D-SOAP: Dynamic Spatial Orientation Affinity Prediction for Caching in Multi-Orientation Memory Systems

Minli Julie Liao
Department of Computer Science and Engineering
School of Electrical Engineering and Computer Science
Pennsylvania State University
University Park, Pennsylvania, USA
Email: mjl5868@psu.edu

Jack Sampson
Department of Computer Science and Engineering
School of Electrical Engineering and Computer Science
Pennsylvania State University
University Park, Pennsylvania, USA
Email: jms1257@psu.edu

***Abstract*—Previous works have shown the possibility of constructing row-column and multi-stride memory systems that can exploit simultaneously dense access along multiple logical data orientations to offer more than 3x speedups on some workloads. However, existing multi-orientation memory (MOM) and MOM-caching approaches presume that the orientation preference of a memory request is statically determinable and rely on both ISA and compiler changes to express and extract these preferences for performance gains. Thus, current MOM-caching approaches cannot readily provide benefits in the presence of dynamism with respect to data layout, data-dependent code behavior, or access ordering. Accurate orientation prediction will allow MOMs to benefit a larger range of workloads.**

In this paper, we describe the sources of orientation preference dynamism and show that the sensitivity of orientation prediction to cache line utilization, as well as to access pattern, differentiates it from stride prediction. We introduce a hardware-managed utilization-focused orientation predictor, D-SOAP, and compare it with a set variants (D-SOAP-*) that make use of utilization, local stride analysis, and prefetcher feedback as sources of information, both in isolation and in combination, to predict orientation preference and evaluate the impact of each information source. We evaluate the D-SOAP mechanisms on workloads with both dynamic, data-value-dependent (DVD) and statically identifiable data-value-independent (DVI) orientation preferences. We demonstrate that the D-SOAP variants using utilization information 1) track the performance of the preferred orientation within 2%, on average, and 18%, at worst, across microbenchmarks sweeping data distributions for DVD patterns, avoiding the up to 267% slowdown seen with misaligned static preferences; 2) provide competitive (4% speedup, on average) performance, compared to prior static annotation approaches relying on a priori data profiling, in DVD scenarios; 3) closely track static annotations for DVI scenarios that lack any exploitable dynamism (1.8% speedup, on average).

***Index Terms*—cache, symmetric memories, orientation prediction**

I. Introduction

Many common data structures support meaningful traversals along multiple logical dimensions, with correspondingly preferred (logically-oriented) spatial locality. For instance, in an in-memory database (IMDB) an update to a table of records will likely exhibit dominant locality within the record being updated, whereas a function applied to a particular record field across multiple records in that same table (e.g. an analytics operation) more likely exhibits field-column locality. Similar reasoning applies even to irregular or dynamic access patterns, such as multiple structures using a common indirection array for indexing or a mix of row and column data being accessed as the result of a filtering condition in a selection query. Despite this, for implementation viability and ISA simplicity, the dominant model for commercially available memories has remained optimized for providing dense access to only a single, statically selected dimension into which all multi-dimensional data must be projected. This limits memory layout optimizations [1]–[6] in the presence of any dynamic behavior and demands code transformations, such as tiling [7]–[9], to emulate higher-dimensional locality.

However, several recent efforts [10]–[12] have explored exposing models with explicitly multi-orientation memory (**MOM**). Several emerging memory technologies [13]–[18] can implement structures with nearly symmetric access costs along multiple physical orientations (e.g. row and column within a crosspoint array [19]) and memories explicitly built to exploit this symmetric access have been proposed [20]. Previous works [11], [12] have shown the potential benefits of coupling these physical row and column accesses to logical row and column request interfaces in both main memories and caches. Even in the absence of a physically MOM technology, proposed augmentations of on-chip memory structures to construct and retain MOM patterns within the cache have likewise demonstrated large potential benefits (>3x speedup) for strided access patterns [10].

A common feature of prior works is that the orientation preference of a particular memory operation is presumed to be **static**. While this may be a reasonable assumption in, for instance, BLAS [21] kernels, it substantially limits the scope of MOM optimizations in more complicated access patterns, such as queries on an IMDB. Oriented caching of memory accesses

This work was supported in part by NSF grant 1500848

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00055

in the presence of structures with dynamically allocated members, indirect (e.g. A[x[i]]) access, control-dependent access, and sparse access patterns to multi-dimensional structures are similarly challenging in that *it is not always clear, at the time of code generation, whether caching the row or the column containing the requested (scalar) access will exhibit better spatial locality during that line's cache-resident period.* Moreover, such fill orientation decisions are not independent; the best orientation option for a given fill depends on both previous and future orientation decisions as there are often many ways to cover the same set of references with different sets of oriented cachelines of similar cardinality that may in turn lead to different sequences of evictions, cache misses, etc.

