RECAST: Boosting Tag Line Buffer Coverage in Low-Power High-Level Caches "for Free"

Won-Ho Park, Andreas Moshovos
Electrical and Computer Engineering
University of Toronto
{wonho, moshovos}@eecg.toronto.edu

Babak Falsafi
Electrical and Computer Engineering
Carnegie Mellon University
babak@cmu.edu

Abstract

We revisit the idea of using small line buffers in-front of caches. We propose ReCast*, a tiny tag set cache that filters a significant number of tag probes to the L2 tag array thus reducing power. The key contribution in ReCast is* S-Shift*, a simple indexing function (no logic involved just wires) that greatly improves the utility of line buffers with no additional hardware cost. S-Shift can be viewed as a technique for emulating larger cache blocks and hence exploiting more spatial locality but* without *paying the penalties of actually using a larger L2 cache block. Using several SPEC CPU2000 applications and a model of an aggressive, dynamically-scheduled, superscalar processor we demonstrate that a practical ReCast organization can significantly reduce power in the L2. Specifically, a 64-entry ReCast comprising eight sub-banks of eight entries each can filter about 50% of all tag probes for a 1Mbyte L2 cache. A conventional line buffer of the same size filters only 32% of all tag probes. The resulting average reduction in L2 tag power is 38% and 85% with writeback or writethrough L1 caches respectively. This translates to a reduction of 16% or 52% of the overall L2 power respectively. We also analyze a few representative applications explaining why S-Shift works well.*

1 Introduction

Caches dissipate a significant portion of the total CPU power. Most of the architectural cache power reduction techniques were targeted at the L1 caches since they dissipate most of the power. Very little research has focused on reducing power dissipation in the L2 or higher caches.

Reducing the power dissipation of high-level caches is becoming increasingly important as their power is bound to increase both in *relative* and in *absolute* terms. There are three reasons why this is the case. First, as a result of the increasing disparity between processor cycle time and memory access time, high-level caches are becoming larger and highly-associative while additional levels of caching are introduced (the size and associativity of L1 caches are severely limited by timing considerations). Accordingly, the *absolute* power dissipated by high-level caches is bound to increase. The trend towards larger L2 caches is further strengthened by techniques that increase cache pressure such as SMT and CMP. Second, as power reduction techniques for the execution core and the L1 caches are perfected, the *relative* power dissipation of high-level caches increases substantially. Finally, recent processor designs aimed at the SMP market incorporate writethrough L1 data caches greatly increasing the frequency of L2 accesses and L2's power dissipation[1].

The applicability of most L1 power reduction techniques has not been demonstrated for the L2 and it is not trivial since different trade-offs apply. For example, way-prediction [23] can result in reduced power in the L1 but its usefulness for the L2 will be limited to the tag array since the tag and data array accesses are serialized. Also, an increase in latency is more tolerable at the L2 than at the L1 thus different techniques may be applicable.

In this work we focus on dynamic power optimizations for the tag arrays (or simply tags) of the L2 or higher caches. Contrary to the L1, the power of the tags in typical L2 caches is comparable to that of their data arrays. This is because the tag and data array accesses are serialized [1,15] and because the data array can be sub-banked activating only its necessary parts.

We revisit the idea of using small line buffers in-front of the tags to reduce the number of probes and thus reduce power in high-level caches [7,14]. We study *ReCast*, a tiny filter placed between the LI and the L2 tags where it caches a small number of recently accessed L2 tag sets. As shown in Figure 1, ReCast prevents some L2 tag probes and thus it reduces power dissipation in the L2 tags. Since ReCast is placed in-between the L1 and the L2 it increases L2 latency for certain accesses.

The key question we ask in this work is whether we can improve the utility of ReCast without hurting performance and miss rates. Accordingly, our key contribution in this work is *S-Shift,* an indexing function that increases the number of accesses that are filtered by ReCast at no additional hardware cost. S-Shift is very simple but at the same time extremely effective. It can be viewed as a technique that emulates having larger L2 cache blocks but without the penalties associated with actually using larger L2 cache blocks. Since S-Shift changes the way we index the L2 it also impacts L2's hit rate. We demonstrate that this trade-off is favorable in virtually all cases.

In addition to S-Shift, our work extends previous work in several other ways: We consider both writeback and writethrough L1 caches (such as those found in some modern processor designs); We consider several practical

[1] Alternatives that do not use writeback caches exist [6].

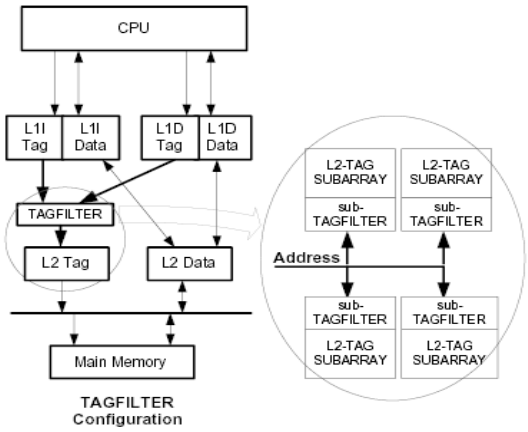

Figure 1: *Cache hierarchy with ReCast. Shown in detail is the ReCast organization where it is distributed along each tag array.*

ReCast organizations where the tag filters are partitioned alongside each tag array sub-bank; We report the results of a sensitivity analysis demonstrating that ReCast is a robust technique that can be potentially useful for higher level caches such as the L3; Finally, we consider the effects of out-of-order speculative execution and take into account a more diverse set of applications and input data sets.

