# Energy Savings via Dead Sub-Block Prediction

Marco A. Z. Alves* Khubaib† Eiman Ebrahimi‡ Veynu T. Narasiman†
Carlos Villavieja† Philippe O. A. Navaux* Yale N. Patt†

*Informatics Institute - Federal University of Rio Grande do Sul - Porto Alegre, RS - Brazil
†Department of Electrical & Computer Engineering - The University of Texas at Austin - Austin, TX - United States
‡NVidia - Austin, TX - United States

***Abstract*—Cache memories have traditionally been designed to exploit spatial locality by fetching entire cache lines from memory upon a miss. However, recent studies have shown that often the number of sub-blocks within a line that are actually used is low. Furthermore, those sub-blocks that are used are accessed only a few times before becoming dead (i.e., never accessed again). This results in considerable energy waste since 1) data not needed by the processor is brought into the cache, and 2) data is kept alive in the cache longer than necessary.**

**We propose the Dead Sub-Block Predictor (DSBP) to predict which sub-blocks of a cache line will be actually used and how many times it will be used in order to bring into the cache only those sub-blocks that are necessary, and power them off after they are touched the predicted number of times. We also use DSBP to identify dead lines (i.e., all sub-blocks off) and augment the existing replacement policy by prioritizing dead lines for eviction. Our results show a 24% energy reduction for the whole cache hierarchy when averaged over the SPEC2000, SPEC2006 and NAS-NPB benchmarks.**

## I. Introduction

In recent years energy efficiency has become a key design parameter in computer architecture. While the number of transistors on a chip has been increasing rapidly, the total power budget has not. Cache memories, while key to high performance, consume a significant fraction of total chip power [1]. As such, designing energy efficient processors starts with efficient design of such power-hungry components.

Caches reduce average off-chip access latency by keeping an application's most used data on-chip. They have been traditionally designed to exploit both temporal and spatial locality. Temporal locality is exploited by replacement policies (e.g., LRU replacement) while spatial locality is exploited by multi-word cache lines. However, for today's processors with a fixed cache line size, energy innefficiency occurs on two levels: 1) on a cache line level where a line is kept alive in the cache much longer than after it is last touched, and 2) on a sub-block level, when parts of a cache line which will never be used are brought into the cache, and also when active sub-blocks become dead after a few accesses but are kept alive until the line is evicted.

Our experiments show that on average the time between the last access to a cache line and its eviction is 40% of the lifetime of the line (similar results were observed by [2]). Looking at the sub-block level, on average only 56% of a cache line is used (similar results were observed in [3], [4], [5]). We make the new observation that 95% of sub-blocks become dead after fewer than four accesses. This implies that there exists considerable opportunity for energy savings in traditional cache designs.

Prior work has made attempts at achieving these benefits by, for example, predicting when a line is last accessed, and then powering it off [2], [6]. In this paper we show that doing so on a sub-block basis can significantly improve energy savings in the cache hierarchy. In fact, by considering sub-block usage patterns, our mechanism increases the potential for energy savings compared to a perfect hypothetical mechanism that turns off an entire cache line immediately after it is last accessed.

To this end, we propose the dead sub-block predictor (DSBP) to improve energy efficiency of cache memories. DSBP uses recent history information to predict which sub-block(s) will be useful and how many accesses each sub-block will receive before it becomes dead. DSBP's main goal is to reduce dynamic and static energy consumption by bringing only useful sub-blocks into the cache, and also by turning off active sub-blocks after their predicted number of accesses. We also use DSBP to improve the existing cache replacement policy by prioritizing dead lines (i.e., lines with all sub-blocks turned off) for eviction. We find that this policy effectively offsets the additional cache misses DSBP may cause when it mispredicts the usage pattern of a cache line.

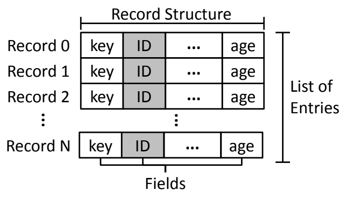
The main contributions of this paper are:

**Sub-block usage predictor:** We present a mechanism to predict and allocate only the useful sub-blocks of each cache line. Unlike prior work, which requires an access to the prediction table after each cache line access, we access the predictor structure only when the mechanism is training a new usage pattern. On average, our mechanism accesses its global structure on only 60% of the cache memory accesses.
**Dead sub-block predictor:** Our mechanism also predicts when each sub-block inside a cache line becomes dead. To our knowledge, this is the first paper that acts on a sub-block level, turning off dead sub-blocks and saving 20% energy on average per cache level compared to a traditional cache design.
**Earlier eviction of dead lines:** Our mechanism improves the cache replacement algorithm. The sub-block predictor gives feedback to the replacement algorithm by introducing dead lines as future victim lines as soon as they become dead. This reduces cache line misses by 5%.

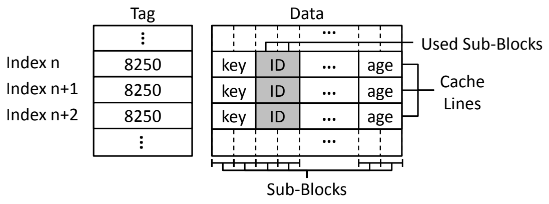
**Summary of Evaluations:** We evaluate our dead sub-block predictor design on a cache hierarchy consisting of a 32KB L1, 256KB L2, and a 2MB L3. We run the SPEC2000, SPEC2006, and NAS-NPB [7] benchmarks and find that DSBP reduces energy consumption of the cache hierarchy by 24% on average compared to the baseline (without any dead line predictor in any cache level), and by 5% compared to a technique that turns off dead lines as predicted by a state-of-the-art dead block predictor [6].

