Embedded Way Prediction for Last-Level Caches†

Faissal M. Sleiman, Ronald G. Dreslinski, and Thomas F. Wenisch
Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, USA
sleimanf, rdreslin, twenisch@umich.edu

***Abstract*—This paper investigates *Embedded Way Prediction* for large last-level caches (LLCs): an architecture and circuit design to provide the latency of parallel tag-data access at substantial energy savings. Existing way prediction approaches for L1 caches are compromised by the high associativity and filtered temporal locality of LLCs. We demonstrate: (1) the need for wide partial tag comparison, which we implement with a dynamic CAM alongside the data sub-array wordline decode, and (2) the inhibit bit, an architectural innovation to provide accurate predictions when the partial tag comparison is inconclusive. We present circuit critical-path and architectural power/performance studies demonstrating speedups of up to 15.4% (6.6% average) for scientific and server applications, matching the performance of parallel tag-data access while reducing energy overhead by 40%.**

I. Introduction

Semiconductor device scaling continues to enable processor designs with ever-larger caches. As larger working sets are captured within the chip, the importance of intra-chip access latency has grown [6]. Server applications are particularly sensitive to last-level cache (LLC) latency because their multi-megabyte instruction footprints overwhelm primary instruction caches, exposing LLC latency on the fetch critical path [7]. While tag and data accesses in large, highly-associative LLCs are often serialized to save energy [18], parallel tag-data access (for reads) can reduce overall access latency by 30% albeit at a 1.47x cost in per-access energy.

To bridge the performance and energy gaps between these two extremes, we consider *way prediction*, where only a subset of data ways are accessed in parallel with the tags. Way prediction has been studied extensively for L1 caches [4], [9], [12], [14], [18], [21]. In this context, it has relied primarily on one of two phenomena: temporal locality (i.e., predict the most-recently-used way) or instruction-correlated locality (i.e., use a PC-indexed prediction table). However, way prediction is fundamentally harder in LLCs because associativity is greater, temporal locality is filtered by the L1 caches, accesses from multiple cores interleave, and instruction addresses are typically not available.

Alternatively, researchers have advocated *partial tag comparison* to rule out cache ways that surely do not contain the data [5], [11], [15], [22]. These designs compare a few low order bits of the incoming tag to those stored in each way, and abort accesses for any mismatches. The most recent design [22] targets small (8-32KB), low-associativity (4-way) L1 caches, which allows it to hide a fully-associative 4-bit partial tag comparison under the data array decoder delay. This design avoids any impact on the cache access critical path, while achieving good energy efficiency. The comparison is implemented with static logic as a content-addressable memory (CAM)—a design facilitated by the small L1 size.

We follow a similar approach, however our target LLC context leads us to a different solution. (1) We demonstrate the need for a wider partial tag comparison of 6-8 bits in order to achieve highly accurate way prediction (over 90% accuracy) at the LLC. (2) This wider comparison leads us to implement the CAM with dynamic logic to minimize latency, and we assess the impact on access latency by performing a circuit-level critical path analysis of the CAM versus decoder delays. (3) Despite a wider partial tag comparison, some accesses still result in partial tag matches in more than one way, which makes the way prediction inconclusive. To tightly limit energy per access, we describe an architectural feature we call the *inhibit bit* to predictively activate the most likely way under such *partial tag collisions*.

Integrating these components, we propose *Embedded Way Prediction*, an architecture and circuit design for effective way prediction in server-class LLCs. We show that embedded way prediction achieves the full potential performance improvement of parallel lookup, improving scientific and server application performance by up to 15.4% (6.6% average) at an energy-per-instruction overhead of 11%, as compared to a 17.5% overhead for conventional parallel lookup (averages are geometric means).

II. Background

Types of cache access. Latency-sensitive L1 caches typically adopt a *parallel* access scheme, shown in Figure 1 (left), wherein all ways of both the tag and data array are read concurrently, thus minimizing latency at the expense of energy efficiency. Conversely, L2 or LLC designs typically perform tag accesses first, followed by an access to the correct data way, as depicted in the *sequential* access scheme in Figure 1 (center). Sequential access saves the energy of accessing irrelevant data ways in larger, higher associativity caches, at the cost of latency. *Way prediction*, illustrated in Figure 1 (right), attempts to offer the best of both, by only accessing a subset of data ways in parallel with the tags.

We highlight two special cases of way prediction. The first is *way filtering*, where the cache access is nominally parallel, but data ways that are known not to contain the requested

†This work was partially supported by NSF CSR-0815457 and grants from ARM, Inc.