In this paper, we develop mechanisms for orientation prediction, and evaluate them on the restricted set of MOMs that support row and column accesses as their two possible orientations. We describe sources of orientation dynamism that limit the applicability of static orientation assignments in a number of common scenarios. We then discuss the intuitions driving a utilization-first approach to orientation prediction, how that subtly distinguishes orientation prediction from the goals of stride predicting prefetchers, and introduce a hardware-managed Dynamic Spatial Orientation Affinity Prediction (*D-SOAP*) scheme using potential utilization information gathered during the miss period for each fill (*D-SOAP-U*) that elides the need for ISA changes, detailed compiler analysis, or a priori knowledge of data value distributions in order to exploit MOM/MOM-caching systems. We then extend the D-SOAP-U model to incorporate other potential sources of information, namely prefetcher and L1-stride information, in lieu of or in combination with utilization information (*D-SOAP-**) in order to understand the relative benefits of each information source in guiding cache-fill orientation preference prediction. We demonstrate, with microbenchmarks, that, in dynamic orientation affinity scenarios, such as IMDB queries dependent on the value distribution of a key field, the D-SOAP mechanisms leveraging utilization information can closely track the optimal selection of orientation affinities across any value distributions, whereas static row/column annotations can impose substantial overheads if the distribution is misaligned from expectations. Moreover, we show that stride analysis or prefetching alone, without utilization analysis, are not as effective for performing orientation prediction. For workloads with readily-analyzable static orientation preferences, we show that employing a D-SOAP approach closely matches static annotation ($<$ 2% average performance gains).

The key contributions of this paper include:

- To the best of our knowledge, this is the first work to explore hardware approaches for dynamically predicting spatial orientation affinity for MOMs.
- We show the primacy of utilization-based approaches for driving dynamic orientation prediction (D-SOAP) mechanisms by exploring the relative benefits of several information sources (utilization, stride, and lower-level cache prefetch information) and their ensembles. Utilization-based approaches are consistently the most effective, although they do benefit from ensembling to compensate for short histories.
- We show that the D-SOAP mechanisms using utilization information (D-SOAP-U*) can gracefully adapt to program dynamism in orientation affinity. They perform within 2% of optimal, on average, and within 18%, at worst, in microbenchmarks that sweep value distributions in an IMDB query and index densities for indirect array accesses. This contrasts with static orientation assignment, which can suffer substantial performance degradation (up to 267%) if the value-based distribution assumption is incorrect.
- Over the 13 IMDB queries annotated by Wang, et al. [11] for a row-column MOM, our best dynamic prediction mechanism outperforms static annotations (that require a priori value distribution knowledge) by 4%, on average, and underperforms by 9%, at worst.
- Our best mechanisms, D-SOAP-U*, perform 1.8% better, on average, and have 0.7% overhead, at worst, of the performance of a prior static orientation assignment and exploitation approach [12] for seven MOM-annotated kernels.

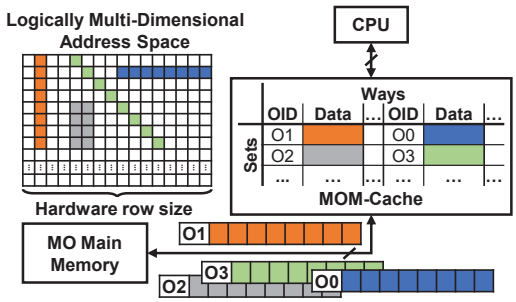


Fig. 1. Abstract view of a MOM system

II. Background and Related Work

In this section, we provide background on the abstract MOM model, fundamental hardware considerations for implementing specific classes of MOMs and MOM-caches, and the types of applications that can benefit from MOMs. We more precisely define what a "multi-orientation" memory system entails, what qualifies as an "orientation" in the scope of this paper, and place orientation prediction in MOMs in the context of related memory optimization works.