## II. Motivation

We present an example illustrating the need for sub-block level cache management, and we also present statistics about cache line usage for different single threaded benchmark applications. The results are based on experiments running SPEC2000, SPEC2006 and NAS-NPB [7] with a 32KB L1 cache, 64 byte line size, and 8 byte sub-block size. Further simulation details can be found in Section V.



(a) Data structure layout in memory.



(b) Cache memory, way number 0.

Fig. 1. Scenario with low cache sub-block usage.

During program execution, cache lines typically have low sub-block usage, as in the case illustrated in Figure 1 where a program streams through a list of records. Each record contains several fields and occupies 64 bytes as shown in Figure 1(a). The program is searching for a specific value in the ID field of each record, and therefore only that field is accessed by the processor. However, traditional caches will bring in all fields of each record as shown in Figure 1(b). Only the sub-blocks which store the ID field are useful, while the other sub-blocks remain unused until the line is evicted, resulting in considerable energy waste.

Figure 2 shows how cache lines are used at the sub-block granularity for the L1, L2 and L3 caches across multiple benchmark suites. For example, looking at the leftmost bar, one can see that 50% of L1 cache lines were evicted with only one sub-block accessed (for SPEC2000 integer). The average number of sub-blocks accessed in a cache line for each suite is presented at the top of each bar. We can therefore conclude that a significant number of cache sub-blocks that are brought into the cache are never used, thereby wasting cache energy, capacity and bandwidth.

Figure 3 shows the number of accesses a sub-block receives before it is evicted from the L1, L2 and L3 data caches. Each stacked bar represents the number of accesses accumulated on each sub-block when a line was evicted. The figure shows that on average for the L1 data cache, 45% of sub-blocks are never used, and about 50% of sub-blocks are used between one and three times. This once again shows opportunity for energy savings. Sub-blocks that are never used should not be brought into the cache. Furthermore, most of the active sub-blocks can be powered off after just a few accesses, saving even more energy. This holds true for the lower level caches (L2 and L3) as well.

Figure 4 shows the potential for static and dynamic energy savings by presenting results for three oracles implemented on the L1: 1) An oracle dead line predictor which saves leakage power by turning off cache lines as soon as they are last accessed. 2) An oracle unused sub-block predictor which saves both leakage and dynamic power by never bringing and never turning on unused sub-blocks into the cache. 3) An oracle dead sub-block predictor, which does everything oracle 2 does and in addition powers off active sub-blocks as soon as they are last accessed. The results are normalized to a baseline without any dead line predictor. Note that these oracles operate without the use of any additional prediction structures (i.e., they consume no additional energy and can only save energy) and therefore truly represent ideal scenarios.

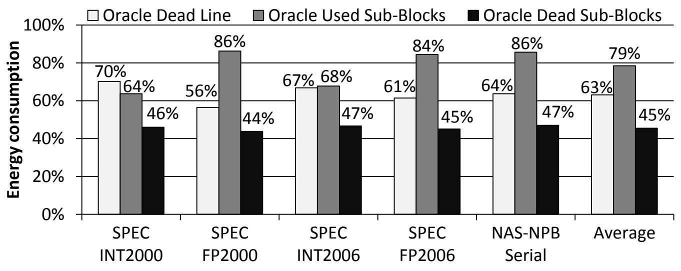


Fig. 4. Potential for L1 cache energy savings for three oracle predictors.

Oracle 1, the dead line predictor, effectively reduces cache energy (37% on average) by turning off dead lines as soon as possible. However, its effectiveness is limited by the fact that it operates only at a cache line level. Oracle 2, the unused sub-block predictor, also saves energy (by 21% on average) but not as much as oracle 1 since it does not power off cache lines or sub-blocks once they have been installed in the cache. Also note that oracle 2 is more effective on integer benchmarks than floating point benchmarks. This can be explained by the fact that the cache line usage for floating point workloads is very high as shown in Figure 2, and therefore not much is achieved by simply not bringing unused sub-blocks into the cache for these workloads. Oracle 3, the dead sub-block predictor which subsumes the two previous oracles and in addition turns off active sub-blocks once they are dead, most effectively reduces cache energy (by 55% on average). These results show that significant energy reduction can be achieved by operating at the sub-block level.

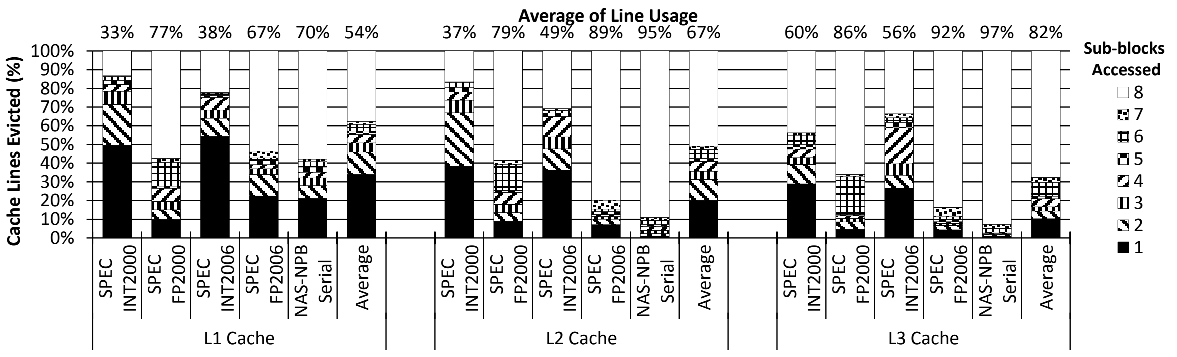


Fig. 2. 64B Cache line usage on 8B sub-block granularity.