The rest of this paper is organized as follows: In Section 2 we present ReCast and the S-Shift indexing function. In Section 3 we present our experimental analysis of ReCast. We start by explaining our methodology and proceed to consider the impact of ReCast on L2 power, miss rate and overall performance. In Section 3.8, we discuss three representative applications and relate ReCast behavior to program behavior and structures. In Section 4, we review related work. We offer concluding remarks in Section 5.

2 ReCast

ReCast builds on the simple and effective technique where a small number of *line buffers* are placed in-front of a much larger and thus much more power demanding cache. The operation of line buffers has been described numerous times in previous work, e.g., [7]. The line buffers collectively form a *tagfilter*. Figure 2 shows a direct-mapped tagfilter organization for the L2 tags while Figure 3 explains how tagfilters impact the L2 cache access sequence. We will use the terms ReCast, tagfilter and line buffers interchangeably in the rest of this paper. Table 1 summarizes the possible combinations of ReCast and L2 hits and misses along with the corresponding impact on power and latency.

Table 1. Breakdown of ReCast operation and its impact on L2 power and latency.

ReCast	*L2*	*L2 Tag Access*	*Latency*	*Power*
Hit	Hit	No	Unchanged/Reduced	Reduced
Hit	Miss	No	Unchanged/Reduced	Reduced
Miss	Hit	Yes	Increased	Increased
Miss	Miss	Yes	Increased	Increased

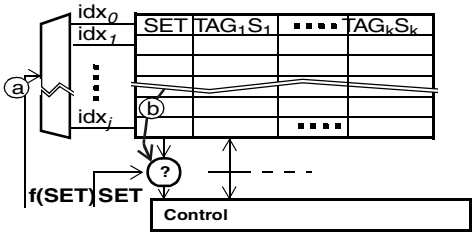

Figure 2: *(a) A direct-mapped ReCast. It caches the TAG and STATUS bits for a few sets of a k-way associative cache.*

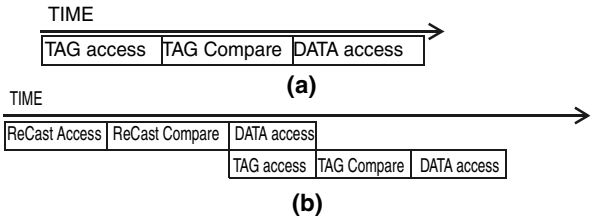

Figure 3: *(a) Conventional Cache Access Sequence: Using the set portion of the incoming address we first access the tags. We access all tag ways simultaneously reading all tag and status bit pairs (step 1). We then compare the stored tags with the tag portion of the incoming address taking the status information into account (step 2). If a match is found we access the corresponding data array using the set portion of the incoming address (step 3). Since we know which tag matched, we need to activate only one of the data arrays. If no match is found, we select one of the blocks for replacement and update the tags accordingly (step 3).*
(b) Access Sequence with ReCast: Prior to accessing the tag arrays, we access the ReCast (step 1). If the set we are looking for is found in the ReCast (a ReCast hit) we can determine whether the access is a hit or miss and we do not need to access the tag arrays. Power is reduced since ReCast is much smaller than the tag array. For the same reason, overall latency may be lower. If the set is not found in ReCast (a ReCast miss), then the access proceeds as in the conventional cache. In this case, power requirements increase as we accessed the ReCast and the regular tag array. Assuming that the ReCast and the tags are accessed serially, the cache latency is increased by ReCast's latency.

2.1 Improving Set Locality: S-Shift Indexing

For a small tagfilter to service many L2 tag probes it is necessary for the L2 tag reference stream to exhibit reasonably high locality. While most programs exhibit locality in their memory reference stream, it is not necessary that this locality will be visible at the L2. This is because the L2 sees only a fraction of the memory stream consisting primarily of those references that do not hit in the L1. In fact, by design the L2 sees those references that did not have sufficiently locality to be serviced by the L1 or that the L1 failed to satisfy for secondary reasons such as conflicts. A key observation to be made is that ReCast requires locality in the *L2 set* reference stream and not necessarily in the reference stream.

The main question we ask in this work is whether *it is possible to increase set locality as seen by the L2 further reducing power.* Answering this question requires an

understanding of what causes set locality. Two main factors give rise to set locality: program reference behavior and typical block sizes. L2 blocks are typically larger than L1 blocks. Programs exhibit spatial locality which often extends beyond the boundaries of L1 blocks. L1 conflict misses also give rise to L2 set locality as they often translate into accesses to the same L2 set. The aforementioned observations suggest that a straightforward technique for increasing set locality would be to increase the L2 block size so that more consecutive addresses map to the same L2 set. Unfortunately, well understood trade-offs apply as miss rates tend to increase beyond a specific L2 block size due to under utilization of L2 space and because of the overheads of bringing in extraneous data. An increased L2 miss rate may result in increased power consumption and reduced performance.

Our solution to increased set locality builds on the observation that it is possible to make consecutive L1 cache blocks map onto the same L2 set by changing the L2 indexing function. One way for doing so is the *S-Shift* indexing function shown in Figure 4(a) where *S* is a number of bits, the higher the *S* the more L1 blocks map onto the same L2 set. At the top we show how the TAG, SET and OFFSET indexes are extracted in conventional cache organizations. S-Shift works by changing the bits used to calculate the TAG and SET indexes so that a larger number of consecutive addresses map to the same set while the L2 block size remains the same. This is done by rotating right the combined TAG and SET fields. 0-Shift corresponds to the conventional index function. With 1-Shift and assuming 32byte L1 blocks and 64byte L2 blocks, four consecutive 32-byte blocks map onto the same set in L2. With 2-Shift eight such blocks would map onto the same L2 set. Figures 4(b) and 4(c) show how a sequential access pattern would map onto a 4-way L2 with 0-Shift and 1-Shift respectively. The desirable effect is possible with other permutations of the TAG and SET fields. However, an investigation of this topic is beyond the scope of this work.