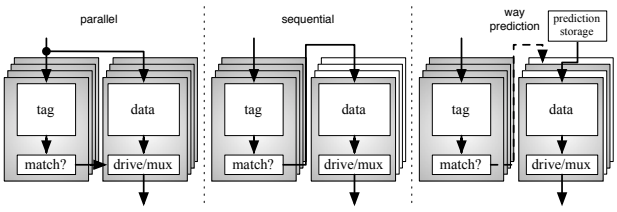

Fig. 1. Cache access schemes.

block are filtered out. Way filtering never incurs a performance overhead relative to parallel access, but may consume as much energy if filtering is not successful. The second is *single-way prediction*, where at most one data way is activated in parallel with the tags. We focus on this flavor of way prediction, as it assures that at most two data ways will be activated per cache access, bounding LLC access energy. In the common case that the prediction is correct, single-way prediction offers the low latency of parallel access at the low energy of sequential access. On a misprediction, the tag comparison triggers a second, sequential data access.

Partial tag matching. Although predicting the most-recently-used (MRU) way at the L1 is known to be 85-95% accurate [4], we find it is typically only 30-60% accurate at the LLC and sometimes little better than a random guess for multithreaded scientific and server workloads (see Section VI-C). Instead, to quickly and efficiently rule out ways that cannot contain the requested cache block, we advocate comparing the low-order bits of each stored tag to the incoming address, an idea known as *partial tag matching*. In the context of parallel caches, partial tag matching can implement way filtering by inhibiting access to ways that mismatch. Prior designs using partial tag matching in this manner have targeted L1 caches, as parallel access is not typically used in LLCs. The key challenge in such designs is engineering the partial tag match so that it has minimal impact on the data array critical path while saving as much energy as possible.

III. Related work

Our work builds on a long history of literature on way prediction, way filtering, and partial tag comparison; the earliest work in this area dates back over 20 years [11]. Broadly, our objective is to revisit these concepts in the context of modern servers because of their growing sensitivity to LLC access times. Our design accelerates LLC tile accesses in the common case of an accurate way prediction.

Way prediction was initially proposed as a performance enhancement to prearrange multiplexor paths at the output of cache data arrays to select the MRU way before the tag comparison is complete [14]. Subsequent work suggested using sources besides the replacement order, such as register or instruction addresses, to predict which way to access first in sequential associative caches (which access cache ways in consecutive cycles) [4]. Later work focused instead on saving energy by accessing a single predicted way in parallel caches [9], [18], predicting wake-up for Drowsy cache cells [12], or selective sub-array precharging [21]. In all of these designs, a key constraint is that the prediction must be made before the cache address is available, a constraint relaxed in LLCs.

Partial tag matching was first suggested by Kessler and co-authors to reduce the number of tags scanned sequentially in early set-associative caches, where tag comparators were expensive [11]. Over the past two decades, partial tag matching has been suggested in various forms as a means to reduce tag comparison energy in sequential caches, most recently using a partial tag bloom filter [17]. Min and co-authors use a partial tag match to gate sense amplification and bit line muxing [15]. Zhang and co-authors conserve nearly all of the data array access energy by performing the partial tag match in parallel with wordline decode, then gating wordline activation [22]. We pursue the same approach. However, whereas their study targets a small (8KB), low-associativity (4-way) L1, we target a comparatively massive (2MB) highly-associative (16-way) LLC tile in a multicore server, leading to a markedly different solution.

IV. Architectural Design

We propose *Embedded Way Prediction* in the context of a server-class chip-multiprocessor with a highly-associative large tiled last-level cache similar to designs from Tilera [2]. We consider both private and shared cache organizations. Similar to [22], we perform partial tag matching by embedding a CAM alongside the wordline decoders of the data SRAM sub-arrays. However, we find server workloads require a far wider partial tag comparison of 6-8 bits (see Section VI-C), which necessitates a dynamic CAM circuit to avoid timing impact. Furthermore, we target single-way prediction as opposed to way-filtering to limit energy per access.

A. Addressing Partial Tag Collisions

Because the partial tags are narrower than full tags, it is possible for the partial tags to match in several ways. These *partial tag collisions* lead to ambiguity as to which one among the matching ways should be predicted. To avoid the energy overhead of multiple way accesses (only one of which can be correct), we instead include an *inhibit bit* in each CAM entry that, when set, prevents that entry from reporting a match. We orchestrate the inhibit bits such that they are set for all but one colliding partial tag, and also use them to disable matches for invalidated lines.

A variety of policies might be used to choose which among a set of colliding tags should remain enabled. In our design, we use our LRU replacement policy as our guide, and clear the inhibit bit for the MRU tag within each collision set. We explore the impact of this scheme on prediction accuracy in Section VI-C. Inhibit bits could also be used for more complex schemes (e.g., using information from more sophisticated replacement policies [10] or confidence counters) or to provide software control over the use of embedded way prediction (e.g., to activate it only for blocks allocated by a particular core/thread). We note that embedded way prediction has no effect on the cache replacement algorithm or coherence protocol.

B. Maintaining Inhibit Bits

To ensure that only the MRU block within each collision set can trigger a parallel lookup, we maintain the following invariant: each time a cache block within a set is accessed or newly allocated (when a miss is filled), its inhibit bit is cleared, while the inhibit bits for any other block matching the same partial tag are set. Given this, at most two inhibit bits can change per cache access.