A. Multi-Orientation Memory Systems

Fig. 1 depicts an abstract view of a MOM system. A MOM either exposes a logically multi-dimensional address space, or imposes one atop the underlying linear space by treating particular patterns (e.g. specific strides) as if they were densely co-located. This paper, focuses on the latter, as the required changes are less radical in nature. The defining feature of MOMs is main memory and/or caches that are capable of containing data in and serving requests for multiple logical (dense) orientations at the same time.

Orientation Fig. 1 depicts the logical multi-orientation overlay for a linear address space. Different colored squares represent different forms of adjacent data along different dimensions. Unlike a one-dimensional address space where only data of contiguous addresses are adjacent (blue squares),

adjacent data in a MOM's address space can have non-contiguous addresses (other colored squares, usually some fixed stride apart). We describe contiguous data along different dimensions to be in different "orientations," and we use "row" orientation to refer to the special case of contiguous data with unit stride ("O0" in Fig. 1). In some MOMs [11], [12], there is a hardware-dependent stride that defines a preferred second dimension, and we refer to data along this dimension to be "column" oriented (e.g. the orange squares when the row size of this address space is hardware-dependent, "O1" in Fig. 1).

Hardware and Technology Support for MOMs Regardless of implementation, a MOM provides a memory interface that appears to provide data in different orientations with roughly similar cost. Directly using gather-scatter operations on top of traditional DRAM would allow for emulating multi-orientation access, but such a system would exhibit highly asymmetric properties among accesses to different orientations. Texture caches [22], [23] exploit tiled locality, which is multi-dimensional. However, they are generally read-only and have high latencies for all accesses. Recent work has proposed memories that provide MOM capability directly by data shuffling on DRAMs [10], or by leveraging the inherent physical symmetries of some two-terminal emerging memory technologies in crossbar configurations [11], [12] to freely allow access to data along either row or column orientations. Symmetric access topologies have also been shown for 3-terminal (FeFET) memory devices [20] and several transpose-access SRAM designs have been described [24]–[26]. While it is possible to use gather-scatter operations on top of these symmetric access MOMs to achieve orientations that cross physical access boundaries, we do not address this in the scope of this paper.

MOM Caching Recent works [10]–[12] that have evaluated architecture with MOMs have also proposed cache hierarchies that can cache oriented data. We refer to all such caches as MOM-caches, regardless of their implementation specifics (e.g. whether in-cache data is only logically or physically contiguous in multiple orientations). While physically multi-dimensional caches have been proposed to implement MOM-caches [12] regular SRAM caches using orientation metadata to virtualize multi-orientation support have been more broadly proposed [10]–[12]. These caches present a common set of challenges in addressing schemes for different orientations and potential duplicates of data in differently oriented, but intersecting, cache lines. A key differentiator among the caching schemes proposed in prior works is the specific addressing schemes and coherence protocol modifications implemented to solve these challenges.

B. MOM Applications

IMDB Applications Data in relational databases is representable as 2D tables (lists of records with multiple fields) and, in classical systems, these tables are either stored record-after-record (row-store) or field-after-field (column-store). The common processes OLTP (on-line transactional processing) and OLAP (on-line analytical processing) have different preferences, with OLTP preferring row-store to access records with high locality, and OLAP preferring column-store to access fields with high locality. Importantly, in a traditional memory system, satisfying both of these localities at once with a single data layout is not possible, whereas MOM systems allow each functionality to simultaneously access the same table in the manner most beneficial to its access locality. Fig. 2 shows a MOM data layout proposed in Wang, et al. [11], where records are stored consecutively in row orientation, and fields consecutively in column orientation.

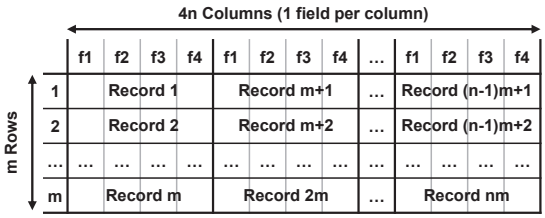


Fig. 2. MOM data layout of an IMDB table

Linear Algebra MOM-aligning matrix data layout (one dimension for each orientation) achieves cheap accesses among multiple dimensions. One simple example is 2D matrix multiplication, where a matrix is accessed in rows and another in columns. Without a MOM, the matrix accessed in column needs multiple memory accesses (cache lines with low utilization) to get consecutive data in a column, and tiling or other blocking optimizations must be employed to preserve access locality within limited resources. In comparison, matrices stored in a MOM offer dense access to both rows and columns thereby limiting complications in algorithm expression, tuning or preprocessing (e.g. transpose).