2.1.1 S-Shift and L2 Miss Rate. The interaction of S-Shift with the memory reference stream is quite complex and the L2 miss rate may increase or decrease. In Sections 3.4 and 3.5 we demonstrate that the hit rate vs. power savings trade-off is favorable for the 1-Shift indexing scheme. Here we offer two remarks regarding the complex trade-offs that apply. First, for programs that exhibit irregular access patterns there is no noticeable increase in set pressure with any of the indexing schemes. This is expected as S-Shift is just a permutation of the address bits. Second, while for large sequential access patterns S-Shift results in higher set pressure, there are two mitigating factors: a) L2's associativity can often absorb most of this pressure and b) often such access patterns result in L2 misses. The trend in L2 design is towards higher associativities, accordingly, we should expect that S-Shift indexing would become more attractive in future designs.

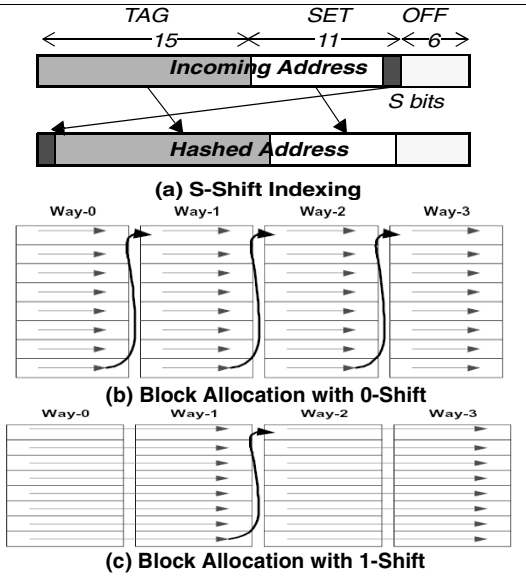

Figure 4: *(a) The S-Shift set index function that aims at increasing set locality in the L2. We assume 32-bit addresses, 64-byte L1 blocks, 64-byte L2 blocks and 2K L2 sets. We start with the incoming address from the L1. We take S bits (dark gray) from the least significant part of the set index and move them to the most significant part of the address. We shift the remaining part of the incoming address to fill in the remaining bits. As a result, a number of consecutive memory blocks (L1 block size) map to the same set in L2. (b) & (c) How consecutive addresses are mapped onto cache sets with the conventional indexing scheme (b) and 1-Shift (c). With conventional indexing addresses first fill up the first way, before starting to occupy the second and so on. In (c) with 1-Shift, consecutive addresses fill the rows of two ways simultaneously. With 2-Shift they would have filled the rows of four ways simultaneously.*

2.1.2 ReCast Updates and Organization. ReCast can aggregate tag updates the same way a writeback cache aggregates data updates. In this case, the updated information is propagated to the L2 tags only when the corresponding ReCast line is evicted. Since this could complicate the design of the ReCast and of the cache controllers we assume a write-through organization. The write-back and write-through options also apply to other state that is held within a typical cache, such as the per set LRU information.

Other ReCast organizations are possible since this is essentially a tag set cache. For example, we could have a 4-way set associative ReCast. While higher associativity may allow us to capture a larger fraction of accesses at the same time it also increases the per access energy requirements. Accordingly, we must properly balance the increase in filter rate vs. the per access energy requirements. Figure 1(a) shows the organization of the ReCast. The ReCast is partitioned into several banks one per sub-array of the L2 tag array. This way we avoid having to ship out the whole tag set from the each tag sub-array to a unified ReCast (something that would require extra wires and would impact power and latency).

2.1.3 ReCast Latency. In the simplest organization ReCast is placed in-series with the L2 tags so that a constant latency penalty is incurred by each access. However, it may be possible to improve overall L2 latency on ReCast hits since ReCast's latency should be smaller than that of the regular L2 tags. In this organization, rather than incurring a fixed latency penalty accesses see different latencies depending on the tag structures accessed. A potential disadvantage of this method is that it introduces greater variability in load latencies.

3 Experimental Analysis

In Section 3.1 we present our methodology. In Section 3.2 we demonstrate that typical programs exhibit reasonably high L2 set locality. In Section 3.3 we confirm that even small ReCast filters partitioned along the L2 banks can service many L2 tag accesses. In Section 3.4 we demonstrate that S-Shift can greatly increase L2 set locality. In Section 3.5 we demonstrate that ReCast can significantly reduce L2 power. In Section 3.6 we summarize the findings of a sensitivity analysis of key design parameters. In Section 3.7 we show how overall performance is affected by ReCast. Finally, in Section 3.8 we discuss three representative applications and how they affected by S-Shift.

3.1 Methodology

We used Simplescalar v3.0 [4] to simulate the processor detailed in Table 2. We used the following SPEC CPU 2000 benchmarks: 164.gzip, 175.vpr, 176.gcc, 177.mesa, 179.art, 181.mcf, 183.equake, 188.ammp, 197.parser, 255.vortex, 256.bzip2 and 300.twolf. We simulated up to the first 30 billion committed instructions or to completion whichever happened first. We compiled all programs for the PISA architecture using GNU's gcc v2.7 (flags: -O2 -funroll-loops -finline-functions).

