SLIP: Reducing Wire Energy in the Memory Hierarchy

Subhasis Das* Tor M. Aamodt† William J. Dally*‡
*Stanford University, †University of British Columbia, ‡NVIDIA
subhasis@stanford.edu, aamodt@ece.ubc.ca, dally@stanford.edu

Abstract

Wire energy has become the major contributor to energy in large lower level caches. While wire energy is related to wire latency its costs are exposed differently in the memory hierarchy. We propose Sub-Level Insertion Policy (SLIP), a cache management policy which improves cache energy consumption by increasing the number of accesses from energy efficient locations while simultaneously decreasing intra-level data movement. In SLIP, each cache level is partitioned into several cache sublevels of differing sizes. Then, the recent reuse distance distribution of a line is used to choose an energy-optimized insertion and movement policy for the line. The policy choice is made by a hardware unit that predicts the number of accesses and inter-level movements.

Using a full-system simulation including OS interactions and hardware overheads, we show that SLIP saves 35% energy at the L2 and 22% energy at the L3 level and performs 0.75% better than a regular cache hierarchy in a single core system. When configured to include a bypassing policy, SLIP reduces traffic to DRAM by 2.2%. This is achieved at the cost of storing 12b metadata per cache line (2.3% overhead), a 6b policy in the PTE, and 32b distribution metadata for each page in the DRAM (a overhead of 0.1%). Using SLIP in a multiprogrammed system saves 47% LLC energy, and reduces traffic to DRAM by 5.5%.

1. Introduction

Last level caches (LLCs) have become a significant source of both static and dynamic energy consumption in modern processors, consuming up to 17% of total core energy [28]. In these large caches, data movement over wires consumes the bulk of the energy (over 90% for a 2 MB LLC according to our simulations). This paper presents Sub-Level Insertion Policy (SLIP), a class of data insertion and movement policies for wire energy dominated lower-level caches, which aims to improve data access plus movement energy.

DOI: http://dx.doi.org/10.1145/2749469.2750398

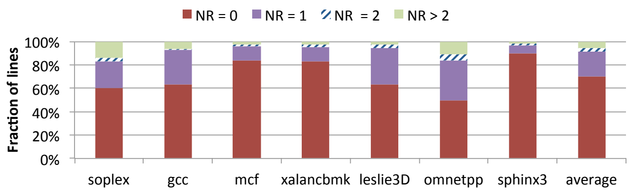


Figure 1: Lines broken down according to number of reuses (NR) before eviction

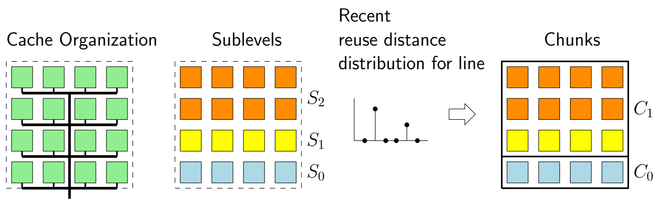


Figure 2: In SLIP, each cache level is partitioned into sublevels each with similar access energy, and sublevels are merged into chunks tailored to each line according to that line's recently observed reuse behavior

Large caches are implemented by connecting smaller SRAM banks. This can lead to significant differences in access energy and latency of the different locations where a line may be mapped. Section 2.1 discusses how cache organizations in modern processors demonstrate this energy asymmetry.

Non-Uniform Cache Access (NUCA) tackles the problem of minimizing the average access latency in a system with variations in access latencies of cache locations by aggressively moving data near to the core. Prior research has examined how to lower the cache access *latency* through intelligent insertion and movement of data [5, 11, 17, 24], data replication [4, 8, 18, 31], and replacement policies within NUCA banks [29]. However, we show that aggressively moving lines increases the cache *energy* consumption significantly.

Line movements affect average cache access latency and cache energy differently. To move a line from one location to another, at least one read and one write needs to be performed. If the movement is performed when a line receives a hit, the movement cost is equal to the read access latency cost of servicing the request. The write happens off the critical path of data servicing, and thus does not impose a direct penalty on cache access latency. Hence, line movements are almost "free" in terms of latency. However, both the read and write operations impose an energy penalty for line movements. Thus, NUCA policies optimized for latency can afford to aggressively move lines to nearer locations, while a policy optimized for cache energy will avoid doing unnecessary movements.

The effect of movement on cache energy consumption is exacerbated due to the fact that a major fraction of lines do not see a significant number of reuses in the LLC [10]. Figure 1 shows the distribution of number of reuses for each line brought into a 2MB LLC for various SPEC-CPU2006 benchmarks [2]. It can be observed that, on an average, more than 70% of lines do not receive any hits after being brought into the cache. Out of the 30% lines that do receive hits, 21% receive only a single hit. If such lines are moved to closer cache locations upon receiving a hit, more energy is wasted in moving them than is saved by subsequent accesses being from a nearby energy efficient location. For energy efficiency, it is better to initially place a line into a location tailored to its access pattern, rather than promoting it upon receiving hits.

In this paper, we propose SLIP, a class of insertion and movement policies that directly minimizes the total access and movement energy in wire-energy-dominated caches. SLIP policy views each cache level as being composed of a few (typically 3) *sublevels*, a group of ways with similar cache access energy. The sublevels are merged to form *chunks* according to the reuse distance distribution of a given line. The chunks dictate how that line is inserted and moved in the cache. An energy optimized SLIP for a line is determined *at runtime* by estimating the access plus movement energy consumption for competing SLIPs, and choosing the SLIP predicted to consume the least energy. An example of such a partitioning is shown in Figure 2. In this case, sublevel 0 consists of the 4 cache banks nearest to the processor, the next further 4 banks form sublevel 1, and the furthest 8 banks are lumped into sublevel 2. The SLIP for the shown line consists of two chunks, one consisting of only sublevel 0, and the other with sublevels 1 and 2. Further, we also show that SLIP policy, despite being less aggressive with respect to moving lines, has latency comparable to NuRAPID [11] and LRU-PEA [29], two representative NUCA policies, while achieving much lower access energy.

Using reuse distances to manage data insertion and movement, SLIP reduces total energy by 35% for the L2 cache and 22% for the L3 cache. Reuse distance has previously been utilized for making cache replacement decisions [13, 25, 36]. This body of earlier work uses reuse distance to reduce cache misses. In contrast, SLIP employs reuse distance to reduce cache access energy by selecting a set of locations for a line to be inserted or moved into. The decision of which line to evict upon such an insertion or movement is left to the underlying replacement policy, and thus SLIP is orthogonal to the choice of a replacement policy.

A wide body of literature on NUCA [4, 8, 9, 12, 18, 20, 23, 24, 31] has focused on how to effectively balance private and shared capacity in a NUCA cache for a CMP platform, and placement, replication and movement schemes for shared data. In this paper, we show that even in the simpler case of uniprocessor or multiprogrammed workloads with low or no data sharing, there exists a significant opportunity for improvement in terms of energy consumption. The more complex case of placement and movement of shared data for energy optimization is outside the scope of this paper.

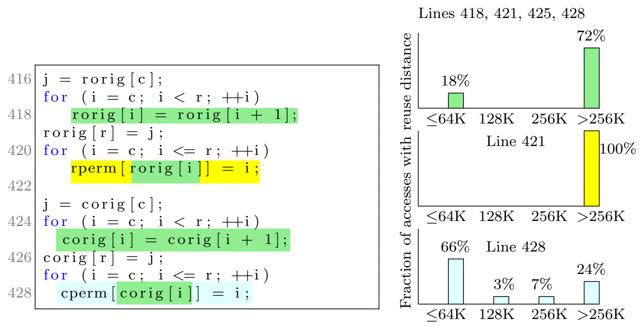

```
416 j = rorig[c];
    for (i = c; i < r; ++i)
418    rorig[i] = rorig[i + 1];
    rorig[r] = j;
420 for (i = c; i <= r; ++i)
       rperm[rorig[i]] = i;
422
    j = corig[c];
424 for (i = c; i < r; ++i)
      corig[i] = corig[i + 1];
426 corig[r] = j;
    for (i = c; i <= r; ++i)
428   cperm[corig[i]] = i;
```

Figure 3: Different classes of access patterns in soplex, the color coding in different source lines show the access pattern class it belongs to.

This paper makes the following contributions.

