[5] J.-L. Baer and W.-H. Wang. On the Inclusion Properties for Multi-Level Cache Hierarchies. In *Proc. of ISCA-15*, 1988.
[6] N. Binkert, E. Hallnor, and S. Reinhardt. Network-Oriented Full-System Simulation using M5. In *Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads*, 2003.
[7] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures. In *MICRO-36*, 2003.
[8] K. Flautner, N. S. Kim, S. Martin, D. Blaaw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proc. of ISCA-29*, 2002.
[9] Z. Hu, S. Kaxiras, and M. Martonosi. Let caches decay: reducing leakage energy via exploitation of cache generational behavior. *ACM Trans. on Computer Systems*, 20(2), 2002.
[10] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA substrate for Flexible CMP Cache Sharing. In *Proc. of ICS-19*, 2005.
[11] C. Kim, D. Burger, and S. W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches. In *Proc. of the ASPLOS-X*, 2002.
[12] M. Mamidipaka, K. Khouri, N. Dutt, and M. Abadir. Analytical models for leakage power estimation of memory array structures. In *Proc. of the 2nd CODES+ISSS*, 2004.
[13] A. Matsuzawa. RF-SoC - Expectations and Required Conditions. In *IEEE Transactions on Microwave Theory and Techniques*, pages 245–253, 2002.
[14] K. Meng and R. Joseph. Process variation aware cache leakage management. *Proceedings of the 2006 international symposium on Low power electronics and design*, pages 262–267, 2006.
[15] K. Nii, H. Makino, Y. Tujihashi, C. Morishima, Y. Hayakawa, H. Nunogami, T. Arakawa, and H. Hamano. A low power SRAM using auto-backgate-controlled MT-CMOS. In *Proceedings of the 1998 ISLPED*, Nov. 1998.
[16] M. Powell, S. Yang, B. Falsafi, K. Roy, and T. Vijaykumar. Gated-vdd: A circuit technique to reduce leakage in cache memories. In *Proceedings of the 2000 ISLPED*, 2000.
[17] G. Sery, S. Borkar, and V. De. Life Is CMOS: Why Chase the Life After? *Proceedings of the 39th Design Automation Conference (DAC02)*, 1:58113–297, 2002.
[18] T. Shanley. *Pentium Pro Processor System Architecture*. MindShare, Inc, 1997.
[19] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proc. of ISCA-22*, 1995.
[20] S. H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar. An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches. In *Proc. of HPCA-7*, 2001.