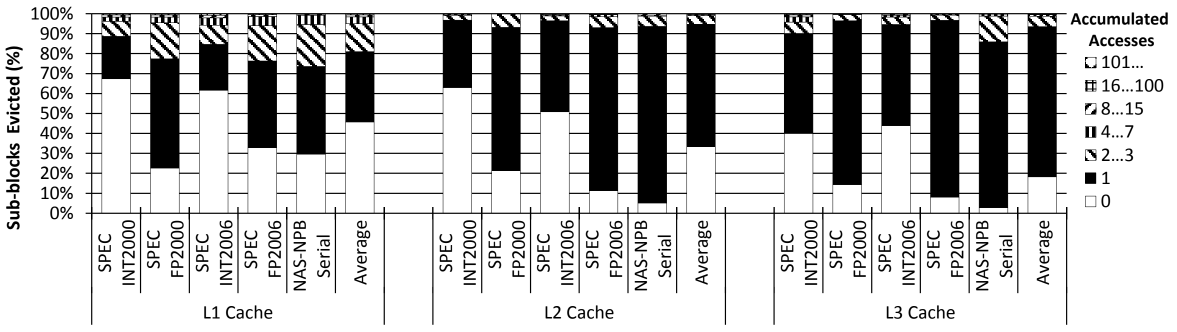


Fig. 3. Accumulated sub-block accesses before the corresponding cache line eviction.

## III. The Dead Sub-Block Predictor

This paper proposes the Dead Sub-Block Predictor (DSBP) for detecting when a given cache line sub-block is dead (i.e., will not be accessed again). We use recent history information stored in a pattern history table to predict usage patterns. Traditional gated $V_{DD}$ circuit techniques [8] are used to power off sub-blocks once they are predicted to be dead to save energy.

Figure 5 shows a traditional *sector cache architecture [9]*, and the additional *cache metadata* and *Pattern History Table (PHT)* required by our mechanism. The *sector cache* containing tag and data arrays in the left part of the figure shows how cache lines are divided into sub-blocks. The *cache metadata* contains information to guide our predictor. Each cache metadata line includes the following fields: 1) sub-block *usage counters* to store the number of accesses the sub-block is predicted to receive before it becomes dead, 2) a set of *overflow bits* to indicate if the predicted number of sub-block accesses exceeds the maximum value the *usage counters* can hold. If set, the sub-block will remain powered on until the line is evicted, 3) a *train flag* to indicate if accesses on that specific cache line should update the pattern in the PHT, and 4) a *PHT Pointer* linking a cache line to its respective entry in the PHT.

The *Pattern History Table* is used to store previous sub-block usage patterns. It is indexed by the program counter (PC) of the load/store instruction that caused the cache miss and the requested cache line offset (byte within the line) of the miss address. The key observation behind using the PC along with the line offset as the index is that a given sequence of memory instructions often times accesses the same fields of a record (see the example in Figure 1 Section II). Although different instances of a record may have different offsets within the cache line, the number of different offsets of an instance is bounded [4] [3] [10]. Therefore, the PC-offset combination provides a high coverage of patterns even with moderately sized PHTs.

Each entry in the PHT includes: 1) a *Pointer flag* indicating that some cache line has a pointer to that specific PHT entry, 2) a *valid flag* indicating if the entry contains valid data, 3) a set of *usage counters* and 4) *Overflow bits*. The usage counters and overflow bits are identical to those in the cache metadata and as you will see are copied from the PHT to the metadata as our mechanism operates.

The main operations performed by the mechanism during cache accesses are:

**Cache Line Miss**: The PHT is searched for an entry matching the PC and offset of the instruction that caused the miss. For a *PHT hit*, the mechanism will copy the PHT's usage counter and overflow bits into the cache metadata and only the sub-blocks predicted to be used are fetched and stored into the cache line. The pattern in the PHT is kept intact. If the PHT entry indicates that no other pointer exists to that entry (pointer flag field zero), the new cache line is linked to the PHT pattern. In the case of a *PHT miss*, the train flag is set, and all usage counter and overflow bits are reset in the cache metadata. The PHT will reset all the usage counter and overflow bits and evict the LRU entry to make room for a new pattern. A PHT pointer is created linking the cache metadata and the new entry. Since the train flag is set, subsequent accesses to this line will update the usage counters in the PHT.

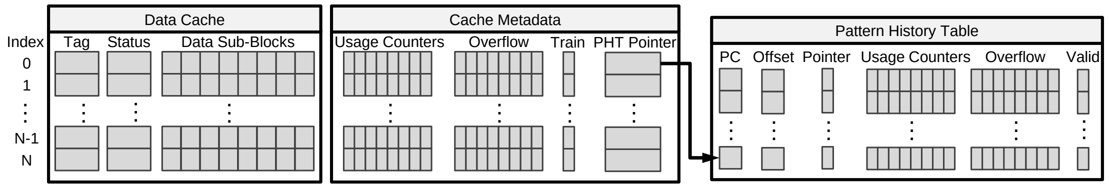


Fig. 5. Cache architecture including cache line sub-block metadata and the Pattern History Table.

**Cache Line Hit and Sub-Block Hit**: If the train flag is disabled, the sub-block usage counter in the metadata is decremented and the sub-block is turned off if its usage counter and overflow bit are zero. The PHT will be updated only when the train flag is enabled.

**Cache Line Hit and Sub-Block Miss**[1]: The requested sub-block will be brought inside the cache line and its overflow bit will be set. If the cache metadata has a valid pointer to a PHT entry, then the mechanism will increment the corresponding usage counter in the PHT entry.