- It proposes a class of policies called SLIP, which can be used to dictate the placement and movement of lines among cache sublevels.
- It proposes a simple analytical model suitable for low cost implementation in hardware which, given the reuse distance distribution of a cache line, can approximate the energy consumed by a particular SLIP when applied to that line.
- It proposes an online scheme to collect reuse distance distributions by using low overhead hardware counters, and proposes an efficient hardware unit that computes an optimized SLIP for a given line based upon that line's reuse distance distribution. This scheme requires only 12b (2.3% overhead) of metadata per line in the cache, and storing 32b of distribution metadata (0.1% overhead) per page in the DRAM. The SLIP is stored in 6 unused PTE bits.
- It evaluates SLIP on memory intensive SPEC benchmarks and shows a reduction in total energy of 35% and 22% for the L2 and L3 respectively for single processor workloads. In the case of multiprogrammed workloads with a shared L3 cache, we show an energy reduction of 47% energy at the L3 level.

2. Motivation

In this section, we look at some example access patterns that arise in SPEC workloads, and discuss how an energy efficient policy can improve the cache energy consumption for those access patterns.

Figure 3 show three example reuse distance distributions from the soplex benchmark in SPEC-CPU2006, and the code fragments which give rise to those reuse distance distributions. All the code is taken from the file `forest.cc` in the soplex source code. We discuss the behavior of this code in a system with a 256KB, 16 way L2 cache (thus, each way is 16KB), where different ways have different access energies. Contemporary cache designs which present such way access energy asymmetry are discussed in Section 2.1.

The `for` loop body in line 418 rotates the `rorig` array, and does a streaming access of the array elements from `c` to `r`. These array elements are then immediately used in the loop body in line 421. Thus, the temporal locality of these lines will depend on how close the parameters `c` and `r` are to one another. Figure 3 shows that either the parameters are very close to one another, and the stream fits within a 64KB L2 cache (this happens in 18% of cases), or the parameters are so far that the stream can not be fit inside the 256KB L2 cache. The behavior of the loop body in line 425 is the same for accesses to `corig`.

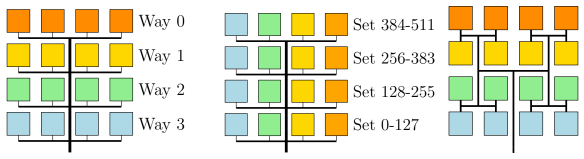
Given the varying locality characteristics described above, it is desirable to *insert* the lines belonging to the `rorig` array in a 64KB energy efficient "chunk" of the cache consisting of the 4 nearest ways to the cache controller, and evict these lines out of the L2 cache when they are evicted out of this chunk. This policy achieves two goals: i) it serves all the hits to these lines from the energy efficient location, without resorting to an energy expensive promotion policy upon hit as employed in NUCA policies, and ii) it avoids pollution of the cache with these lines when there is insufficient locality to the accesses.

The loop body in line 421 also reads the locations `rperm[rorig[i]]`. Thus, the access characteristics of the `rperm` array depend on the values stored in the `rorig` array. Figure 3 shows that these accesses almost always miss in the cache due to the random nature of the `rorig` array. Thus, it is better to bypass these lines entirely.

On the other hand, due to locality present in the `corig` array, 66% of the accesses to the `cperm` array at line 428 could be served from a 64KB cache. Another 10% of accesses, however, require a larger 256KB cache, and the remaining 24% of accesses do not fit in the cache.

Thus, for lines holding data from `cperm`, it is better to initially place these lines in an energy efficient 64KB chunk of the cache consisting of the nearest 4 ways of the cache. However, unlike the case of lines holding data from `rorig` or `corig`, here there are a significant number of hits possible from the full cache capacity. To take advantage of the full cache capacity, one can *move* these lines into the remaining 12 ways of the cache when they are evicted from the more energy efficient first four ways. This mechanism is similar to having two exclusive caches of capacities 64KB and 192KB, and thus should lead to approximately the same number of hits to these lines as a 256KB cache. This policy serves most of the references from the energy efficient 64KB portion of the cache, while also not degrading the hit rate of the line.

Different insertion and movement policies are suited to different cache lines within the same program. Also, the decision of which policy to follow can be made by estimating the amount of hits to a line given a particular cache capacity. Prior work, such as Berg and Hagersten [6], has observed that reuse distances can be used to accurately estimate the hit rates of programs at different cache capacities. We use the reuse distributions of lines collected using a low overhead hardware



(a) Hierarchical bus, way interleaving

(b) Hierarchical bus, set interleaving

(c) H-Tree

Figure 4: Various cache topology and interleaving schemes. Different shades are different ways. 4a has non-uniform energy access, adopted by Intel [19] and SRAM macro by Samsung [35]. 4b and 4c have uniform energy access.

counter mechanism to determine the number of accesses to a line that can be expected to be serviced by differently sized "chunks". These access frequencies are then used to determine an energy efficient insertion and movement policy for a line.

2.1. Cache Organizations

Large caches are constructed by joining smaller cache banks. In such a cache, a line can be located in several locations, e.g., due to the cache's associativity. SLIP exploits the energy difference between different cache ways to intelligently insert and move lines across the cache hierarchy.

There are two design decisions that impact the energy variation between the different locations where a cache line may be located. The first factor is the *topology* of the cache, i.e., how the smaller cache banks are connected by the interconnect. The second factor is the *interleaving*, i.e. how the cache ways are distributed among the SRAM banks. Below, we discuss a few cache topology and interleaving schemes, and discuss how those designs affect decision making of SLIP.

Hierarchical Bus: Figures 4a and 4b show this topology for two different interleaving schemes. A topology similar to Figure 4a was adopted for the design of an LLC slice in the Intel® Xeon® E5 family of processors [19] and a proposed SRAM macro design by Samsung [35]. In such a topology, if the ways of the cache are interleaved across the various SRAM banks, there can be a significant energy difference between the various line locations. Hence, insertion and movement decisions impact the energy consumption of caches employing such a topology and interleaving scheme. A different interleaving scheme, where all the ways belonging to the same set are mapped to the same SRAM bank, is shown in Figure 4b. In this interleaving scheme, all possible locations where a line can reside consume exactly the same wire energy, and hence there is no incentive to do any data movement.

H-Tree: [32] Figure 4c shows an H-Tree topology where reading any location consumes the same energy as reading the furthest location. Since there is no difference in access energy between cache locations, there is no reason to move a line in such a topology. While this topology provides the same access time to each bank, in wire energy dominated caches it can lead to significantly higher energy consumption compared with other interconnects that use smaller wires to access the nearer

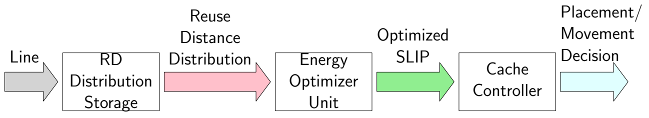


Figure 5: SLIP Overall Block Diagram: The reuse distance of a line is obtained from the reuse distribution storage described in Section 4.1. The EOU, described in Section 4.4, finds an energy-optimized SLIP for that distribution using an analytical model described in Section 3.2 in a random subset of TLB misses. This SLIP is then used to make placement and movement decisions as described in Section 3.1.

banks. We have observed that using an H-tree interconnect increases the cache energy consumption by 37% for the L2 cache and by 32% for the L3 cache compared to the baseline system (similar to Figure 4a) that we have considered, while performance is the same. A discussion on how SLIP can be applied to caches connected using a ring or mesh interconnect is given in Section 7.

3. Sub-Level Insertion Policy (SLIP)

For energy efficient cache accesses, it is desirable that the insertion and movement of a line across energy-asymmetric cache ways be customized to its reuse distance characteristics. We observe that the three energy efficient policies presented in Section 2 all follow this general pattern: *initially, insert the line into a "chunk" consisting of a certain number of cache ways, once it is evicted from that chunk, insert it into another chunk.* We call any policy with this general form a Sub-Level Insertion Policy (SLIP). Section 3.1 discusses how SLIPs are used to determine the insertion and movement of a line in a cache level. The procesure to use SLIPs for cache energy reduction is shown in Figure 5.

First, the cache controller creates a *reuse distance distribution* for each line using a low overhead hardware counter based mechanism. Section 4.1 discusses the details of how the reuse distances are represented.

Second, using this reuse distance distribution, a specialized hardware unit *finds a SLIP that minimizes the estimated total access and movement energy* for a particular line. This unit is called Energy Optimizer Unit (EOU). Section 3.2 discusses the analytical model employed by the EOU, and Section 4.4 describes how the EOU uses this model to find a suitable SLIP for a given reuse distance distribution.

Finally, the SLIP of a line is used to insert and move the line among the cache ways. Details of how this is done can be found in Section 3.1.