To avoid the need for a read port on the CAMs, the tag array stores a copy of the inhibit bit for each way (1 bit of overhead for every 32-bit tag). Rather than calculate and update inhibit bit state within the embedded way predictors at the data array, we instead rely on the tag array to maintain their state, sending updates to the CAMs when needed. We reuse the low order bits of the tag array's tag comparator to identify matching partial tags. With the information stored in the inhibit copies, we can identify which way was predicted within the data arrays. We can also identify if the prediction was correct, by checking it against the full tag comparison. At the tag array we then set the inhibit bit for all matching partial tags, except that we clear the inhibit bit for the hit way (if the access was a hit). The new inhibit bit state is driven to the data arrays along with the way select signal, and modified inhibit bits are written into the appropriate CAM entry.

On a misprediction, the (sequential) access to the correct way within the data arrays must override the partial tag comparison to ensure the word line is activated. Rather than add an override input to the CAM/decoder circuits, which would impact the critical path of one (or both), we solve this problem architecturally, by driving the partial tag comparison and inhibit match lines with the (known) content of the CAM. We make use of the fact that the inhibit bit already takes part in the CAM comparison: during a way prediction we clear the comparison lines to inhibit bits so that cleared inhibit bits can match. Now, during the sequential access, the comparison line is set for the hit way to force a match, while the rest of the ways are disabled by setting their inhibit bits to the opposite of their known values.

Finally we address cache replacements and invalidations that target the MRU matching partial tag. Depending on the LRU implementation, these events make identifying the next-most-recently-used matching partial tag ambiguous (for example, an approximate LRU implementation). In these scenarios, we clear the inhibit bit of an arbitrary other matching partial tag. We see in Section VI-C that the potential impact of a wrong choice in this situation is low.

C. LLC Tile Organization

Each LLC tile in our design is 2MB, 16-way set associative, and divided into 4 independently operating banks. Within a bank, tag and data pipelines are separately scheduled, to facilitate coherence traffic that often requires only tags. Each 512KB bank contains 512 sets of 16 ways each with 64B blocks. Within a bank, the tag and data arrays are further sub-divided into sub-arrays to optimize the latency-area-power trade-off. We assume a physical layout like that modeled by CACTI [16], where tag and data sub-arrays are grouped into *mats*: two-by-two squares of sub-arrays that share a common predecoder. The mats are interconnected via intra-bank H-trees, which in turn connect to the bank-level routing. As these interconnect busses are long and wide, they account for the majority of cache dynamic power (and, to a lesser degree, latency) for sequential accesses.

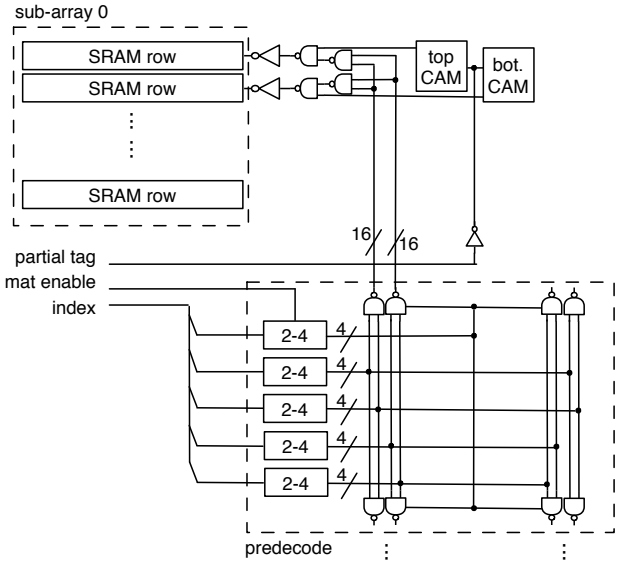

Fig. 2. Organization of a single data "mat".

We organize ways across mats such that each 32KB way occupies two 256x523 bit sub-arrays (512 data + 11 ECC) from the same mat. Our data arrays rely on ECC rather than bit interleaving to provide tolerance against soft error. Otherwise, bit interleaving would complicate partial tag matching because several data words (with different tags) share a single wordline. Hence, the way predictor would have to activate the wordline if any of the corresponding tags matched, requiring an OR function in addition to several CAM comparisons (one per interleaved word).

An arriving read that finds the data pipeline unscheduled forwards its partial tag and set index to all data mats/ways to initiate a way prediction. This in turn activates the CAMs and decoders for all data ways, which proceed to read out at most one uninhibited block with a matching partial tag. In the meantime, the result from the full tag comparison is used to confirm the prediction. On a misprediction, the correct data way is then activated, thus incurring an extra data way access relative to a sequential cache.