**Cache Line Eviction**: If the cache line contains a valid link to a PHT entry, the flag which indicates that a valid pointer exists must be disabled in the PHT. Also, if any of the usage counters in the metadata are non-zero (indicating that the corresponding sub-block was accessed fewer than the predicted number of times) the corresponding usage counters in the PHT entry are updated by decrementing each counter by the non-zero value.

## A. A Working Example

Figure 6 illustrates how DSBP learns and predicts based on previous usage. In the piece of code present in Figure 6(a), the program is streaming through a list of records of 64 bytes each, but is accessing only a single field of the record. Therefore, just the sub-block starting from the offset value 16 is being loaded to a register. For this example, the cache and PHT are initially empty.

Figure 6(b) presents the state of the cache, metadata, and PHT after the first iteration of the loop. Since there was no matching entry in the PHT, a new PHT entry is allocated, the valid bit is set, all the usage counters and overflow bits are reset, and since no other pointer exists to that PHT entry, a new pointer will be stored in the cache metadata and the pointer flag in the PHT entry is set. Because no previous pattern was available for the prediction (i.e., PHT miss), all sub-blocks are brought into the cache line. The train flag is also set in order to capture all the subsequent access to that line and learn the usage pattern.

Assume that a single access was made to the 3rd sub-block, the corresponding usage counter in the PHT entry is incremented to one.

Figure 6(c) shows a cache miss and this time, the predictor probes the PHT and finds a matching entry. The usage bits indicate that only the 3rd sub-block will be used, and therefore only a single sub-block will be brought into the cache. Since the PHT pointer flag is already set, no new pointer will be generated. After the mechanism copies the usage counters and overflow bits to the metadata, the data can be used. Once the sub-block is used, the usage counter will be decremented to zero and the sub-block will be turned off. Subsequent loop iterations would operate in exactly the same way.

[1]The term "sub-block miss" applies to the situation where the cache line is present in the cache (i.e., a matching tag is found), but the requested sub-block is not. This is in contrast to a "cache line miss" where no matching tag is found (i.e., the entire line is not in the cache)

## B. Augmenting the Cache Replacement Policy

We also use our mechanism to improve the traditional LRU cache replacement policy by prioritizing lines with all sub-blocks powered off for eviction. If a line has all sub-blocks powered off, that means our predictor has identified this line as dead (i.e., will not be accessed again). Evciting dead lines early before they actually become victim (being at the LRU position) can reduce the cache miss ratio by letting the not-dead lines stay longer in the cache. This also offsets the additional sub-block misses our predictor may cause when it underpredicts the usage pattern of a cache line.

## C. Implementation on Multiple Cache Levels

Our mechanism operates at the sub-block level and therefore implementing our predictor on first level caches is straightforward since requests from the processor are also made at the sub-block level. However, next level caches receive requests from the previous level at a cache line granularity. Therefore, in order to implement our mechanism on systems with multi-level cache hierarchies, miss requests must be forwarded from one level to the next on a sub-block granularity. This also implies that our mechanism can be applied to a cache level only if applied to all previous levels. For example, in a 3-level cache hierarchy, we could apply our mechanism to just the L1, both the L1 and L2 but not the L3, or all three levels.

## IV. Related Work

Previous work has introduced several line usage predictors and dead line predictors which were applied to problems such as reducing static energy consumption, prefetching, and cache pollution among others.

## A. Line Usage Predictors

Chen et al. [4] proposed a Spatial Pattern Predictor (SPP) to predict cache line usage patterns. The mechanism uses the program counter (PC) and the referenced data offset to correlate historical data about line usage in order to predict future usage patterns of L1 cache lines. The goal of this technique is to reduce leakage energy by bringing into the cache just those sectors predicted to be useful. The authors also introduce a prefetching technique to bring only the predicted spatial patterns for contiguous groups of up to 512 bytes. SPP also implements a similar sectored cache with metadata to

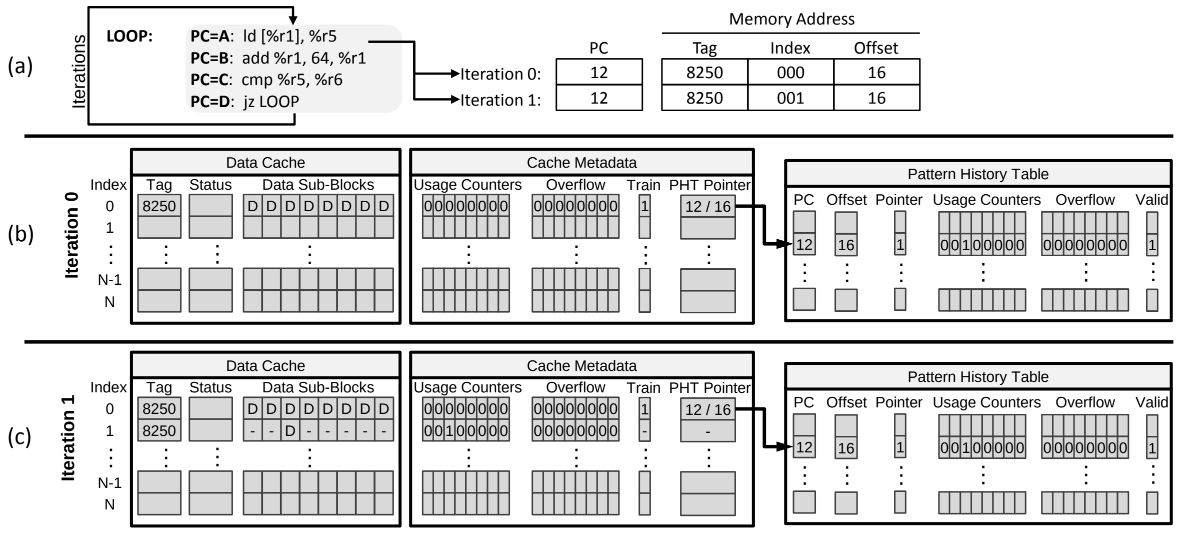


Fig. 6. Working example of DSBP

store cache line usage. However, unlike our predictor, SPP requires a PHT access for every cache access, and it only predicts if a given sub-block will be used or not but does not predict when active sub-blocks become dead.