Table 2. Base processor configuration

Branch Predictor	**Fetch Unit**
16k GShare +16K bi-modal 16K selector 2 branches per cycle	Up to 8 instr. per cycle 64-entry Fetch Buffer Non-blocking I-Cache
Issue/Decode/Commit	**Scheduler**
any 8 instr./cycle	128-entry/64-entry LSQ
FU Latencies	**Main Memory**
same as MIPS R10000	Infinite, 100 cycles
Cache Geometries	**L1D/L1I/L2/UL3 Latencies**
See text	3/3/16/30 cycles

We used CACTI [13] to determine the optimal number of cache sub-arrays for a 0.10um process. To model the serialized L2 cache access, we optimized the access delay of the tag and data paths separately and modified the L2 tag array model to appropriately account for bitlines, wordlines and sense amps. To measure power dissipation at the architectural level, we used the Wattch framework [3]. We made several modifications to Wattch in order to track and model the power of read and write operations separately, to track invalidations, replacements and writebacks, to serialize accesses to the tag and data arrays and to sub-bank the data array. Unless otherwise noted, the base memory hierarchy consists of split level-one data (LID) and instruction (L1I) caches and unified level-two (L2) and level-three (UL3) caches. The L1I and LID caches are 32 Kbytes each with 32-byte blocks and are 2-way set-associative. The L2 is 1Mbytes with 64-byte blocks and is 8-way set-associative. The L2 tags are partitioned into eight sub-arrays to minimize latency. The L2 data array is partitioned into 32-byte sub-banks to minimize its power dissipation. The UL3 is 4-Mbytes with 64-byte sub-banked 128-byte blocks and is 8-way set-associative. All caches use the LRU replacement policy.

3.2 Set Locality

To measure L2 set locality we simulated fully-associative tagfilters of four through 128 entries with LRU replacement. We define *filter rate* as the probability that an L2 tag probe finds its set in the tagfilter. The filter rate is a metric of locality since a hit in a tagfilter of *n* entries suggests that the same set has been accessed within the last *n* unique L2 sets accessed. The smaller the *n* the higher the L2 set locality. The results are shown in Figure 5. The average filter rates are 25% to 42% for four through 128 entries respectively. Although the filter rates vary per program, significant locality can be observed even with the smaller organizations. For example, a 64 entry tagfilter can reduce the L2 tag accesses by 34%.

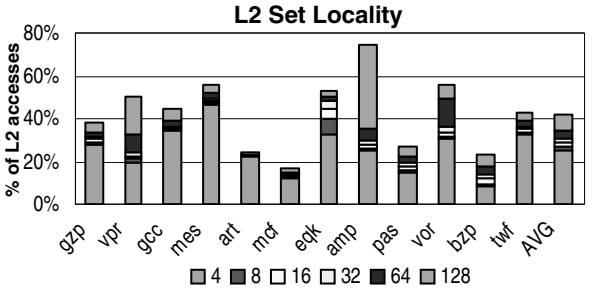

Figure 5: *Measuring L2 set locality: Shown are the filter rates for tag set buffers of 4 through 128 entries in power of two steps.*

3.3 Partitioned ReCast Organizations

Figure 6 shows filter rates with 0-Shift for various practical ReCast organizations consisting of eight sub-tagfilters, one per L2 tag sub-array. We use an *SxW* naming scheme where *S* is the number of rows and *W* is the associativity of each sub-array. There are *8xSxW* entries in total. Even small ReCasts achieve high filter rates which are on average 29% through 42% for the 4x1 through 16x2 organizations respectively. In the rest of this evaluation we will focus on the "4x2" organization since it offers a favorable size vs. filter rate ratio.

3.4 S-Shift: Locality and Miss Rate

Figures 7(a) and 7(b) report filter rates and the L2 miss rate respectively with the 4x2 ReCast and for three indexing schemes: 0-Shift (conventional), 1-Shift and 2-Shift. The average filter rates are 32%, 50%, and 60% for 0-, 1-, and 2-Shift respectively. Significantly higher filter rates are achieved with 1- and 2-Shift. Focusing on filter

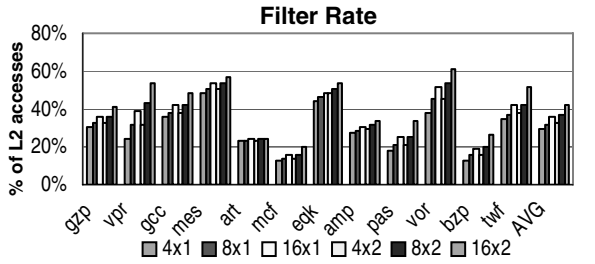

Figure 6: *Filter rates with* partitioned *ReCast organizations (0-Shift). The tagfilters consist of eight sub-arrays one per sub-array of the L2 tags. We use an SxW naming scheme where S is the number of rows and W is the associativity of each sub-array. The total number of entries is 8xSxW.*

rate alone we would conclude that 2-Shift is best. However, in some cases the increase in filter rate comes at the expense of a significant increase in miss rate. A higher miss rate implies additional L3 accesses (we do take into account the power overhead of these accesses in Section 3.5). Overall, the effect of S-Shift on the miss rate is varied. On average, the miss rate increases by 7% (relative not absolute increase) with 1-Shift and by 26% for the 2-Shift scheme as shown in Figure 7(b). Four observations are in order: First, increased miss rates are mainly seen in art and twolf; Second, even in these applications we later show that overall power is reduced because the power savings by ReCast are higher than the power overhead incurred due to additional L3 accesses; Third, in applications that exhibit irregular access streams such as gzip and gcc, there is virtually no change in the miss rate with 1-Shift; Fourth, in some applications the miss rate decreases with 1-Shift (e.g., ammp and equake).