V. Circuit design

We study the effect of partial tag width on LLC way prediction accuracy in Section VI-C, and determine the need for 6-8 bits to achieve an accuracy over 90%. The crucial question then is to determine whether such a wide partial tag comparison can be hidden within the wordline decoder delay and to determine the energy requirements of the CAMs themeselves. In this section, we perform a critical path analysis of the partial tag comparison and wordline decoder circuits. We first describe the physical layout of our LLC bank, and

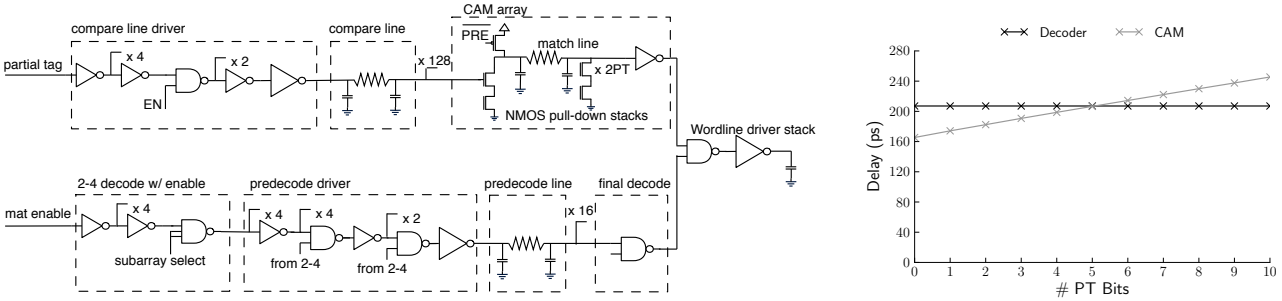

Fig. 3. Critical path analysis. Left: CAM (top) vs. Decoder (bottom) paths. Right: delay comparison.

finally provide energy estimates for parallel, sequential, and way-predicted cache accesses.

A. Data Mat Organization

Figure 2 shows the internal layout of a single data mat, including the additions required to support embedded way prediction. In a conventional sequential access, the tag arrays indicate which data way to access, and one of the sub-arrays within a single mat returns the data. The 9 index bits, along with an enable signal, arrive via the intra-bank H-tree at the predecoder, which comprises four 2-4 decoders, one of which is gated by the enable, a NAND combining stage which creates a one-hot encoding of the combined outputs of a 2-4 decoder pair, and drivers to transmit the predecoded index to the sub-arrays. A final NAND stage combines the predecode signals into wordlines, which are driven across the sub-arrays.

To support embedded way prediction, each data sub-array within the mat is augmented with a 256-entry CAM. Because a CAM cell requires two horizontal routing tracks, while modern SRAM cells have only one, we place two separately-driven 128-entry CAM arrays (top CAM and bottom CAM) side-by-side, with each CAM row spanning two SRAM rows. Note that the match line of one CAM must route over the other, requiring resources on an additional metal layer. When predicting, all 4 CAMs (in each of the 16 ways) operate in parallel with the mat predecoder and sub-array wordline decoders.

The need for wider comparison forces us to use a dynamic CAM circuit [13], rather than using the transmission-gate XOR-NOR comparator proposed in [22]. The 10T CAM cell performs the (mis)match operation by pulling down a precharged matchline through an NMOS stack. Indeed, Zhang et al. considered a dynamic CAM and rejected it because their static logic design is more energy efficient for small sub-arrays; we reach the opposite conclusion for LLCs.

B. Critical path analysis

Our objective is to establish the limit on the number of partial tag bits that can be compared within the timing constraint of the wordline decoder circuit for our chosen configuration. We optimize the critical paths of the decoder and CAM circuits using the method of logical effort, and investigate their timing using Cadence Virtuoso Spectre targeting an industrial 65nm process. For our target technology, the SRAM bit cell is 1.05μm by 0.46μm, and the CAM cell is 1.05μm by 0.92μm. We stop short of analyzing a complete mat layout, as this would need to consider additional factors beyond the scope of this study, including component-level energy trade-offs, reliability under process variations, and peripheral logic unrelated to embedded way prediction.

We begin our analysis with the assumption that the mat input signals arrive simultaneously. From there, the decoder and CAM circuits follow different critical paths before converging at the wordline driver stack, as depicted in Figure 3 (left). Optimizing decoder critical paths is discussed extensively in [1]. We only note here that the critical path of a decoder is complicated by the intermediate wire load of the predecode lines. We simplify the optimization process by following a heuristic adhered to in CACTI and suggested in [1], namely, to set the NANDs after the predecode lines to minimum size, thereby improving the energy-delay characteristic of the decoder. The resulting critical path of our decoder, illustrated in Figure 3 (bottom left) yields a decode time of 207ps for our 65nm process.

On the other hand, our CAM operates by broadcasting each bit to be compared (along with its inverse) to all rows. Since the highly regular CAM cells are constrained by stringent sizing and layout considerations, rearranging logic within the CAM cell is not possible. At best, the comparison line driver can be optimized to reduce the delay on the comparison line. Thus, as we increase the number of partial tag bits in the CAM, the load on the NMOS stack (which cannot be sized to compensate) also increases, while each comparison line driver continues to drive the same load.