One may think that line usage predictors like SPP [4] could be easily extended in order to start predicting dead-sub blocks. We extended SPP to do just that. However, our results show that DSBP performs on average 10% better in terms of energy reduction than the adapted SPP. Moreover, the adapted SPP increases total cache misses on all cache levels by 20% on average compared to the baseline without any predictor. This increase in total cache misses causes a 9% increase in execution time. The reason why the adapted SPP achieved poor results is because it uses an algorithm that resets the old pattern and starts a new one every time a new cache line comes into the cache so the new patterns can be learned quickly. However, this allows multiple PHT pointers to the same PHT entry to simultaneously exist and therefore incorrect patterns are recorded more frequently.

Our study performs better than previous work by tackling the energy consumption problem on all levels of the cache hierarchy, and not only predicts which sub-blocks should be brought into the cache, but also when active sub-blocks become dead. Moreover, DSBP reduces the number of accesses to the PHT by updating it only when a new pattern is detected.

## B. Counter Based Dead Line Predictor

Kharbutil et al. [5] presented two counter-based mechanisms (AIP and LvP). The paper indicates that the Live-time Predictor (LvP) delivers higher accuracy with less complexity. LvP records the number of accesses to a cache line and predicts the line as dead when the access counter reaches a certain threshold. The mechanism uses a hash of the PC which caused the cache miss to index into a table that stores the history of the number of accesses to previously evicted lines. The mechanism is used to identify dead lines early, and also to bypass dead-on-arrival cache lines.

## C. Trace Based Dead Line Predictors

Lai et al. [11] [12] introduced the Last-Touch Predictor (LTP) which uses an execution trace to predict the last touch to a cache line. The mechanism generates a signature based on a trace of instructions that access a cache line. By matching current signatures with previously stored signatures that lead to dead cache lines, the mechanism can predict when a given line becomes dead. The goal of this work is to allow the lines to self-invalidate when their last access is detected. Each cache line is augmented with 2 extra fields, one to store the signature, and the other is a single bit indicating whether the line is predicted dead or not.

Kahn et al. [13] proposed a Skewed Dead Block Predictor (SDP) to predict dead lines and used these lines as a virtual victim cache. This skewed predictor is very similar to the LTP mechanism, but uses two global tables indexed by different hash functions to reduce the impact of conflicts between them.

## D. Time Based Dead Line Predictors

Kaxiras et al. [2] presented a cache decay mechanism which uses theories from competitive algorithms to create a time-based strategy. They exploit long dead periods by turning off cache lines during such periods. This approach aims to reduce leakage power dissipated by the cache. Once the algorithm indicates that a decay interval on the order of thousands of cycles arrives, a hierarchical counter mechanism is adopted to reduce the bits required for the counters per cache line.

Abella et al. [6] introduced the Inter-Access Time per Access Count (IATAC) mechanism to predict and turn off dead lines with the objective of reducing L2 cache leakage energy. This mechanism predicts a cache line to be dead when it detects that the line has not received any accesses for a period greater than the average time between different accesses. The mechanism keeps track of the average time between accesses in a global table. To implement this, each cache line is augmented with a decay counter which is updated every 1000 cycles and therefore requires an extra adder or finite state machine for every cache line. We conservatively

TABLE I
BASELINE SYSTEM CONFIGURATION.

| | |
|---|---|
| Out-of-Order Execution Core | 96-entry reorder buffer; 32-entry load-store queue; 12 stages; fetch/decode/retire up to 4 instructions, issue/execute up to 8 micro instructions; |
| Branch Predictor | Fetch up to 2 branches; 4K-entry BTB; 64K-entry PAs predictor; |
| On Chip Caches | L1 I and D: 32KB, 8-way, 2-cycle; L2:256KB, 8-way, 5-cycle; LLC:2MB, 16-way, 12-cycle; 64B line size; 8B sub-block size (8 sub-blocks per line); LRU replacement policy; |
| DRAM and Bus | On-chip DRAM controller, Open-row; 8 DRAM banks, 4KB row buffer per bank; Row-buffer hit: 60-cycle, conflict: 180-cycle; 8B-wide core-to-memory bus at 4:1 frequency ratio; |

model this overhead as 3% of total cache energy in our evaluations.

We show that our dead sub-block predictor presents better opportunities for energy savings than conventional dead line predictors. The presented results show that DSBP outperforms previously proposed dead line predictors since it operates at the sub-block level. Moreover, DSBP is suitable to be implemented on all cache levels and does not suffer from negative interference when applied to all cache levels like other proposals do. DSBP also does not require the high overhead of previous proposals which constantly update counters for every cache line.

## V. METHODOLOGY

### A. Simulation Environment

We use an in-house cycle-accurate x86 processor simulator for our evaluation. Table I shows the baseline configuration for the processor, cache memory and main memory system.

For our evaluation, we use a total of 64 benchmarks from 3 different suites: all (12 integer and 14 floating-point) benchmarks from SPEC 2000 suite, all (12 integer and 17 floating-point) from SPEC 2006 suite, and all except DC (9) from NAS-NBP-3.3.1 [7] suite. The SPEC benchmarks were run using reference input set, and NAS-NBP using size A input set. Each benchmark was run for 200M representative instruction slice which was selected with Pinpoints [14]. All benchmarks were compiled for x86-64 using GCC 4.6.3 or GFORTRAN 4.6.3 with the -O3 option.