3.1. SLIP Description

The SLIP for any particular line partitions the cache ways into a few *chunks*. Figure 6 shows the state diagram for line insertion and movement using SLIP. When the line is inserted into the cache, a victim candidate is chosen from the chunk nearest to the processor (denoted by C_0) using the underlying cache replacement policy (LRU in our case; Section 7 discusses how state-of-the-art replacement policies such as DRRIP [22] and

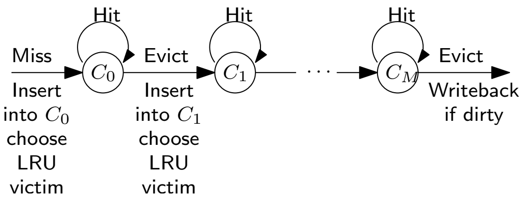

Hit
Hit
Hit
Miss
Evict
Evict
Insert into C_0 choose LRU victim
Insert into C_1 choose LRU victim
Writeback if dirty
C_0
C_1
C_M
...

Figure 6: SLIP State Diagram: Online reuse distance is used to select state diagram for line upon initial insertion. A line is initially inserted into chunk C_0, and upon eviction from chunk C_i is inserted into chunk C_{i+1}. During each insertion, the underlying cache replacement policy is used to choose a victim candidate

SHiP [39] can be adapted to support SLIP). This victim candidate is moved to a new location according to its own SLIP. When a line is evicted from a way in chunk C_i, it is inserted into a way from the next chunk C_{i+1} defined by that line's SLIP. When the line is evicted from the last chunk C_M, it is written back to the next cache level if it is dirty.

In this paper, we describe a SLIP by listing out the ways in each of the chunks in that SLIP. Thus, for example, the third policy in Section 2: "insert into the 4 nearest ways, and on eviction from those ways, insert into the next 12 ways, and upon eviction from them evict the line entirely", is denoted by {[0,1,2,3], [4,5,6,...,15]}. The second policy which bypasses the cache altogether, is denoted by {}. The first policy which inserts the line into the nearest 4 ways and bypasses the other ways is denoted by {[0,1,2,3]}.

Explosion in number of SLIPs: As the number of ways in a cache increases, the number of possible SLIPs increases dramatically. It can be shown that for a cache with W ways, there are 2^W possible SLIPs. For example, for a 3-way cache, all possible SLIPs are: {}, {[0]}, {[0,1]}, {[0],[1]}, {[0,1,2]}, {[0,1],[2]}, {[0],[1,2]}, and {[0],[1],[2]}[1]. For a 16-way cache, there can be 65,536 SLIPs. Finding the energy optimal policy among so many policies at runtime would be challenging.

To solve this problem, we lump multiple cache ways with similar access energies into a *sublevel*. Subsequently, the ways in the same sublevel are always used together in a chunk. Thus, the number of possible SLIPs is reduced to 2^S, where S is the number of sublevels. For example, in Figure 4a, each row of 4 cache ways can be considered to be a sublevel. This brings down the number of possible SLIPs from 65,536 to 16.

Representation and Storage: There are 2^S SLIPs that apply to a cache level with S sublevels. Thus, the SLIP for a level with S sublevels can be represented in S bits. Since our implementation uses 3 sublevels for both L2 and L3, both these SLIPs can be represented in 3 bits each. As explained in Section 4.1, for reducing storage overhead we use a single SLIP per cache for all cache lines in the same page. Thus, 6b need to be stored per page (3b for L2 and 3b for L3), which

[1] {[1]}, {[2]}, {[0,2]}, etc. are not considered since "skipping" ways leads to $< 1\%$ energy savings, and requires more bits to represent all SLIPs.

can fit in the ignored bits of a 64 bit page table entry (PTE)[2].

Next, we discuss two special SLIPs, which are equivalent to already known policies.

The Default SLIP: When the reuse distribution of a line is not known (during warmup), or when the reuse distance distribution is close to uniform, it is preferable that the line should treat the cache exactly as it would without SLIP. In these cases, a replacement candidate should be chosen from all the ways. The SLIP which has only one chunk consisting of all the ways does this, and is called the Default SLIP.

The All-Bypass Policy (ABP): As discussed in Section 2, in cases where a line almost always misses in the cache, it is preferable to bypass the line. This policy is the same as a SLIP with no chunks.

3.2. SLIP Energy Optimization

As shown in Figure 5, the Energy Optimizer Unit (EOU) determines an energy-optimized SLIP given a particular reuse distance distribution. The reuse distance distribution of a line is the probability P_x^d that the line x has the reuse distance d. As discussed in Section 4.3, for a random subset of TLB misses, the EOU estimates the access + movement energy for all possible SLIPs given the P_x^d of the line. It then assigns the SLIP with the minimum energy to the line x. An efficient hardware implementation of the EOU is discussed in Section 4.4.

The total access, movement, and replacement energy of a line x when using a SLIP with M chunks is

$$E_x = \sum_{0\le i<M} E_{x,i}^{access} + \sum_{0\le i<M-1} E_{x,i}^{move} + E_x^{miss} \tag{1}$$

Here, $E_{x,i}^{access}$ denotes the average access energy of the line x from chunk i, $E_{x,i}^{move}$ denotes the average energy required to move line x from chunk i to chunk $i+1$, and E_x^{miss} denotes the average miss energy consumed by the line x.

Access Energy: To find the average access energy of a line, we estimate how many accesses are made to x from chunk i. To find this quantity, note that the SLIP state machine inserts a line into chunk i after being evicted from the chunk $i-1$. Thus, the chunks act as exclusive caches to one another. The total fraction of accesses served from chunks $\le i$ can be approximated by the fraction of times the reuse distance of the line is less than the cumulative capacity of all the chunks $\le i$.[3] This implies that the fraction of accesses to line x, served from chunk i is

$$f_{x,i}^{access} = \sum_{CC_{i-1}<d\le CC_i} P_x^d$$

Here, CC_i denotes the cumulative capacity of all chunks $\le i$. Thus, the average access energy of line x from chunk i is

$$E_{x,i}^{access} = \mathbf{\bar{E}_i} \times f_{x,i}^{access} = \mathbf{\bar{E}_i} \sum_{CC_{i-1}\le d\le CC_i} P_x^d \tag{2}$$

[2]The Intel Software Developer's Manual [1] states that there are at least 14 such bits in the default 64 bit paging mode of x86-64.

[3]Here we use the approximation that a cache can only serve references that have a reuse distance less than its capacity. This is true for reuse distance defined as stack distance, LRU replacement and fully associative caches.

Here, $\mathbf{\bar{E}_i}$ denotes the average access energy from chunk i, which is obtained offline by averaging the access energies of all the sublevels belonging to that chunk.

Movement Energy: We approximate that a line is evicted from chunk G_i and inserted into G_{i+1} whenever the reuse distance of the line is greater than CC_i. A movement incurs a read from G_i and a write to G_{i+1}, so the movement energy is equal to the sum of $\mathbf{\bar{E}_i}$ and $\mathbf{\bar{E}_{i+1}}$. Thus, total movement energy of line x from chunk G_i to chunk G_{i+1}, $E_{x,i}^{move}$ is

$$E_{x,i}^{move} = (\mathbf{\bar{E}_i} + \mathbf{\bar{E}_{i+1}}) \sum_{d>CC_i} P_x^d \tag{3}$$

Miss Energy: The last component of energy is the miss energy, i.e. data access energy from the next level of the cache. We model the number of misses as the number of references that have reuse distance greater than CC_M, where M is the total number of chunks in the SLIP. CC_M is simply the sum of capacities of all the sublevels in a SLIP, and since SLIP may bypass SLIP some sublevels, CC_M may not be equal to the total capacity of the level. For example, for the SLIP [[0,1],[2,3]], CC_M is equal to the total size of the first 4 ways. We approximate the energy per miss to be equal to average access energy of all the cache ways in the next level, E_{NL}. This approximation is correct if all accesses to the next level hit and are uniformly distributed across the cache ways. Thus, the miss energy is

$$E_x^{miss} = \mathbf{E_{NL}} \sum_{d>CC_M} P_x^d \tag{4}$$

4. Implementation

In this section, we discuss a practical implementation of SLIP. First, we discuss the representation of the reuse distance distribution (P_x^d). Then, we discuss the details of how the reuse distance distributions are stored and accessed. Finally, we discuss the details of the SLIP energy optimizer unit (EOU).