For the 1-Shift scheme the miss rate increase is mainly seen in art and twolf (51% and 19% respectively). The initialization of art uses a poor allocation strategy and as a result the main algorithm suffers numerous, otherwise unnecessary cache misses. It is straightforward[1] to improve art's allocation strategy and then 1-Shift outperforms 0-Shift both in terms of the miss and the filter rate (more on this in Section 3.8). The L2 miss rate of twolf is very low, 0.21%, so a small absolute miss rate increase, 0.04%, results in a large relative increase. In the rest of this paper we focus on 1-Shift.

3.5 Power Savings

We report *power savings* with the 4x2 ReCast using the 1-Shift indexing scheme and the writethrough L2 tag update policy on ReCast replacements. In calculating the power savings we take into account any additional power overheads or savings resulting from changes in the L3 access frequency. We also factor in the power overhead of the ReCast itself. The results are shown in Figure 8(a) for a writeback L1D. We report *power savings* as a percentage of the original L2 tag array (left, light bar) and of the overall L2 power (right, dark bar). The latter includes the

[1] Of course, changing application code is rarely an option.

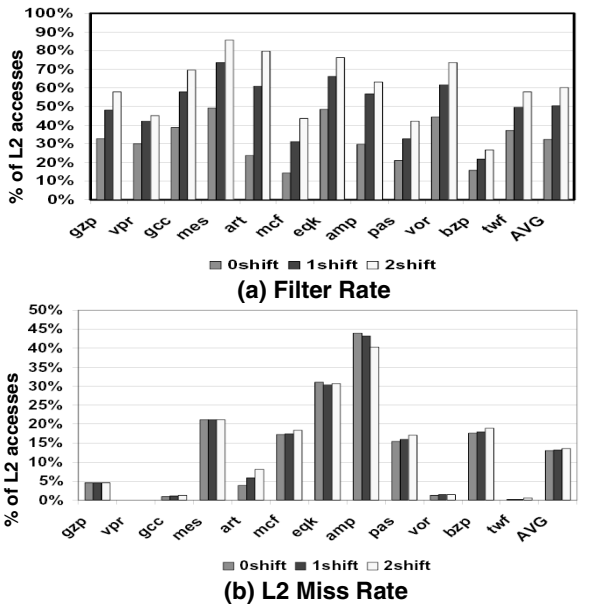

Figure 7: *Using the alternate indexing schemes 1-Shift and 2-Shift with the 4x2 ReCast. (a) ReCast filter rates. (b) L2 miss rates. We use the "4x2" ReCast in these experiments (64 entries in total).*

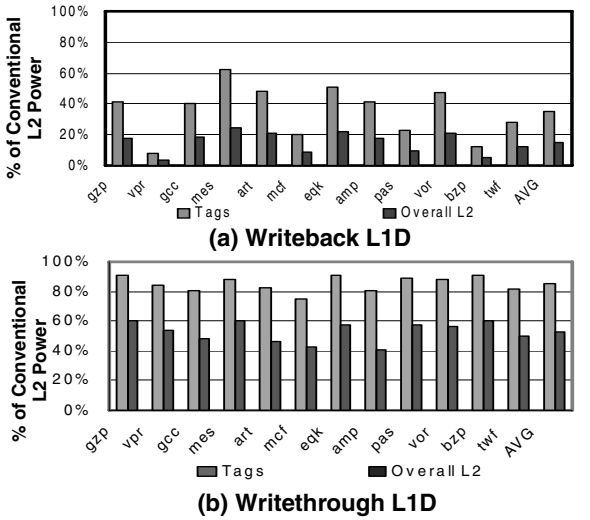

Figure 8: *Relative tag (light bars) and overall (dark bars) L2 power savings compared to the conventional L2 design.*

power dissipated by the L2 data array also. Compared to the original tag hierarchy, the ReCast offers 38% savings. (We note that a writeback ReCast organization resulted in a 43% average reduction in power.) On average, the new L2 tag array dissipates 55.7% of the original L2 tag array power and the ReCast dissipates 6.3% of the original L2 tag array power. These translate into a 16% reduction in overall L2 power.

In Figure 8(b) we report power savings for a writethrough L1D cache. In this case, the average power savings with ReCast are much more pronounced reaching

85% over the original L2 tag array and 52% in overall L2 power.

3.6 Sensitivity Analysis

We have performed several experiments to determine how sensitive ReCast's performance is to various memory hierarchy parameters. Due to space limitations we do not present these results in detail. We note that increasing the L1 size resulted only in a minor decrease in ReCast filter rate (e.g., less than 2% decrease on the average for 64K L1 caches). Increasing the L1 associativity had virtually no effect on the filter rate (less than 1% change on the average). The same was observed when we increased the L2 size up to 8Mbytes. Increasing the L2 associativity resulted in a minor increase in the filter rate (about 3% on the average). We have also experimented with using ReCast in the L3 and found that there too it offers a significant reduction in the L3 tag probes. Specifically, we have found that a 128 entry ReCast appropriately partitioned for our L3 configuration can filter about 63% and 70% of all L3 tag probes on the average and for the 0-Shift and 1-Shift indexing schemes respectively.

3.7 Performance

ReCast being an additional level in the tag hierarchy impacts L2 latency and hence overall performance. In this section we study the performance impact of ReCast. As per the discussion of Section 2.1.3, we consider two possible ReCast access models *fixed* and *variable*. Under the fixed model, ReCast increases L2 latency by a fixed number of cycles for all accesses. Under the variable model, L2 latency varies depending on whether we have a ReCast hit (decrease) or miss (increase). Given that each ReCast bank is very small (eight entries) we expect to be able to complete a ReCast access within a processor cycle. However, in order to expose the underlying latency vs. performance trend we report results for various latency overheads.