Therefore, we see the delay of the CAM circuit degrade in Figure 3 (right) with increasing number of partial tag bits (CAMs also include one extra inhibit bit cell). Our results show that a CAM of width $5 + 1$ can be designed within the timing constraint of the decoder. However, even for slightly wider CAM widths, which are desirable for the embedded way prediction architecture, the CAM delay does not exceed that of the decoder by more than 20ps. For a CAM including 7 partial tag bits, needed for accurate way prediction at the LLC (Section VI-C), such an overhead is negligible compared to the overall cache access time (several nanoseconds).

C. Energy per Cache Access

Finally, we estimate the energy per access of sequential, parallel, and way-predicted (correct and mispredict) accesses, using estimates obtained from CACTI 6.5 and our characterization of the embedded CAM circuit using Spectre. We resort

TABLE I
PER-ACCESS DYNAMIC ENERGY IN NJ.

	Sequential	Way-Predict		Parallel
		Corr.	Mispr.	
Routing	0.8594	0.8594	0.8594	0.8594
H-Tree	0.5470	0.5470	0.5470	0.5470
Decoder	0.0003	0.0045	0.0047	0.0045
Subarrays	0.1016	0.1016	0.2033	0.8130
Tags	0.0119	0.0119	0.0119	0.0119
CAM	-	0.2051	0.2051	-
Total	1.5203	1.7295	1.8315	2.2359

to CACTI, rather than using the estimates available from an SRAM compiler, because CACTI provides a detailed energy breakdown across components of the cache bank, while the available SRAM compiler provides only a black-box total. We require the breakdown to accurately assess the overheads of embedded way prediction.

We configure CACTI 6.5 to target the 65nm ITRS-LOP process and constrain CACTI's search to generate a 2MB 4-bank cache with a physical layout conforming to the organization described in Section IV-C. We report the resulting energy breakdowns in Table I. We categorize the various components into inter-bank Routing, intra-bank H-Tree, data row Decoders, data Subarrays (precharge, bitlines, sense amps, and array-internal muxing and output drivers), Tags (all subcomponents), and CAM (all CAMs within a bank). The sequential access energy breakdown is taken directly from CACTI; the other estimates are formed by multiplying per-element energy by the number of elements activated during the given type of access. Based on CACTI's access time estimates, we determine a parallel access latency of 15 cycles and a sequential access latency of 21 cycles for a 4GHz clock assumption.

Like many large caches, the energy consumption of our design is wiring-dominated, except in the case of parallel access, where the concurrent accesses to all 16 data ways dwarf all other components. Overall, the per-access overhead of a successful way prediction is only 13.8%, and a misprediction is 20.5%. In contrast, a parallel access incurs 47.1% more total energy. We use these estimates to construct the full LLC power and energy estimates in Section VI.

VI. EVALUATION

We first establish the effectiveness of our final, tuned embedded way prediction design with 7 partial tag bits (denoted 7) relative to sequential (S), parallel (P), and per-set MRU-way-prediction (M) baselines. We then study the sensitivity to partial tag width, the impact of partial tag collisions, and the importance of including inhibit bits in the design. Our designs are denoted by the number of partial tag bits, and an additional inhibit bit is present in all cases. Finally, we examine energy and power implications.

A. Methodology

We evaluate the architectural impact of embedded way prediction on a suite of scientific and server applications using the Flexus full-system simulator [19]. We configure our processor model to approximate the hardware resources of recent Intel Xeon microarchitectures; details appear in Table II. We simulate a 16-core tiled chip multiprocessor with 32MB aggregate on-chip LLC capacity, composed of 16 per-core 2MB tiles with MOESI coherence. To study the generality of embedded way prediction, we examine two baseline organizations: 1) a *private* organization where each core queries and allocates blocks in its local tile, and 2) a unified *shared* organization where the address space is interleaved across tiles and a block resides in one location in the LLC for all queries and allocations from all cores. For each workload we present results for the highest performing baseline and apply our technique to it. We measure performance using the SimFlex multiprocessor sampling methodology [19].

TABLE II
SYSTEM CONFIGURATION & WORKLOADS.

Component	Configuration
Cores	16 OoO Cores @ 4.0 GHz 8-stage pipeline; 4-wide OoO 96-entry ROB, LSQ
Architecture	Ultra Sparc III ISA
L1I Caches	64KB, 2-way, 64B blocks
L1D Caches	64KB, 4-way, 64B blocks, 32 MSHRs
LLC Cache	Tiled, 2MB per-core private
LLC Tiles	16-way, 64B blocks 15-cycle parallel, 21-cycle sequential
Interconnect	2-D folded torus, 2-cycle router 1-cycle link latency
Directory	MOESI coherence 8K entries per tile (128K total) 16-way, 4-cycle latency
System Memory	3GB, 4KB pages, 150 cycle latency

In Section V we performed our circuit analysis targeting an industrial 65nm process (the latest process technology for which a design kit is available to us), in which a chip of this size is likely infeasible. However, we pursue these cache sizes and core microarchitecture to match the scale of our workloads and model a near-future server-class CMP. We configure our simulation with the cache latency and energy results derived in Section V.