4.1. Reuse Distance Distribution Quantization

To optimize the SLIP of a particular line, we need to store the reuse distance distribution of a line. In our work, we use a compressed form of reuse distance distribution that consumes only 16b, and store a single distribution per page (4KB) for each lower level cache (total of 32b for L2 and L3).

To evaluate Equations 2, 3, and 4, one needs to know the probability values $P_x(CC_i \le d < CC_{i+1})$, and the $P_x(d > CC_K)$, where CC_i is the cumulative capacity of sublevel i. For example, if the sublevels are of size 64KB, 64KB, and 128KB, we only store the terms $P_x(d < 64KB)$, $P_x(64KB \le d < 128KB)$, $P_x(128KB \le d < 256KB)$, and $P_x(d \ge 256KB)$. Thus, for a level partitioned into K sublevels, $K+1$ counts are stored. In our evaluation, each of L2 and L3 are split into 3 sublevels, thus requiring 4 counts for each distribution.

To represent each of these counts, we use a low precision integer (4 bits in our case). To avoid saturation, we halve all the counters once any counter overflows. For example, if

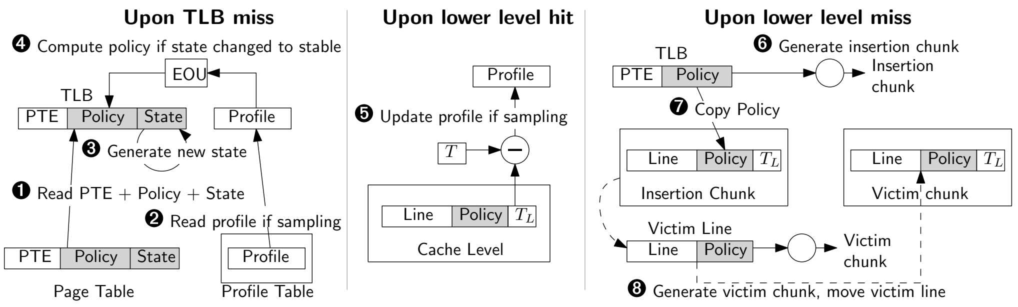


Figure 7: SLIP Usage Diagram, the policies consume 6b/line and are stored in ignored bits of a PTE, T_L consumes 6b

the previous bin counts were [4, 15, 0, 12], and a new access is made whose reuse distance lies in the bin which currently contains 15, the new counts are [2, 8, 0, 6]. Halving counters to prevent saturation also helps ensure that statistics reflect recent program behavior. Thus, the reuse distance distribution for each level consumes 16 bits (= 4 bits/bin $\times$ 4 bins).

To further reduce the storage overhead, we only store reuse distance distributions for a contiguous chunk of memory, called the *reuse distance block* (hereby referred as *rd-block*). In our evaluation, we assume a rd-block to be equal to a page (4KB) for simplicity, although this need not be the case.

We assume that all references that do not hit in the cache have a reuse distance larger than the cache size, and thus misses are counted in the last distribution bin. To collect the reuse distances of references that hit in the cache, a timestamp is associated with each cache line (T_L in Figure 7). This timestamp is derived from a few MSBs (6 in our case) of a counter associated with the cache level (T in Figure 7). T counts the number of accesses to that level and wraps around every $4C$ accesses, where C is the number of cache lines in the level. Whenever a cache hit occurs, the difference $T - T_L$ is used to calculate the reuse distance bin for this access. Subsequently, the counter corresponding to the reuse distance bin in the online reuse distance profile of the line L is incremented.

With the implementation of SLIP described so far, we observed that for some workloads with high TLB miss rates (e.g., soplex, mcf, xalancbmk, astar, and omnetpp), *distribution* metadata traffic from the lower-level caches and DRAM can be significant (Xalancbmk is the worst, with 27% increase in L2 traffic and 6% increase in DRAM traffic). This metadata traffic adversely affects energy consumption.

Also, some workloads (e.g., mcf) have phases, where the reuse behavior of lines change over time. Lines that previously caused misses and were bypassed can start to cause hits. If the bypassing SLIP is used to collect the reuse distance distribution, these hits occurring later in the program will not be observed. To solve both the problems mentioned above, we introduce the notion of time-based sampling.

4.2. Time-Based Sampling

In a time based sampling approach, a page is in one of two states: *sampling* or *stable*. The reuse distribution of a page is collected only when the page is in a sampling state. The reuse distribution for a sampling page is loaded into the TLB upon each TLB miss, and all lines in that page are inserted with the Default SLIP. On the other hand, when a page is in the stable state, the reuse distance distribution is not sampled, and the SLIP stored in the PTE is used to make insertion decisions. The state information is stored in an additional bit in the PTE. The SLIP is recalculated whenever the page becomes stable.

State transitions for each page are made randomly. A sampling page becomes stable with a probability $1/N_{samp}$ and a stable page becomes sampling with probability $1/N_{stab}$. Thus, on an average, the distribution data is fetched with a probability $N_{samp}/(N_{samp}+N_{stab})$. Our implementation uses $N_{samp} = 16$, and $N_{stab} = 256$, thus 6% of TLB misses need fetch distribution data. In Section 6, we observe that as a result, the DRAM traffic overhead is never more than 1.5%. The L2 traffic overhead is also reduced to below 2% of the baseline.

In the following sections we show a area- and energy-efficient hardware design to calculate the SLIP of a page from its reuse distance distribution.

4.3. SLIP Hardware

We outline the modifications required in the various parts of the memory hierarchy to implement SLIP in Figure 7. We discuss caches and TLBs in a system with SLIP below.

On a TLB miss, the PTE for the requested page is read from the memory hierarchy ❶. As discussed in Section 3.1, the L2/L3 SLIPs for a page are stored in the PTE itself. If the state stored in the PTE indicates that the page is in the sampling state, the reuse distance distribution for the page is also loaded ❷. The new state (sampling/stable) of the page is determined randomly ❸, and upon a transition to stable state, the SLIP of the page is recalculated ❹. During an L1 miss, the SLIPs are sent to the lower-level caches along with the miss request so that they are available in the event of a lower-level miss. If the page is in the sampling state, the Default SLIP is sent instead, as outlined in Section 4.2.

On a lower level cache hit, if the page is in a sampling state and the access hits in a cache level, the difference between the timestamp of the cache level (T) and the last access timestamp of the cache line (T_L) is used to obtain the reuse distance of the access, and T is copied back to line timestamp T_L. Recall that T counts the number of accesses to a level. This reuse distance is sent back to the core which then increments the corresponding distribution bin ❺.

On a lower level cache miss, the SLIP of the line is used to determine the chunk into which the line is inserted ❻. A victim is then chosen from the cache ways belonging to that chunk using the underlying cache replacement policy (we use LRU for evaluation). During insertion, the SLIP for both cache levels (a total of 6b) is sent along with the cache access request and is copied alongside the line as metadata ❼. This copying ensures that the SLIP of evicted lines is available without having to probe the TLB. If the incoming line is in sampling state, the last distribution bin for it is incremented in a similar fashion as for a lower-level cache hit ❺, and the cache timestamp T is copied to T_L.

When a line is evicted from a chunk, the SLIP of the line is consulted to find if there is a next chunk where this line can be moved to ❽. If there is such a chunk, a victim candidate is chosen from the ways contained in that chunk using the underlying replacement policy. Subsequently, the line and its SLIP are moved to the location of the victim ❽. This can, in turn, lead to a cascade of evictions, and we model such cascades in our evaluation. If such a chunk does not exist, the line is evicted from the level completely and a writeback request is sent to the lower cache level if the line is dirty.

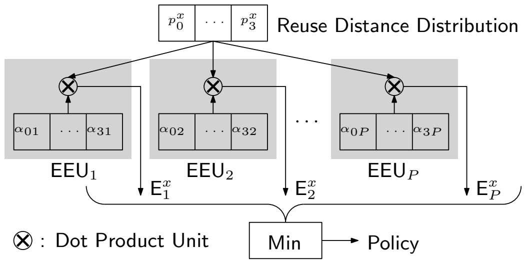
For looking up cache lines, the lookup mechanism of the underlying cache organization is used. SLIP only performs line movements and insertions during a cache miss, and leaves the cache lookup mechanism almost unchanged. The only difference to the lookup mechanism is that cache lines which are being moved by SLIP from one way to another also need to be probed to ensure correctness. In order to probe lines being moved, SLIP maintains a *movement queue* that holds lines that are being moved until they are written to their destination way. A queue for lines in movement is necessary since a moving a line can take multiple clock cycles, and can be overlapped with other cache accesses. In our implementation, we assume a 16 entry movement queue.