Figure 9(a) shows relative performance with the fixed model. Shown is performance relative to a conventional L2 with a 15 cycle latency (our base configuration has a 16-cycle L2). We consider penalties of one through six cycles (reported along the X-axis is the overall L2 latency). Programs form two groups. In the first are mesa, bzip2, ammp and equake that are mostly insensitive to an increase in L2 latency. The other applications exhibit a linear decrease in performance as L2 latency increases. For most programs this performance decrease is relatively small (less than 1% for each additional cycle of L2 latency. The only program that is very sensitive to L2 latency is mcf where each additional cycle results in an about 2% decrease in performance.

Figure 9(b) reports performance under the variable model. Performance is reported relative to a conventional L2 with a 16 cycle latency. We consider several options marked as *HL-ML* where HL and ML are the L2 latency decrease and increase respectively on a ReCast hit and miss respectively. As expected performance degradation is lower even when we assume that ReCast never reduces overall L2 latency (0-X configurations). We expect that the latency overhead of ReCast to be a single processor cycle on a miss and a latency reduction of one cycles on a hit. Under these assumptions the overall performance impact of ReCast is below 0.6% for all applications.

Assuming that it is possible to obtain a 3% reduction in overall power by reducing frequency and hence performance by 1%, it follows that overall chip power should be reduced more than 1.8% for ReCast to be viable. We have found that this is the case with writethrough L1 caches and only in some cases with writeback L1 caches. This analysis ignores the numerous power reduction techniques that have been proposed for other processor structures. Accordingly, further investigation may demonstrate that ReCast is viable even with writeback L1 caches.

3.8 1-Shift: Application Analysis

We take a closer look at the memory behavior of art, ammp and mesa to demonstrate that the 1-Shift indexing scheme may impact miss and filter rates either way. We chose these three applications for two main reasons. A key consideration with S-Shift is whether the increase in tag set locality in ReCast comes at the expense of a disproportionate increase in L2 miss rate. Accordingly, we wanted to study applications that are representative of the various behaviors we have seen. Specifically, all three applications benefit greatly in filter rate terms. However, the L2 miss rate in art increases, it remains practically unchanged in mesa and it decreases in ammp. The second consideration in choosing these applications was a practical one. We chose applications whose memory accessing behavior could be characterized with reasonable effort and that could be presented concisely.

3.8.1 Art. Art exhibits the highest relative increase in miss rate of 51% with the 1-Shift scheme which is a direct result of a poor memory allocation strategy during the initialization. It allocates two arrays, *tds* and *bus*, in the heap in such a way that their elements are interleaved. However, these elements are never accessed at the same time. Together with another array, *flJayer*, the two arrays occupy a heap area of about one megabyte as shown in Figure 10(a). The core program behavior is to process either flJayer area (1) and tds (3) or flJayer area (2), and bus (3). With 0-Shift (the left side in Figure 10(a)), these actions require mostly eight blocks per set and nine blocks for some sets. With 1-Shift (the right side in Figure 10(a)), these actions still require mostly eight blocks per set, but for some sets 10 blocks are required because two blocks per set are accessed simultaneously. This situation is represented as 128 bytes in a dotted circle (two blocks) on the 1-Shift side. This results in additional conflict misses with 1-Shift and thus in an increased miss rate from 3.89% to 5.89%. Applying a straightforward optimization (i.e., first allocate all the elements of *tds* and then all the elements of *bus*) alleviates the aforementioned problem. The particular modification is important even for a conventional cache (0-Shift) and at the source code level

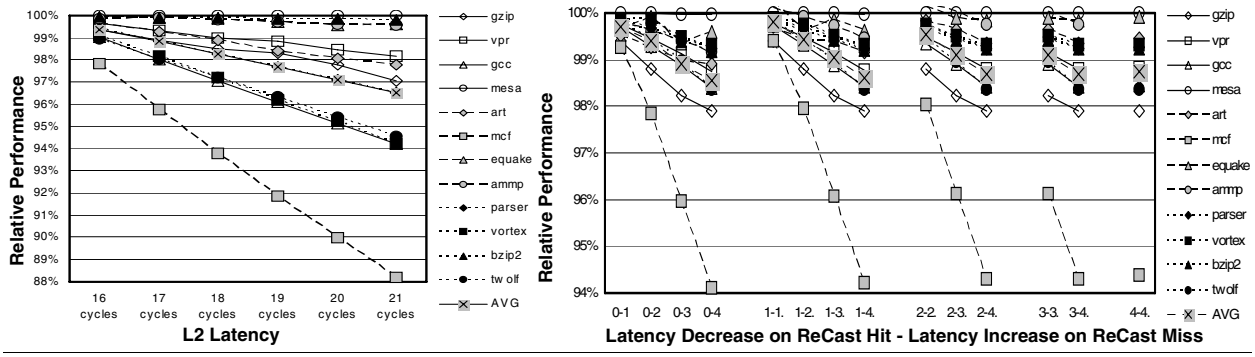

Figure 9: *Relative performance with a ReCast filter. (a) Constant latency penalty. In this model we assume a constant penalty for accessing the ReCast filter. Reported is performance relative to an L2 latency of 15 cycles. (b) Variable latency penalty depending on whether the access hits in the ReCast filter. Reported is performance relative to an L2 latency of 16 cycles. Several configurations are shown marked as HL-MS where HL is the latency decrease when the access hits in ReCast and ML is the increase in latency when the access misses in the ReCast.*

amounts to defusing a loop body into several separate loops. We divided the two arrays so that each occupied a consecutive space in the heap (160KB each). Figure 10(b) depicts the memory footprint after the modification. Now, with the 0-Shift scheme (the left side in Figure 10(b)), processing requires mostly seven blocks per set. With 1-Shift (the right side in Figure 4.5(B)) processing requires six blocks for some sets and eight blocks for others because areas (3) and (4) are not accessed simultaneously. This modification decreased the L2 miss rate from 3.21% (with 0-Shift) to 3.05% (with 1-Shift). Of course, changing an application to better fit our mechanism is not without cost and in some cases not even feasible. What our analysis has shown is that art is not written to work well even with conventional caches. We have also shown that a cache-aware optimization results in improved behavior with our mechanism also.