### B. Modeling Energy Consumption

In order to improve the energy consumption at a finer granularity than current cache memories, we turn off sub-blocks from the cache line using gated $V_{DD}$ circuit techniques as in [8]. Gated $V_{DD}$ techniques use a transistor to gate the supply voltage ($V_{DD}$) of the cache SRAM cells. Previous work reports that the transition delay of turning on a gated-ground transistor shared by a 16 byte sub-block is only 0.20 ns (i.e., one clock cycle in a 5 GHz microprocessor) [4]. Therefore, we assume our 8-byte sub-blocks can be powered on in a single cycle. Furthermore, this single-cycle latency can be hidden since a sub-block can be powered on while the data being requested is fetched from the next level in the cache hierarchy.

In order to model the dynamic and static energy savings due to dead sub-block prediction, we model both the baseline cache architecture and our proposed mechanism with CACTI 6.5 [15] at 45 nm technology. To take into account a sector cache memory, we model a cache with 8 sectors (i.e., sub-blocks) and the additional bits required to control the sub-blocks. Since our proposed mechanism requires extra metadata, the cache lines were also modeled with the extra bits necessary to support the usage counters, the overflow bits, the learn flag and the PHT pointer. After modeling the 8 sub-blocked cache with all the metadata, the CACTI power model was used to compute the access energy and leakage when all sub-blocks are enabled.

To compute the energy when just part of the cache line is turned on, we modeled cache architectures with the same number of lines but fewer 8-byte sub-blocks (from 1 to 7 sub-blocks). The energy consumed by these smaller cache lines is used to model the energy consumption when just a few sub-blocks in the cache line are enabled.

Gated $V_{DD}$ techniques require a 3% area overhead on the data array [8]. In order to model this overhead, two extra bytes (3.1% of line size) were added to the cache line in our mechanism's cache tag as the $V_{DD}$ technique overhead.

Turning off sub-blocks inside the cache line can help reduce the heat from the cache memory sub-system, which in turn reduces leakage energy[16]. However, the heat reduction benefits of our mechanism are not modeled in this work.

## VI. RESULTS: PREDICTOR MECHANISM EVALUATION

TABLE II
STORAGE OVERHEAD OF THE EVALUATED PREDICTORS.

| Predictor Mechanism | Cache Level | Cache Metadata | Predictor Structure | Total Size |
|---|---|---|---|---|
| SPP | L1[†] | 2 KB | 2 KB (512 entries) | 4 KB |
| | L2 | 16 KB | 2 KB (512 entries) | 18 KB |
| | L3 | 64 KB | 2 KB (512 entries) | 66 KB |
| LvP | L1 | 1.1 KB | 40 KB (65536 entries) | 41.1 KB |
| | L2[†] | 9 KB | 40 KB (65536 entries) | 49 KB |
| | L3 | 36 KB | 40 KB (65536 entries) | 76 KB |
| LTP | L1 | 1 KB | 8 KB (32768 entries) | 9 KB |
| | L2[†] | 8 KB | 8 KB (32768 entries) | 16 KB |
| | L3 | 32 KB | 8 KB (32768 entries) | 40 KB |
| SDP | L1 | 1 KB | 8 KB (32768 entries) | 9 KB |
| | L2[†] | 8 KB | 8 KB (32768 entries) | 16 KB |
| | L3 | 32 KB | 8 KB (32768 entries) | 40 KB |
| IATAC | L1 | 1.9 KB | 72 B (32 entries) | 2 KB |
| | L2[†] | 19 KB | 72 B (32 entries) | 19 KB |
| | L3 | 76 KB | 72 B (32 entries) | 76 KB |
| **DSBP** | L1[†] | 3 KB | 3.1 KB (512 entries) | 6.1 KB |
| | L2[†] | 24.5 KB | 3.1 KB (512 entries) | 27.6 KB |
| | L3[†] | 98 KB | 3.1 KB (512 entries) | 101.1 KB |

[†] Indicates which cache level the mechanism was originally proposed.

Previous research predicts cache usage patterns or when a cache line becomes dead. To evaluate our mechanism, we implemented the following existing predictors: *Cache line usage predictor*: SPP [4]. *Dead line predictors*: LvP [5], LTP [11], [12], SDP [13], IATAC [6]. These predictors were proposed to be used for bypassing dead lines or prioritizing them for eviction. However, in our evaluations we extend their

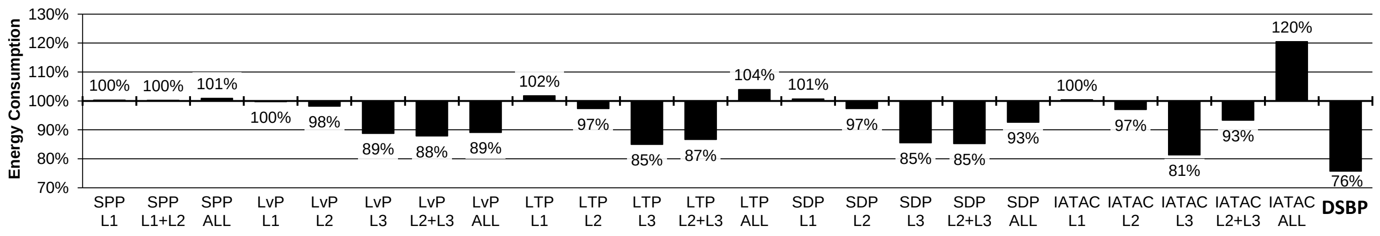


Fig. 7. Total energy consumption of the cache sub-system normalized to the baseline.

use to power off the cache lines predicted to be dead. For a more detailed explanation of each predictor, see Section IV.