For invalidating cache lines, again the internal invalidation mechanism of the cache is used. An invalidation request also needs to probe the movement queue for presence of the line being invalidated.

Cache coherency with SLIP: Maintaining cache coherence in a cache with SLIP only requires the addition of the movement queue (discussed above), since cache coherence only requires the implementation of a lookup and an invalidation mechanism. Also, in cases where an inclusive last-level cache is used to simplify coherence, using the all bypass policy (ABP) is not desirable. A line which is bypassed from the last-level cache using the ABP cannot be inserted into any higher level in an inclusive hierarchy, leading to significant performance degradation.

4.4. Energy Optimization Unit (EOU)

The Energy Optimization Unit is a simple and energy efficient hardware unit that uses an array of Energy Evaluation Units



p_0^x ··· p_3^x
Reuse Distance Distribution
α_{01} ··· α_{31}
α_{02} ··· α_{32}
α_{0P} ··· α_{3P}
EEU$_1$
EEU$_2$
EEU$_P$
E$_1^x$
E$_2^x$
E$_P^x$
⊗ : Dot Product Unit
Min
Policy

Figure 8: Energy Optimizer Unit Implementation: Each energy optimizer unit is built out of an array of EEUs, each EEU performs a dot product of the reuse distance distribution with the vector of coefficients corresponding to its SLIP according to Equation 5. Here, P = total number of SLIPs.

(EEU) to compute the energy for every possible SLIP given a particular reuse distance distribution. A schematic for the overall optimization unit is shown in Figure 8.

Given a particular reuse distance distribution and a SLIP, an EEU uses Equation 1 to compute an approximate energy per access. Equation 1 is a sum of the terms in Equations 2, 3, and 4. Note that in these equations, the quantities in boldface ($\bar{\mathbf{E}}_\mathbf{i}, \mathbf{E}_\mathbf{NL}$) are only dependent on hardware parameters and the SLIP itself. $\bar{\mathbf{E}}_\mathbf{i}$ is the average access energy of all the sublevels in chunk i, and thus depends only on the chunk itself and the access energies of the sublevels within that chunk. $\mathbf{E}_\mathbf{NL}$, defined in Section 3.2, is the average access energy of the cache ways in the next level and is a constant. Thus, given a particular reuse distance distribution, all the three terms in Equation 1 are linear combinations of the reuse distance probabilities. Thus, E_j^x, the average total access, movement and miss energy consumed when using SLIP j for line x, can be expressed as a linear combination of reuse distance probabilities:

$$E_j^x = \sum_i \alpha_{ij} p_i^x \qquad (5)$$

Since α_{ij} are constants given the actual hardware and the SLIP, each EEU corresponding to a SLIP is preprogrammed with the coefficients α_{ij}. Given a particular reuse distance distribution p_i^x, an EEU performs a dot product between the coefficients and the reuse distance distribution to arrive at an energy estimate. Sunsequently, the EOU picks the lowest energy SLIP out of all the alternatives.

5. Methodology

We use MARSSx86 [33], a full system simulator for x86 ISA. The system parameters used in the simulation are shown in Table 1. The system runs Ubuntu 11.04 over Linux 2.6. In the simulation, less than 0.1% were OS instructions.

The cache organizations are assumed to be similar to Figure 4a. The SRAM banks were modeled as being similar to that of the Intel® Xeon® E5 processor LLC design [19]. The L2 is modeled as a 2 (wide) ×4 (high) array of 32KB SRAM banks. Each SRAM bank is assumed to contain two complete ways of the L2. The L3 is modeled as a 16 × 4 array of 32KB

Core parameters	
Core	4-way OoO, 128 ROB entries, 2.4 GHz
L1 I/D cache	32 KB, 8 way, 4 cycle
L2 cache	256 KB, 16 way, 7 cycles
L3 cache	2 MB, 16 way, 20 cycles
DRAM latency	100 cycles
SLIP parameters	
L2 Sublevel sizes	64 KB, 64 KB, 128 KB
L2 Sublevel latency	4, 6, 8 cycles
L3 Sublevel sizes	512 KB, 512 KB, 1 MB
L3 Sublevel latency	15, 19, 23 cycles
RD-vectors	4 bits $\times$ 4 bins $\times$ 2 vectors
Timestamp accuracy	6 bits

Table 1: System parameters

Technology node	45 nm
Wire energy per transition	0.16 pJ/bit/mm
Wire delay	0.3 ns/mm
L2 Baseline access	39 pJ
L2 Sublevel access	21 pJ, 33 pJ, 50 pJ
L2 Metadata access	1 pJ
L3 Baseline	136 pJ
L3 Sublevel access	67 pJ, 113 pJ, 176 pJ
L3 Metadata access	2.5 pJ
DRAM energy	20 pJ/bit

Table 2: Energy parameters

SRAM banks. Each row of the array is assumed to contain a single way of the L3.

For both L2 and L3, the nearest 4 ways to the cache controller comprise the first sublevel, the next nearest 4 ways comprise the second, and the furthest 8 ways comprise the third sublevel. To model the cache port contention due to line movements, the cache port is blocked while performing the read and write operations for line movement. A fully associative 16 entry movement queue is used for the correctness of lookup mechanism. A synthesized RTL model of the queue requires 0.3pJ per lookup, which is included in the results.

The energy and latency per access of the caches and the sublevels, obtained with HSPICE simulations using PTM CMOS and wire models [7], are shown in Table 2. In this table, metadata includes the policies for both the levels (3b each) and the timestamp (6b), for a total of 12b of data per line. DRAM energy was obtained as the sum of Idd4 and Idd7RW energies reported by Vogelsang [38].

To evaluate the energy consumption of the EOU, we synthesized and verified a RTL design in TSMC 45nm technology. The latency for each operation (evaluating the energy of all the SLIPs and finding the minimum energy SLIP) was found to be 2 processor cycles at a processor clock rate of 2.4GHz and the throughput is one computation per cycle. The EOU occupies 0.00366 mm^2 ($<$ 0.1% of LLC area), and each optimization operation consumes 1.27 pJ ($<$ 0.5% of LLC access energy) including pipeline registers. To model the policy update overhead, the TLB is blocked for one cycle whenever an update of the SLIP for a page takes place. The EOU computation occurs while updating the SLIP of a page, which is off the data read/write critical path. Since the EOU is fully pipelined, there is no extra queuing occurring at the EOU.

For this study, we used memory intensive SPEC-CPU2006 benchmarks, identified by Jaleel [21], since non-memory intensive benchmarks do not have opportunity to save energy at L2 and L3 levels. For each workload, we simulate up to 10 simpoints of 500M instructions each that account for over 90% of the overall execution, obtained by using PinPoints [34].

We simulate NuRAPID and LRU-PEA policies and compare them against SLIP, both with and without ABP policy. For a fair comparison with NuRAPID and LRU-PEA, we take the d-group sizes and bankcluster sizes to be same as the SLIP sublevel sizes.

6. Results

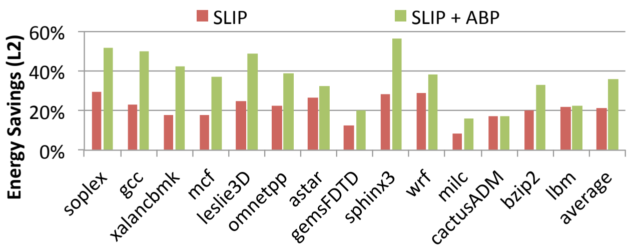
Energy: Figure 9 shows the energy reductions achieved by the various policies at the L2 and L3 cache levels respectively. SLIP without ABP saves 21% energy at L2 and 13% at L3. Adding ABP to the pool of SLIPs increases the energy savings to 35% and 22% respectively. With ABP many insertions are bypassed entirely, reducing insertion energy. On average, NuRAPID consumes 84% more energy at L2 and 94% more energy at L3 compared to the baseline cache, while LRU-PEA consumes 79% more energy at L2 and 83% more energy at L3. We omit NuRAPID and LRU-PEA in these figures, as these two policies consume more energy than the baseline.

The higher energy savings in L2 can be attributed to the fact that a higher number of insertions are bypassed in L2. This happens because the access energy differential between the L2 and the L3 is much lower than that between the L3 and the DRAM. Thus, SLIP is more aggressive in bypassing lines from L2 since the energy penalty for a miss is low. Figure 14 shows that 27% of the lines are bypassed at L2 and 14% are bypassed at L3.