3.8.2 Ammp. In ammp the L2 miss rate decreases with 1-Shift. Most of the memory references in ammp are characterized by large strides. Specifically, it accesses the L2 cache in such a way that only one in every 32 sets (out of the total 2048 sets) is used heavily while the intervening sets are rarely touched. With 1-Shift the L2 is utilized better as now one set in every 16 is heavily used. The alternate indexing scheme has a positive effect of spreading out L2 cache accesses into more sets, thereby reducing conflict misses.

3.8.3 Mesa. Mesa exhibits the highest filter rate amongst all applications. It allocates a 1280 by 1024 frame buffer on the heap which it then accesses sequentially. Each frame buffer element is four bytes hence the frame buffer is about 5 Mbytes in total. With 32-byte L1D blocks, one in every eight accesses to the L1D is a miss and hence goes to the L2. Hence the L2 sees a sequential access stream over a large region. Using S-Shift does not impact the miss rate which comprises mostly capacity misses. A ReCast with 0-Shift filters out half the L2 accesses with 64-byte L2 blocks (50% filter rate). The 1-Shift ReCast filters out three quarters (75% filter rate) because two consecutive blocks share the same set.

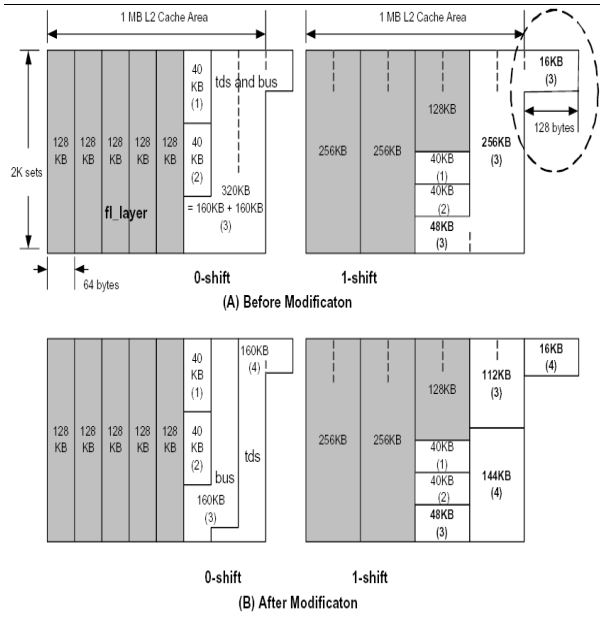

Figure 10: *Art L2 footprint. (a) Original application. (b) After a straightforward modification to the allocation sequence. See text for a discussion of the modification.*

4 Related Work

The idea of using a small cache to service some of the requests to a much larger one has been applied in many contexts in the past. Specifically, *line-buffers* have been proposed for increasing L1 bandwidth [19]. The *filter cache* is a small cache that is placed in-front of the L1 where it services a large fraction of the L1 accesses [9]. Accesses that miss in the filter cache incur increased latency. The idea of using small buffers in-front of the tag and data arrays for the L2 so that power is reduced has been studied before in [7]. Our work differs primarily in that it proposes the S-Shift indexing function that results in a significant increase in filter rate. Furthermore, our work considers the effects of speculative, out-of-order execution (which improve the utility of ReCast), it

considers alternate tag array configurations and it takes into account a more diverse set of applications. *Cool-Mem* leverages compiler information to reduce power in the L1 caches including its tag array [20]. Partial tag matches have also been proposed for reducing complexity [8] or power [5]. The *CAT* caches a common prefix for most tags to reduce area [18]. Conventional cache hierarchies were designed solely for high performance: reducing access latencies and miss rates. Many architectural techniques have been proposed to reduce on-chip cache power dissipation, e.g., [2, 7, 9, 10, 11, 12, 14, 17, 16, 18, 21, 22, 23]. Reviewing these works is beyond the scope of this paper. We briefly state that in their majority these techniques involve partitioning caches vertically and/or horizontally, possibly using small auxiliary structures that collect program behavior related information, thereby activating the smallest chunk of the arrays.

5 Conclusion

We studied *ReCast* a technique for optimizing power in the tag arrays of high-level caches. ReCast is a tag set cache placed between the LI and the L2 tag arrays where it caches a small number of recently accessed L2 tag sets. ReCast acts as a filter preventing L2 tag accesses and thus it reduces power dissipation in the L2 tag array. The key contribution of this work is S-Shift a simple and effective indexing scheme that greatly enhances the utility of ReCast at zero additional cost. We have shown that typical L2 access patterns exhibit high locality in their set stream and that practical (i.e., small and partitioned) ReCast organizations can exploit most of this locality to avoid accessing the L2 tag array. We demonstrated that a practical ReCast of 64-entries partitioned into eight sub-banks of eight entries each and that uses our *S-Shift* indexing scheme can filter about 50% of all L2 tag probes on the average. This translates to a 38% reduction in power compared to the conventional L2 tag array organization for a writeback L1 and to an 85% reduction in power for a writethrough L1. In terms of overall L2 power, we demonstrate an average reduction of 16% and 52% for the writeback and writethrough L1 caches respectively. The overall impact on performance was found to be negligible.