We study the TPC-C v3.0 OLTP workload on IBM DB2 v8 ESE and Oracle 10g Enterprise Database Server. We also evaluate a selection of SPLASH2 [20] and PARSEC 2.1 [3] benchmarks that are sensitive to on-chip access latency. These are *barnes*, *moldyn* and *ocean* from SPLASH2, and the *canneal* benchmark from the PARSEC 2.1. We found that the two OLTP workloads *db2* and *oracle*, as well as *barnes* and *moldyn*, favor the private organization. On the other hand, *ocean* and *canneal* perform better on the shared baseline. The rest of this section presents these workloads running under their respective baselines.

B. Impact of Embedded Way Prediction

As shown in Figure 4, embedded way prediction with 7 partial tag bits (denoted 7) achieves the performance potential of parallel access (P). The figure shows speedup normalized

to a sequential access baseline. The error bars indicate 95% confidence intervals obtained by our sampling methodology, thus the apparent speedup of our design with respect to parallel access is not statistically significant. Figure 5 shows a normalized breakdown of cycles-per-instruction spent on various stall sources for a range of designs. The graph is normalized to sequential access (S), and includes the same three designs as Figure 4 (including a breakdown for the sequential baseline). The time breakdowns are broken into (from bottom to top) busy time, stalls on store instructions, stalls on L1D accesses, stalls for L1I misses, stalls on data accesses to the LLC, stalls on main memory and other stall sources.

Though prior work [4], [14] has shown that predicting the MRU way (M) is effective in low-associativity L1 caches, it underperforms in the highly-associative LLC, achieving an average 17% of the speedup potential of parallel access.

We find the two database applications are particularly sensitive to our scheme because it accelerates the many L1I misses serviced at the LLC, with up to 15.4% speedup for *db2*. The instruction footprints of these two applications (over 2MB [8]) overwhelms the small L1I cache, creating a significant performance bottleneck. Unlike data stalls, these misses cannot usually be hidden through out-of-order execution, and prior work [6] indicates that adding an intermediate cache level (e.g., a 256KB L2) is not likely to be effective, as the instruction footprint is so large (over 2MB). Barnes and *moldyn* gain only modest benefits from parallel access, as their runtime is dominated by computation (busy time). Ocean is more sensitive to LLC time. Interestingly, *ocean* experiences a slowdown under MRU way prediction. MRU prediction is little better than random guessing for *ocean* (see next sub-section) and the additional data array bandwidth pressure created by mispredictions (each misprediction incurs two data array accesses) leads to significant queueing delays in bandwidth-intensive execution phases. Although *canneal* exhibits a large fraction of LLC stalls, these misses are largely coherence misses (i.e., accesses to dirty data), which spend their time traversing the on-chip network rather than in an LLC tile.

C. Sensitivity to Partial Tag Width

In Figure 6, we examine the sensitivity of prediction accuracy to the width of the partial tag comparison and explore the impact of the inhibit bit to reduce partial tag collisions. The prediction is successful if it activates the correct way during a cache hit, either because the partial tag match identifies a unique way (Predicted-Unique), or the inhibit bit correctly discerns amongst colliding partial tags (Predicted-Collision). When the cache access will miss, the predictor ideally should not activate an erroneous way (NoPredict-Miss). On the other hand, mispredictions occur when collisions obscure which way to predict during a hit (Mispredict-Collision) or when any way is predicted during a miss (OverPredict-Miss).

Naturally, as we increase partial tag width from 0 to 8 bits, the Predicted-Unique and NoPredict-Miss fractions increase steadily towards perfect accuracy. The relative size of the

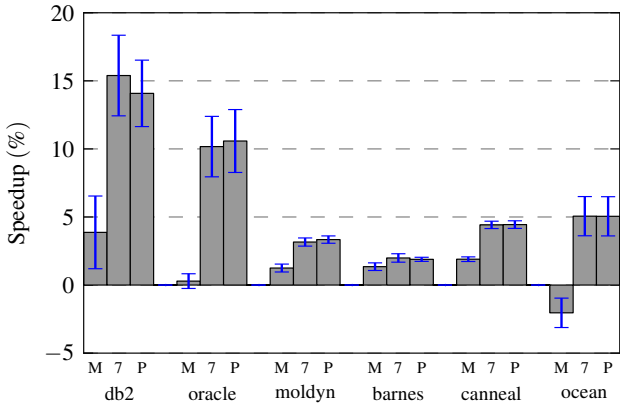

Fig. 4. Percent speedup over sequential access.