We also obtained full system dynamic energy savings (core + all caches + DRAM) for SLIP and SLIP+ABP. The results are shown in Figure 10. SLIP reduces full system energy by 0.73%, while SLIP+ABP reduces it by 1.68%.

In order to evaluate the benefits of SLIP in different technology nodes, we also simulated SLIP on a 22nm technology node using the same parameters as Table 1. In this technology, SLIP+ABP saves 36% of L2 energy, and 25% of L3 energy.

Access and movement energy breakdown: Figure 11 shows the normalized access and movement energies for L2 and L3 caches. Movement energy includes inter-sublevel movement energy, insertion energy, and writeback energy. The figure shows that movement energy dominates the energy of lower-level caches. Both LRU-PEA and NuRAPID achieve lower access energy than SLIP, but consume excessive move-



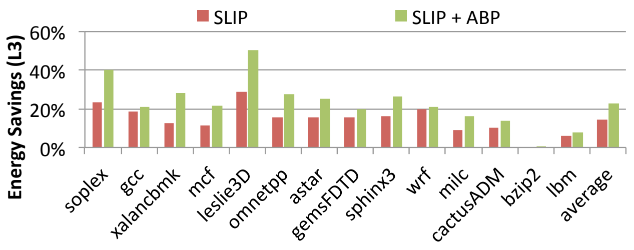


Figure 9: Energy savings over regular cache hierarchy by various policies at different cache levels. NuRAPID and LRU-PEA are omitted here, since they both increase energy consumption at both the cache levels. NuRAPID increases L2 and L3 energy by 84% and 94% respectively. LRU-PEA increases them by 79% and 83% respectively.

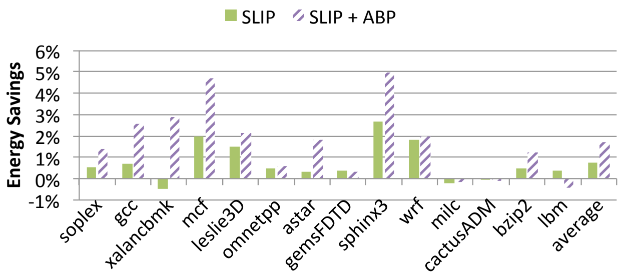
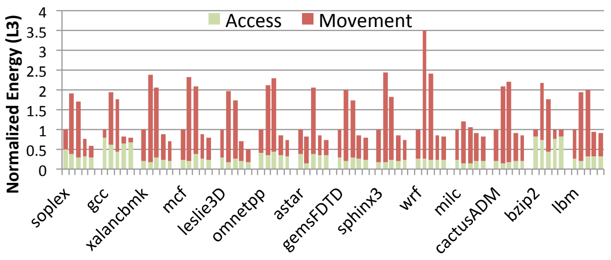


Figure 10: Full-system energy savings



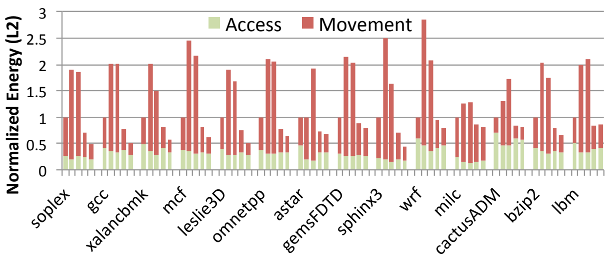


Figure 11: Energy breakdown into access and movement energy. The bars in each group show the energy for baseline, NuRAPID, LRU-PEA, SLIP and SLIP+ABP respectively. Movement energy includes inter-sublevel movement energy, insertion energy, and writeback energy.

ment energy. SLIP optimizes the sum of access and movement energy rather than targeting them individually.

Cache Hit Rates: Figure 12 shows the relative miss traffic broken down into demand misses and metadata for the L2 and L3 levels for SLIP and SLIP+ABP. It can be observed that, even including metadata, SLIP and SLIP+ABP decrease the total miss traffic by avoiding cache pollution. On average, they decrease the miss traffic by 1.7% and 2.4% for L2, and 1% and 2.2% for L3 respectively.

It can also be observed that the number of L2 misses due to metadata overhead can be significant in some cases such as soplex. However, most metadata requests get serviced from the L3, and thus the metadata traffic to DRAM is low. NuRAPID, being a movement-only policy, does not change the number of DRAM accesses. Our simulations show that LRU-PEA reduces the DRAM traffic by 0.8%, due to its prioritizing evicting demoted blocks.

Speedup: NuRAPID, LRU-PEA, SLIP, and SLIP+ABP have average speedups of 0.06%, 0.16%, 0.24%, 0.75% respectively, shown in Figure 13. The higher speedup achieved in SLIP+ABP policy (up to 3%) over other policies is due to higher hit rates from cache bypassing. The difference in our measured performance for LRU-PEA and that reported in [29] is due to the difference in workload (SPEC vs PARSEC) and in latency to the different levels of memory. The hit rates of SPEC workloads in L2 and L3 are low. Thus, the average memory access time is dominated by DRAM access time.

Classification of insertions according to SLIPs: SLIPs can be classified into four classes: a) the All Bypass Policy, b) policies apart from the ABP which bypass one or more sublevels, e.g., $\{\{S_0\}\}$, $\{\{S_0\},\{S_1\}\}$ etc. which we call "Partial Bypass" policies, c) the Default policy, i.e., $\{\{S_0,S_1,\ldots,S_{K-1}\}\}$, and, d) other policies which do not bypass any levels but are not Default, e.g., $\{\{S_0\},\{S_1,\ldots,S_{K-1}\}\}$. As shown in Figure 7, a SLIP is assigned to a line upon insertion to a cache level. Figure 14 shows the fraction of insertions with each category of SLIP described above. We can observe that a significant fraction of inserts are either fully or partially bypassed, and that partial bypassing, full bypassing and Default constitute more than 95% of all insertions. Thus, we conclude that the SLIPs with all levels are not too useful and used more in L3 than in L2.

Sublevel Access Fractions: Figure 15 shows the average fraction of accesses from different sublevels for NuRAPID, LRU-PEA, SLIP, and SLIP+ABP. It can be observed that all the policies increase the number of references to the most energy efficient sublevel – sublevel 0. NuRAPID and LRU-PEA aggressively promote lines to nearer locations and thereby have a much higher fraction of accesses from those sublevels. However, as we can see from Figure 11, this higher fraction is achieved at the cost of a significant increase in movement

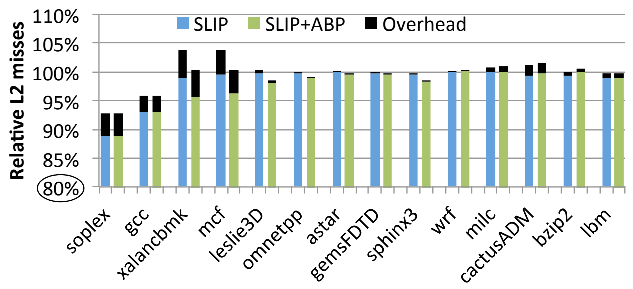
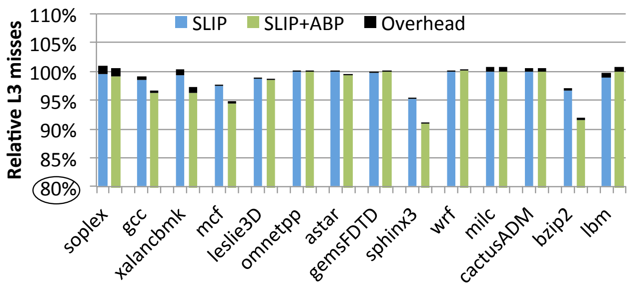



Figure 12: Relative miss traffic from L2 and L3 broken down into demand misses and overheads, for SLIP and SLIP+ABP policies