The Dead Sub-Block Predictor (DSBP) evaluated in this paper uses a 512-entry PHT organized as an 8-way set associative cache. For the cache metadata described in Section III, we used 2-bit usage counters per sub-block. Table II shows the total storage overhead for the predictors evaluated in this section broken down into cache metadata storage, and global predictor table storage for each cache level. Despite requiring the second most storage (134 KB total across all 3 levels), DSBP saves the most energy due to very accurate usage pattern prediction, and relatively rare updates to the PHT.

The LvP predictor is the most costly in terms of storage (166 KB for all 3 levels) but is not as accurate as our mechanism since LvP's global prediction table is indexed using a hash function which creates some conflicts. The SPP, LTP and SDP predictors use even less storage but these mechanisms require an access to their global prediction tables every time a cache line is accessed or evicted. Therefore, these predictors consume more dynamic energy compared to our mechanism due to these extra accesses. Compared to the IATAC predictor, our mechanism only updates cache line metadata when that line is accessed whereas IATAC updates metadata for *all* cache lines every 1000 cycles consuming more dynamic energy.

## A. Energy Savings

Figure 7 presents total energy consumption for each predictor. The results are shown when the predictors are applied to each cache level in isolation (if applicable), and also when applied to multiple levels.

Among the previous work, IATAC, LTP and SDP mechanisms achieve the best results (energy reduction of 19%, 15% and 15% respectively) when the mechanism is applied in isolation to the L3. However, the energy consumption actually increases when applied to multiple levels. For example, the IATAC predictor increases energy by 20% (compared to the baseline) when applied to all cache levels. The reason for this is that IATAC's algorithm does not work well for L1 caches and the mispredictions that occur in the L1 trickle down to the next level caches and destroy their prediction accuracy as well. This negative interference between multiple cache levels destroys the energy saving benefits that occur when the mechanism is applied in isolation.

The LTP and SDP predictors use a cache line access signature to predict when a line becomes dead. When implemented in isolation on the L2 or L3 cache, many accesses are filtered out by the L1 and therefore the signatures remain mostly unique. However, when implementing these predictors on the L1 cache, no filter exists and therefore there are many more accesses to each cache line. This results in signature conflicts which in turn reduces the prediction accuracy and makes these mechanisms unsuitable for L1 caches.

The SPP predictor results show a slight energy increase due to the high associativity of its PHT. LvP reduces energy but not by as much as the other predictors due to a high number of conflicts in the global prediction table.

Finally, DSBP outperforms all of these previous proposals in terms of energy savings. It improves by 5% even the best result obtained with IATAC. DSBP is most effective because 1) it powers off data stored in the cache at the sub-block level, and 2) it does so with relatively few updates to its global prediction table (i.e., the PHT).

## B. Performance Impact

As shown in the previous section, successfully predicting dead sub-blocks can significantly reduce cache energy consumption. However, incorrect predictions may introduce a negative impact on cache performance and actually increase energy consumption due to extra cache sub-block misses. Figure 8 shows the total number of cache misses normalized to the baseline cache architecture.

The result bars shown in Figure 8 are broken down into cache *line misses* and *sub-block/extra misses*. Cache line misses are accesses where no matching tag entry was found in the cache (i.e., the entire line is not present in the cache). Sub-block misses occur when the requested tag is present in the cache, but the requested sub-block is not available. Note that true sub-blocks misses can only occur for DSBP and SPP since they are the only mechanisms that operate at the sub-block level. For the other mechanisms, it is still possible for an access to have a matching entry in the tag but that the entire line in the data store is powered off. We call such accesses *extra misses* since they only happened because the mechanism incorrectly identified a line as dead and powered it off prematurely.

Figure 8 shows that DSBP reduces the number of cache line misses since it augments the existing replacement policy by prioritizing dead lines for eviction. This reduction offsets some of the additional sub-block misses DSBP causes due to under predictions (especially for the L3). Even though DSBP slightly increases the total number of misses compared to the baseline, the cache miss ratio is increased by less than 1% (even for the L1 cache). Overall DSBP preserves the level of performance that the baseline provides while significantly reducing cache energy.

Figure 9 shows a comparison of normalized execution time

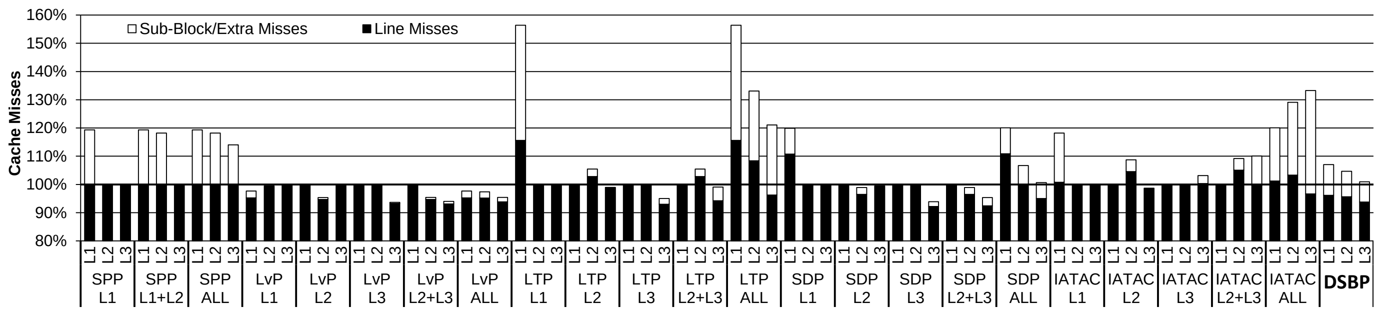


Fig. 8. Normalized cache misses.