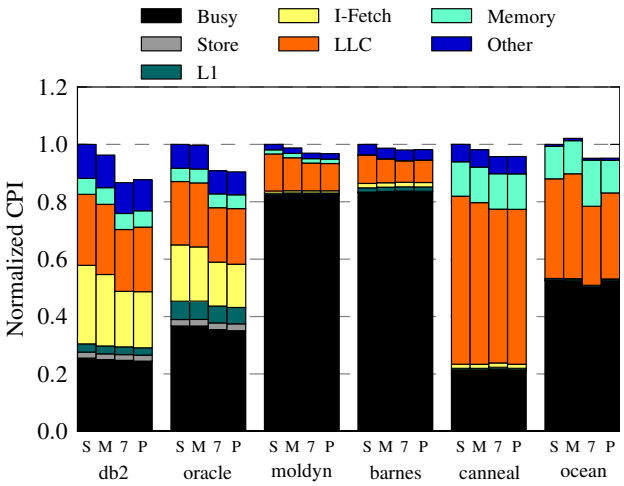

Fig. 5. Impact on Cycles Per Instruction.

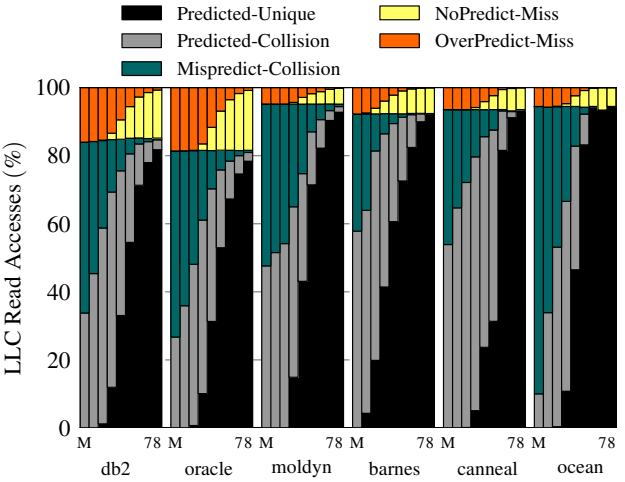

Fig. 6. Impact of partial tag matching.

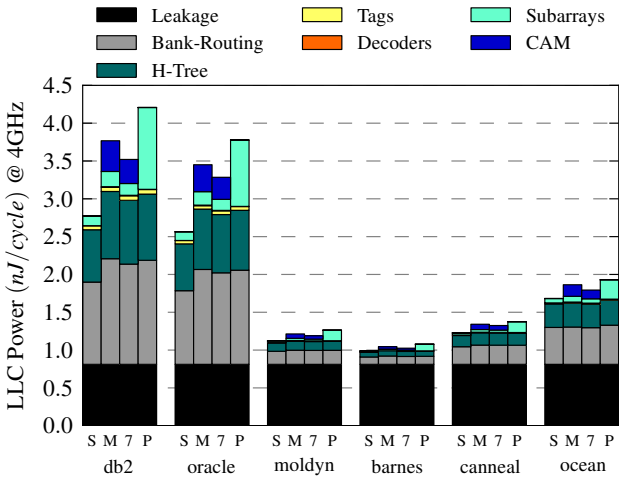

Fig. 7. LLC power.

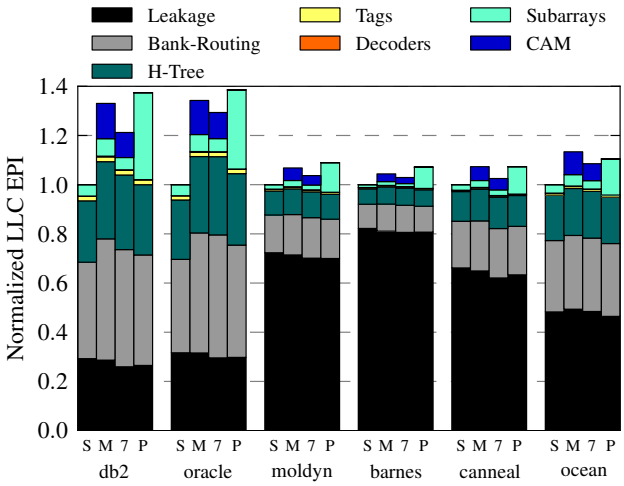

Fig. 8. Normalized LLC energy per instruction.

Predicted-Collision segment indicates the importance of the inhibit bit. Broadly, the results indicate that inhibit bits are critical for accurate prediction when the partial tag width is 4 or less, while their impact shrinks for wider partial tags. The inhibit bit plays no role during misses.

From these results, it is clear that server applications require 6-8 partial tag bits to maximize prediction accuracy, in contrast to the 3-4 partial tag bits recommended in earlier studies [15], [5], [22]. We also see that MRU-based prediction (M)—equivalent to a partial tag width of zero (0)—never achieves prediction accuracy over 65% and can never inhibit a data array access during a cache miss.