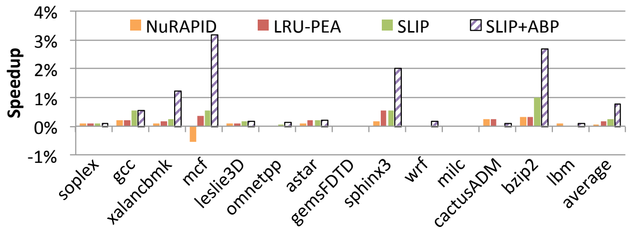


Figure 13: Speedups of policies vs. regular memory hierarchy

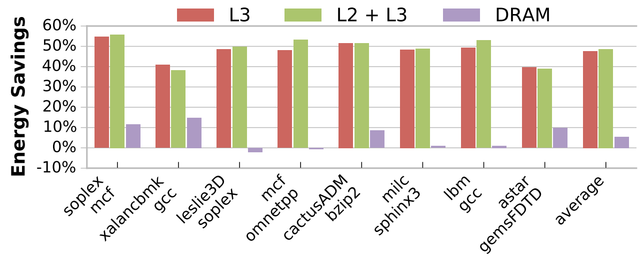


Figure 16: Multicore results

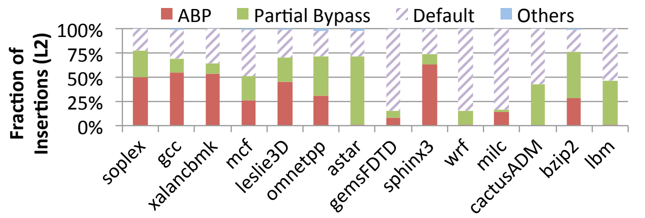
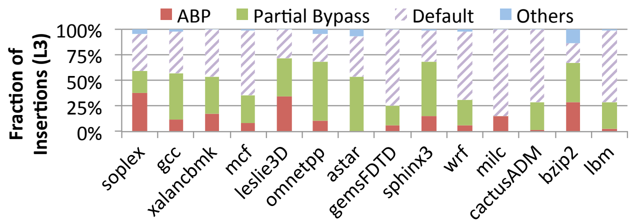
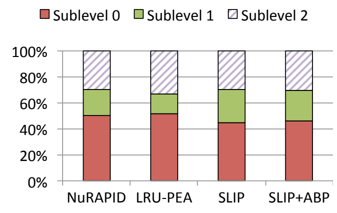



Figure 14: Breakdown of insertions according to optimal SLIP

energy, which adversely impacts cache energy.

Impact of distribution accuracy: We varied the bit width of each distribution bin. We obeserved that with an accuracy of 4 bits the energy savings were within 1% of larger bit widths. There is a sharp drop in energy efficiency when 2 bit wide bins are used, as a smaller number of hits rounded off to zero increases line bypassing, hence increasing LLC and DRAM


(a) L2 sublevel access fractions

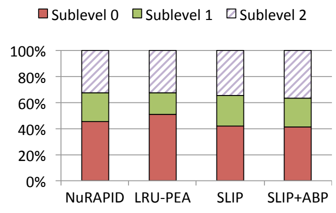

(b) L3 sublevel access fractions

Figure 15: Fractions of accesses served from different sublevels for NuRAPID, LRU-PEA, SLIP and SLIP+ABP

accesses. We omit detailed results due to space constraints.

Evaluation for multicore workloads: We evaluated the SLIP+ABP strategy for a two core system with 2 MB shared LLC and 256 KB private L2 caches. We used 8 randomly selected multiprogrammed workload mixes, each simulated for a total of 500M instructions, after fast forwarding 3B instructions. We collect statistics only for the duration that the executions of the benchmarks overlap.

In this case, the energy savings in L2 are identical to that of the single core scenario since the L2s are private. The total energy reductions the L2 and L3 caches for each of the mixes are shown in Figure 16. It can be observed that on average SLIP saves 47% of energy at the L3 level. Also, the DRAM traffic is reduced by 5.5% on average. The worst case degradation in DRAM traffic (2%) is experienced by the leslie3D and soplex mix. We also simulated NuRAPID and LRU-PEA for these workload mixes. NuRAPID and LRU-PEA increase the L3 energy by 97% and 85% on an average respectively. In a shared multicore LLC, the reuse distance of each reference is higher. Thus, more references cause misses and end up being completely bypassed from the cache. This leads to less insertions and thus more energy savings in a multicore system as compared to a single-core system.

7. Discussion

Applicability of different replacement policies: To choose a replacement candidate for a line from a chunk, SLIP needs to choose a victim line from *any subset of ways*. State-of-the-art replacement policies such as DRRIP [22], SHiP [39] can be adapted for this form of replacement in the following way.

Each sublevel can have its own replacement policy metadata. To choose a victim candidate from a chunk, a random sublevel can be chosen from the sublevels contained in the chunk in the ratio of their sizes, and then a victim line chosen from that sublevel. Below we show that this does not affect the scan and thrash resistance of DRRIP.

The scan resistance of DRRIP depends on the scan length being smaller than $(2^M - 1) * (A - w)$, where M is the length of the re-reference prediction value (RRPV), A is the cache associativity and w is the working set size. Using a random sublevel in the ratio of sublevel sizes preserves the scan resistance of DRRIP since a) sublevels have proportional associativity, and, b) randomly distributing lines into sublevels in the ratio of their sizes distributes proportional amounts of scan and working set lines into the different sublevels. The thrash resistance of DRRIP works by inserting a random small fraction of lines with a *long* RRPV. Randomly inserting the lines into the sublevels should insert the same fraction of lines with a long RRPV. Thus, the scan and thrash resistance of DRRIP is preserved and the miss rates using such a randomized insertion mechanism should not be affected significantly.

Extension to larger page sizes: In this work, we have considered 4 KB page sizes, and we have assumed that the access properties of lines in a page are homogeneous. However, this assumption may not hold true for larger page sizes. In such situations, the rd-block size can be made to be smaller than a page size, and a SLIP and distribution can be stored per rd-block. This design will require a SLIP-cache to cache the SLIPs for each rd-block, which can be managed like a TLB.

SLIP for CMPs using ring/mesh interconnect: In a large chip multi-processor system, smaller per-core cache slices are joined by ring or mesh interconnect. In such a system, there are two more problems in addition to insertion and movement policy for private data: i) how to partition the capacity of the cache between the different cores, and ii) how to place and migrate shared data. Several works (e.g., [5, 8, 17, 20]) have dealt with the first problem. Given a partitioning of the cache among the various cores, one can apply SLIP to minimize the access energy within each partition. Thus, SLIP is orthogonal to such cache partitioning approaches.

Several authors have also dealt with the second problem (e.g., [4, 17, 31]). Hardavellas et al. [17] have shown that even placing the shared data in a random slice according to a hash of the data address works near optimally. They also showed an OS based mechanism to classify data into one of shared, private or instruction. In case of applying SLIP to a CMP platform, one can utilize such a classification mechanism and apply SLIP only to the core private data. SLIP is agnostic to the placement and migration policies of shared data.

8. Related Work

NUCA techniques: To address the growing contribution of wire delay in modern caches, Non-Uniform Cache Access (NUCA) schemes have been proposed. Kim et al. [26] propose the Static NUCA (S-NUCA) and the Dynamic NUCA (D-NUCA) schemes. In the S-NUCA scheme, each cache line is mapped to exactly one location in the cache banks, making the insertion policy simple but causes energy inefficient accesses to frequently accessed lines placed in distant bank. In the D-NUCA, scheme hotter lines are moved to nearer banks by generational promotion leading to excessive movement energy.

Chishti et al. [11] propose the NuRAPID scheme, where a large cache is partitioned into a few distance groups (d-groups) of banks with similar delay. Lines are initially placed in the nearest d-group. A line is demoted by being evicted from a d-group and promoted upon receiving hits. Lira et al. [29] propose the LRU-PEA policy that preferentially evicts demoted lines, based on the observation that lines which receive a single hit tend to receive more hits. Incoming lines are mapped to a random bank, and lines are promoted upon receiving hits. Both these policies increase energy consumption due to excessive line movements. In contrast, SLIP intelligently inserts lines to an initial location based on the reuse distance history of the line, and avoids excessive line movement.

In order to reduce energy for NUCA architectures, Udipi et al. [37] and Gracia et al. [15] show how to improve the network energy consumption. Udipi et al. propose using low swing wires and bus based interconnects instead of expensive on-chip networks. SLIP can be employed in caches using such interconnects to further reduce the energy consumption. Bardine et al. [3] propose a NUCA organization where the workload footprint is estimated and unnecessary ways are turned off to save leakage energy. SLIP, being a dynamic energy reduction technique, can be used in conjunction with such static energy reduction techniques.

Cache Bypassing: A wide body of work [14, 16, 27, 30] has explored how to perform cache bypassing to improve LLC hit rates. A central theme of such work is to determine when an incoming cache block is less likely to be reused than the already present blocks in the cache. SLIP can adopt such sophisticated policies, further reducing movement energy.

Reuse Distance Based Replacement: Keramidas et al. [25] propose a cache replacement policy based on reuse distance prediction, in which they predict the Estimated Time to Access (ETA) of every cache line based on their past reuse distances, and replace the line with the highest ETA. This scheme and other cache replacement proposals [22, 39] are orthogonal to SLIP and can be used to determine a victim candidate during a line insertion or movement.