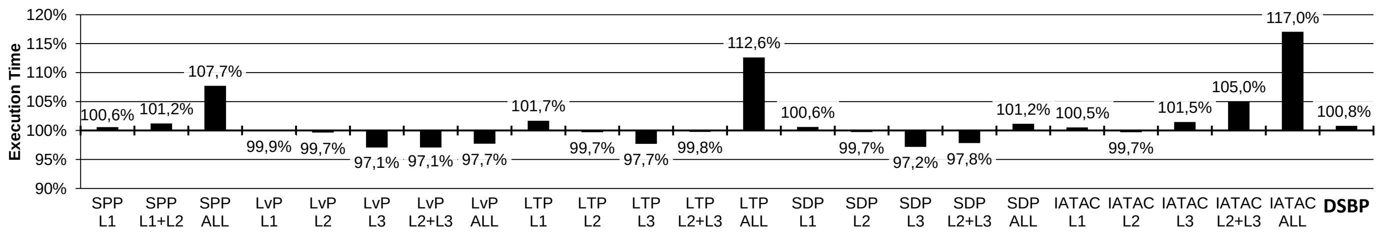


Fig. 9. Normalized execution time.

for the simulated mechanisms. The execution time correlates well with the number of misses in the last level cache since these misses are the most costly in terms of latency. Sub-block extra misses introduced by the L1 and L2 predictors can be serviced by the last level cache with minimal impact on execution time.

As mentioned before, DSBP has a negligible impact on system performance (less than 1%). This is because the additional sub-block misses DSBP may cause are largely offset by the improved replacement policy DSBP offers.

## VII. Conclusions

To our knowledge, our Dead Sub-Block Predictor (DSBP) is the first proposal to exploit dead cache line prediction at the sub-block granularity. DSBP is used to reduce energy consumption in the cache sub-system by loading into the cache only those sub-blocks predicted to be useful, and turning off active sub-blocks as they are predicted dead. In addition, the LRU replacement policy is augmented by prioritizing dead lines for eviction. This modification reduces the number of dead lines that remain inside the cache which leads to better utilization of cache space.

The results in terms of energy consumption shows significant improvement of 24% on average for the whole cache hierarchy compared to the baseline. The execution time had a negligible impact of less than 1% degradation.

We also have future directions for this work. Dead-on-arrival lines can be filtered with bypassing algorithms using the information available from our predictor. Also, our mechanism can be used to design prefetchers that bring into the cache only those sub-blocks that are predicted to be useful thereby decreasing the bandwidth demand of systems that employ aggressive prefetching.

## VIII. Acknowledgements

We gratefully acknowledge the support of the CNPq, FAPERGS, Microsoft and Cockrell Foundation.

## References

[1] S. Li *et al.*, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 469–480.

[2] S. Kaxiras *et al.*, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *IEEE/ACM Int. Symp. on Computer Architecture*, 2001, pp. 240–251.

[3] S. Kumar *et al.*, "Exploiting spatial locality in data caches using spatial footprints," in *ACM SIGARCH Computer Architecture News*, vol. 26, no. 3, 1998, pp. 357–368.

[4] C. F. Chen *et al.*, "Accurate and complexity-effective spatial pattern prediction," in *IEEE Int. Symp. on High Performance Computer Architecture*, 2004, pp. 276–287.

[5] M. Kharbutil *et al.*, "Counter-based cache replacement and bypassing algorithms," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 433–447, 2008.

[6] J. Abella *et al.*, "Iatac: a smart predictor to turn-off l 2 cache lines," *ACM Transactions on Architecture and Code Optimization*, vol. 2, no. 1, pp. 55–77, 2005.

[7] H. Jin *et al.*, "The openmp implementation of nas parallel benchmarks and its performance," in *Technical Report: NAS-99-011*, 1999.

[8] M. Powell *et al.*, "Gated-$v_{DD}$: a circuit technique to reduce leakage in deep-submicron cache memories," in *ACM Int. Symp. on Low Power Electronics and Design*, 2000, pp. 90–95.

[9] I. S. P. Division, "System/370 model 155 theory of operation/diagrams manual (volume 5)," 1974.

[10] P. Pujara *et al.*, "Cache noise prediction," *IEEE Transactions on Computers*, vol. 57, no. 10, pp. 1372–1386, 2008.

[11] A.-C. Lai *et al.*, "Selective, accurate, and timely self-invalidation using last-touch prediction," in *IEEE/ACM Int. Symp. on Computer Architecture*, 2000, pp. 139–148.

[12] A. C. Lai *et al.*, "Dead-block prediction & dead-block correlating prefetchers," in *IEEE/ACM Int. Symp. on Computer Architecture*, 2001, pp. 144–154.

[13] S. Khan *et al.*, "Using dead blocks as a virtual victim cache," in *IEEE/ACM Int. Conference on Parallel Architectures and Compilation Techniques*, 2010, pp. 489–500.

[14] H. Patil *et al.*, "Pinpointing representative portions of large intel itanium programs with dynamic instrumentation," in *IEEE/ACM Int. Symp. on Microarchitecture*, 2004, pp. 81–92.

[15] N. Muralimanohar *et al.*, "Architecting efficient interconnects for large caches with cacti 6.0," *IEEE Micro*, vol. 28, no. 1, pp. 69–79, 2008.

[16] Y. Liu *et al.*, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *Design, Automation and Test in Europe*, 2007, pp. 1526–1531.