D. Power and Energy

We turn finally to examine the power and energy impacts of embedded way prediction. Figure 7 shows absolute LLC power, while Figure 8 shows normalized LLC energy per instruction, which is equivalent to an energy-delay product. Although embedded way prediction increases power by 13% on average over sequential access, we find that its performance benefits compensate for its power costs through savings in leakage energy, resulting in a net EPI increase of 11%. This energy efficiency improvement arises because the cache leakage power is amortized over more instructions per unit time. Comparatively, parallel access incurs an almost doubled power increase of 23.4%, resulting in an EPI increase of 17.5%. The dynamic power overhead of parallel access is higher than the other designs, because it activates all 16-ways each access, whereas our design activates at most two. While a full-system power analysis is beyond the scope of this paper, the EPI metric is expected to improve for high performing designs.

VII. Conclusion

Server applications are growing increasingly sensitive to on-chip cache access latency, as larger capacities allow larger working sets to be captured on chip. In this paper, we have revisited way-prediction using partial tag matching as a means to accelerate accesses in large LLCs without abandoning the energy efficiency advantages of sequential cache access. The central challenge of deploying way prediction in highly-associative LLCs is to enable the wider partial tag comparison called for in the server context while still overlapping the partial tag comparison with wordline decode. To this end, we have proposed embedded way prediction, an architecture and circuit design that embeds a dynamic CAM circuit within data sub-array decoders. We demonstrate that embedded way prediction achieves all the potential performance of parallel lookup, improving performance by up to 16% (7% on average) while incurring only a 8.2% average increase in LLC energy per instruction.

References

[1] B. Amrutur and M. Horowitz. Fast low-power decoders for rams. *IEEE J. Solid State Circuits*, 2001.

[2] S. Bell et al. Tile64-processor: A 64-core SoC with mesh interconnect. In *IEEE Intnl. Solid-State Circuits Conf.*, 2008.

[3] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[4] B. Calder, D. Grunwald, and J. Emer. Predictive sequential associative cache. In *Proc. Intnl. Symp. on High-Performance Computer Architecture*, 1996.

[5] J. Chen, R. Peng, and Y. Fu. Low power set-associative cache with single-cycle partial tag comparison. In *6th Intnl. Conf. on ASIC*, 2005.

[6] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proc. 17th Intnl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2012.

[7] M. Ferdman, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Temporal instruction fetch streaming. In *Proc. 41st Ann. Intnl. Symp. on Microarchitecture*, 2008.

[8] 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*, 2009.

[9] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *Proc. of the Intnl. Symp. on Low Power Electronics and Design*, 1999.

[10] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer. High performance cache replacement using re-reference interval prediction. In *Proc. 37th Ann. Intnl. Symp. on Computer Architecture*, 2010.

[11] R. E. Kessler, R. Jooss, A. Lebeck, and M. D. Hill. Inexpensive implementations of set-associativity. In *Proc. 16th Ann. Intnl. Symp. on Computer Architecture*, 1989.

[12] N. Kim, K. Flautner, D. Blaauw, and T. Mudge. Drowsy instruction caches. leakage power reduction using dynamic voltage scaling and cache sub-bank prediction. In *Proc 35th Ann. Intnl. Symp on Microarchitecture*, 2002.

[13] H. Kodata, J. Miyake, Y. Nishimichi, H. Kudo, and K. Kagawa. An 8kb content-addressable and reentrant memory. In *Intnl. Solid-State Circuits Conf.*, 1985.

[14] L. Liu. Cache designs with partial address matching. In *Proc. Ann. Intnl. Symp. on Microarchitecture*, 1994.

[15] R. Min, Z. Xu, Y. Hu, and W.-b. Jone. Partial tag comparison: A new technology for power-efficient set-associative cache designs. In *Proc. 17th Intnl. Conf. on VLSI Design*, 2004.

[16] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: A Tool to Model Large Caches. Technical Report HPL-2009-85, HP Labs, April 2009.

[17] H. Park, S. Yoo, and S. Lee. A novel tag access scheme for low power l2 cache. In *Proc. Design, Automation Test in Europe Conf.*, 2011.

[18] M. Powell, A. Agrawal, T. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proc, 34th Ann. Intnl. Symp. on Microarchitecture*, 2001.

[19] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: Statistical sampling of computer system simulation. *IEEE Micro*, 26:18–31, 2006.

[20] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. In *Proceedings of the 22nd annual international symposium on Computer architecture*, pages 24–36, 1995.

[21] S.-H. Yang and B. Falsafi. Near-optimal precharging in high-performance nanoscale cmos caches. In *Proc. 36th Annual Intnl. Symp. on Microarchitecture*, 2003.

[22] C. Zhang, F. Vahid, J. Yang, and W. Najjar. A way-halting cache for low-energy high-performance systems. *ACM Trans. on Architecture and Code Optimization*, 2(1), 2005.