9. Conclusion

In this paper, we have proposed and evaluated SLIP, a class of insertion and movement policies that uses reuse distance distributions to minimize the total access and movement energy. An energy-optimized SLIP is assigned to each page at runtime based on the reuse distance distribution of the lines in the page. In a single core system, SLIP reduces L2 and L3 cache energy by 35% and 22% respectively, and reduces DRAM traffic by 2.2%. It is also applicable without any modifications to a multicore scenario, saving 47% energy for a shared L3, and reducing DRAM traffic by 5.5%.

References

[1] "Intel® 64 and IA-32 architectures software developer's manual," pp. 4–28, 2014. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf

[2] "SPEC CPU^{TM} 2006," 2014. [Online]. Available: http://www.spec.org/cpu2006/

[3] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, and C. Prete, "Way adaptable D-NUCA caches," *International Journal of High Performance Systems Architecture*, vol. 2, no. 3, pp. 215–228, 2010. Available: http://inderscience.metapress.com/index/L71373X85236V576.pdf

[4] B. Beckmann, M. Marty, and D. Wood, "ASR: Adaptive selective replication for CMP caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006, pp. 443–454.

[5] N. Beckmann and D. Sanchez, "Jigsaw: Scalable software-defined caches," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*. IEEE Press, 2013, p. 213–224. Available: http://dl.acm.org/citation.cfm?id=2523721.2523752

[6] E. Berg and E. Hagersten, "StatCache: a probabilistic approach to efficient and accurate data locality analysis," in *Performance Analysis of Systems and Software, 2004 IEEE International Symposium on - ISPASS*, 2004, pp. 20–27.

[7] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "Predictive technology model," 2002. Available: http://ptm.asu.edu

[8] J. Chang and G. S. Sohi, "Cooperative caching for chip multiprocessors," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture*. IEEE Computer Society, 2006, pp. 264–276. Available: http://dx.doi.org/10.1109/ISCA.2006.17

[9] M. Chaudhuri, "PageNUCA: Selected policies for page-grain locality management in large shared chip-multiprocessor caches," in *IEEE 15th International Symposium on High Performance Computer Architecture, 2009.*, Feb. 2009, pp. 227–238.

[10] M. Chaudhuri, "Pseudo-LIFO: The foundation of a new family of replacement policies for last-level caches," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 401–412. Available: http://doi.acm.org/10.1145/1669112.1669164

[11] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Distance associativity for high-performance energy-efficient non-uniform cache architectures," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003, p. 55–66. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1253183

[12] S. Cho and L. Jin, "Managing distributed, shared L2 caches through OS-level page allocation," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006, pp. 455–468.

[13] M. Feng, C. Tian, C. Lin, and R. Gupta, "Dynamic access distance driven cache replacement," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 3, pp. 14:1–14:30, Oct. 2011. Available: http://doi.acm.org/10.1145/2019608.2019613

[14] J. Gaur, M. Chaudhuri, and S. Subramoney, "Bypass and insertion algorithms for exclusive last-level caches," in *Proceedings of the 38th annual international symposium on Computer architecture*. ACM, 2011, pp. 81–92. Available: http://doi.acm.org/10.1145/2000064.2000075

[15] D. Gracia, G. Dimitrakopoulos, T. Arnal, M. Katevenis, and V. Yufera, "LP-NUCA: Networks-in-cache for high-performance low-power embedded processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1510–1523, Aug. 2012.

[16] S. Gupta, H. Gao, and H. Zhou, "Adaptive cache bypassing for inclusive last level caches," in *Proceedings of the 27th IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2013, pp. 1243–1253. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6569900

[17] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: Near-optimal block placement and replication in distributed caches," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*. ACM, 2009, p. 184–195. Available: http://doi.acm.org/10.1145/1555754.1555779

[18] E. Herrero, J. González, and R. Canal, "Elastic cooperative caching: An autonomous dynamically adaptive memory hierarchy for chip multiprocessors," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*. ACM, 2010, pp. 419–428. Available: http://doi.acm.org/10.1145/1815961.1816018

[19] M. Huang, M. Mehalel, R. Arvapalli, and S. He, "An energy efficient 32-nm 20-MB shared on-die l3 cache for intel® xeon® processor e5 family," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1954–1962, Aug. 2013.

[20] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. Keckler, "A NUCA substrate for flexible CMP cache sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1028–1040, Aug. 2007.

[21] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation," *Web Copy: http://www.glue. umd. edu/ajaleel/workload*, 2010. Available: http://www.jaleels.org/ajaleel/workload/SPECanalysis.pdf

[22] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," in *ACM SIGARCH Computer Architecture News*, vol. 38, 2010, pp. 60–71. Available: http://dl.acm.org/citation.cfm?id=1815971

[23] L. Jin and S. Cho, "SOS: A software-oriented distributed shared cache management approach for chip multiprocessors," in *18th International Conference on Parallel Architectures and Compilation Techniques, 2009.*, Sep. 2009, pp. 361–371.

[24] M. Kandemir, F. Li, M. Irwin, and S. W. Son, "A novel migration-based NUCA design for chip multiprocessors," in *International Conference for High Performance Computing, Networking, Storage and Analysis.*, Nov. 2008, pp. 1–12.

[25] G. Keramidas, P. Petoumenos, and S. Kaxiras, "Cache replacement based on reuse-distance prediction," in *25th International Conference on Computer Design, 2007. ICCD 2007*, 2007, pp. 245–250.

[26] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *Acm Sigplan Notices*, vol. 37, 2002, p. 211–222. Available: http://dl.acm.org/citation.cfm?id=605420

[27] L. Li, D. Tong, Z. Xie, J. Lu, and X. Cheng, "Optimal bypass monitor for high performance last-level caches," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, 2012, pp. 315–324. Available: http://dl.acm.org/citation.cfm?id=2370862

[28] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 469–480. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5375438

[29] J. Lira, C. Molina, R. N. Rakvic, and A. González, "Replacement techniques for dynamic NUCA cache designs on CMPs," *The Journal of Supercomputing*, vol. 64, no. 2, pp. 548–579, May 2013. Available: http://link.springer.com/article/10.1007/s11227-012-0859-6

[30] R. Manikantan, K. Rajan, and R. Govindarajan, "NUcache: An efficient multicore cache organization based on next-use distance," in *IEEE 17th International Symposium on High Performance Computer Architecture.*, 2011, pp. 243–253. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5749733

[31] J. Merino, V. Puente, and J. Gregorio, "ESP-NUCA: A low-cost adaptive non-uniform cache architecture," in *2010 IEEE 16th International Symposium on High Performance Computer Architecture*, Jan. 2010, pp. 1–10.

[32] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2007, pp. 3–14. Available: http://dx.doi.org/10.1109/MICRO.2007.30

[33] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSSx86: A Full System Simulator for x86 CPUs," in *Design Automation Conference*, 2011.

[34] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi, "Pinpointing representative portions of large intel® itanium® programs with dynamic instrumentation," in *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2004, p. 81–92. Available: http://dl.acm.org/citation.cfm?id=1038933

[35] T. Song, W. Rim, J. Jung, G. Yang, J. Park, S. Park, K.-H. Baek, S. Baek, S.-K. Oh, J. Jung, S. Kim, G. Kim, J. Kim, Y. Lee, K. S. Kim, S.-P. Sim, J. S. Yoon, and K.-M. Choi, "13.2 a 14nm FinFET 128mb 6t SRAM with VMIN-enhancement techniques for low-power applications," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, Feb. 2014, pp. 232–233.

[36] M. Takagi and K. Hiraki, "Inter-reference gap distribution replacement: An improved replacement algorithm for set-associative caches," in *Proceedings of the 18th Annual International Conference on Supercomputing*. ACM, 2004, pp. 20–30. Available: http://doi.acm.org/10.1145/1006209.1006213

[37] A. N. Udipi, N. Muralimanohar, and R. Balasubramonian, "Non-uniform power access in large caches with low-swing wires," in *International Conference on High Performance Computing.* IEEE, 2009, pp. 59–68. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5433222

[38] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2010, pp. 363–374. Available: http://dx.doi.org/10.1109/MICRO.2010.42

[39] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr, and J. Emer, "SHiP: Signature-based hit predictor for high performance caching," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 430–441. Available: http://dl.acm.org/citation.cfm?id=2155671