Acknowledgements

This research was supported in part by the Semiconductor Research Corporation under contract 901.001, an NSERC Discovery Grant, an NSERC Equipment Grant and a Canada Foundation for Innovation Equipment Grant.

References

[1] B. Bateman, C. Freeman, J. Halbert, K. Hose, and E. Reese. *A 450Mhz 512KB Second-Level Cache with a 3.6GB/S Data Bandwidth.* In the Proc. of the IEEE International Solid-State Circuits Conference, 1998.

[2] N. Bellas, I. Hajj, C. Polychronopoulos, and G. Stamoulis. A*rchitectural and compiler support for energy reduction in the memory hierarchy of high performance processors.* In the Proceedings of the International Symposium on Low Power Electronics and Design, Aug. 1998.

[3] D. Brooks, V. Tiwari, and M. Martonosi. *Wattch: A framework for architectural-level power analysis and optimizations.* In Proc. of the 27th Annual International Symposium on Computer Architecture, June 2000.

[4] D. Burger and T.M. Austin. *The SimpleScalar Tool Set, Version 2.0.*Technical Report UW-CS-97-1342. Computer Sciences Department, University of Wisconsin-Madison, June 1997.

[5] Y.-J. Chang, S.-J. Ruan and F. Lai, *Design and analysis of low-power cache using two-level filter scheme,* IEEE Transactions on VLSI, vol 11, no. 4, Aug. 2003.

[6] R. Espasa, F. Ardanaz, J. Gago, R. Gramunt, I. Hernandez, T. Juan, J. S. Emer, S. Felix, G. Lowney, M. Mattina, A. Seznec. *Tarantula: A Vector Extension to the Alpha Architecture.* In the Proceedings of the 29th Annual International Symposium on Computer Architecture, June 2002.

[7] M.B. Kamble and K. Ghose. *Reducing Power in Superscalar Processors using subbanking, multiple line buffers and bit-line segmentation.* In the Proceedings of the International Symposium on Low Power Electronics and Design, 1999.

[8] R. E. Kessler and R. Jooss and A. Lebeck and M. D. Hill, *Inexpensive implementations of set-associativity*. In the Proceedings of the 16th Annual International Symposium on Computer Architecture, 1989.

[9] J. Kin, M. Gupta, and W. Mangione-Smith. *The Filter Cache: An Energy Efficient Memory Structure*. In the Proceedings of the 30th International Symposium on Microarchitecture, pages 184-193, Nov. 1997.

[10] Uming Ko, Poras T. Balsara, and Ashwini K. Nanda. *Energy Optimization of Multi-Level Processor Cache Architectures.* In the Proc. of the International Symposium on Lower Power Design, Aug. 1995.

[11] H.S. Lee and G.S. Tyson. *Region-Based Caching: an energy-delay efficient memory architecture for embedded processors.* In the Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pages pp. 120-127, Nov. 2000.

[12] R. Panwar and D. Rennels. *Reducing the frequency of tag compares for low power I-Cache design.* In the Proceedings of the International Symposium on Low Power Electronics and Design, Aug. 1995.

[13] G. Reinman and N.P. Jouppi. *An Integrated Cache Timing and Power Model. Technical report,* COMPAQ Western Research Lab, 1999.

[14] C. Su and A. Despain. *Cache Designs for Energy Efficiency. I*n the Proceedings of the 28th Annual Hawaii International Conference on System Sciences, pages 306-315, 1995.

[15] W.J. Bowhill et al. *Circuit Implementation of a 300Mhz 64-bit Second Generation Alpha CPU.* Digital Journal vol. 7., 1995.

[16] D. H. Albonesi. *Selective cache ways.* In the Proceedings of the 32nd Annual International Symposium on Microarchitecture, Nov. 1999.

[17] U. Ko, P. T. Balsara, and A. K. Nanda. *Energy Optimization of Multilevel Cache Architectures for RISC and CISC Processors.* In the Proceedings of the International Symposium on Low Power Electronics and Design, Aug. 1998.

[18] H. Wang, T. Sun, and Q. Yang. *CAT – caching address tags: A technique for reducing area cost of on-chip caches.* In the Proceedings of the 22nd Annual International Symposium on Computer Architecture, June 1995.

[19] K. M. Wilson, K. Olukotun, and M. Rosenblum. *Increasing cache port efficiency for dynamic superscalar microprocessors*. In the Proceedings of the 23rd Annual International Symposium on Computer Architecture, May 1996.

[20] Raksit Ashok, Saurabh Chheda, Csaba Andras Moritz, *Cool-Mem: Combining Statically Speculative Memory Accessing with Selective Address Translation for Energy Efficiency,* In the Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 2002

[21] K. Inoue, T. Ishihare and K.Murakami, *Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption,* In the Proceedings of the International Symposium on Low-Power Electronic Design, August 1999.

[22] M. Huang, J. Renau, S.-M. Yoo and J. Torellas. *L1 Data Cache Decomposition for Energy Efficiency.* In the Proceedings of the International Symposium on Low-Power Electronics and Design, Aug. 2001.

[23] M.D. Powell, A. Agarwal, T. N. Vijaykumar, B Falsafi and K. Roy, *Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping,* In the Proceedings of the 34th Annual Symposium on Microarchitecture, Dec. 2001.