Other Applications As has been previously noted [10], a broad range of applications, including key-value stores, graph processing, and graphics, can potentially derive benefits from MOMs and MOM-caching by exploiting simultaneously dense accesses to both multiple fields within a memory object and the same field across many memory objects beyond the obvious case of IMDBs. However, no widely used compiler infrastructures or codebases target MOM models or provide memory allocators designed to align with the logical-physical mapping requirements of proposed row-column MOMs or otherwise pack data in multiple simultaneous access localities. Thus, while we believe that there is broad applicability for MOMs, we limit our experimental investigations in this work to applications that can be manually re-optimized for MOMs.

C. Other Related Work

Symmetric and Transpose Memories In addition to two-terminal technologies such as STTRAM [13], [14], PCM [15], [16] and ReRAM [17] that, in crosspoint topologies, can perform read and write operations via either of the vertical or horizontal wire sets [19], symmetric access has also been explored in other technologies and array structures, each with their own tradeoffs in density, power, and timing. For example, FeFET [27] memories have been proposed with symmetric access capabilities [28], as have multi-layer SRAMs built with monolithic 3D integration [24], and symmetric access

DRAMs [29] and SRAMs [25], [26], [30]. While we acknowledge that many of the technologies that are the strongest candidates for enabling practical MOMs are not yet fully mature and have their own technology-specific limitations (e.g. wear-leveling in NVMs, ECC overheads in robust implementations of symmetric memories, etc.) the performance potential offered by MOMs and MOM-caches for select applications still warrants deeper exploration of whether these benefits can be achieved at broader scope. There is a space of tradeoffs in potential ECC solutions ranging from word-oriented ECC (where the ECC codes are either interleaved or arranged in a physically parallel, multi-oriented array) to creating line-level ECC codes in both orientations (which offers the potential for greater robustness through overlapping ECC, but would require multiple ECC updates per memory write). An investigation of the best ECC approaches for multi-oriented main memories, while a worthwhile topic, is orthogonal to this work.

Access Pattern Locality Many prior works aim to improve performance by increasing data locality. Various compiler optimizations, such as layout optimizations [1]–[5], loop reordering and tiling [7]–[9], [31], etc. improve data locality by co-aligning data layout and access patterns in a cache-favorable fashion. Recent works have provided hardware support for more diverse data locality by allowing caching of non-unit-stride data [10]–[12], and have demonstrated considerable performance benefits when combined with proper compiler optimization.

Unlike these previous approaches, the D-SOAP schemes developed in this work will discover orientation preferences dynamically, allowing for adaptation to run-time determined layouts or access sequences. Although this work limits its scope of evaluation to 2D MOMs with unit-stride (row) and 1 fixed-stride (column), D-SOAP schemes could be extended with a multi-bit stride size field to accommodate multiple stride patterns. Moreover, for irregular, non-stride patterns, demand-driven analysis can still provide insight for spatial locality affinity among temporally close accesses.

Cache Utilization The bandwidth and allocation utilization efficiency benefits in MOM-caching have similarities with footprint [32] and compression techniques [33]–[36] for caches and memories. However, the mechanisms are largely orthogonal, and both MOM support and orientation prediction could provide new opportunities for existing approaches to concisely express which data is useful to cache. Likewise, techniques such as sectored caches [37] are complementary to supporting oriented cache lines, and could further reduce bandwidth and boost cache utilization for lines that exhibit exploitable sparsity in both dimensions.

Hardware Prediction Mechanisms There is a vast literature in hardware prediction for diverse use cases. For example, branch predictors (e.g. [38]–[41]) for control flow prediction, value predictors (e.g. [42]–[44]), critical path predictors (e.g. [45]), prefetchers (e.g. [46]–[53]), and access granularity predictors (e.g. [54], [55]) for multi-granularity memory accesses. D-SOAP resembles prefetching in the sense that it aims to minimize misses on subsequent accesses that share a common address pattern. However, D-SOAP's primary focus is on predicting which fill request orientation will maximize intra-line data utility, rather than prefetching's inter-line temporal access prediction. D-SOAP also shares certain goals with granularity predictors that also aim to improve cache utilization. However, orientation utilization is not a bipolar (fine or coarse grain) decision; sparsity in one orientation does not imply higher density in another.

Similar to many of these previous prediction mechanisms, all of our proposed D-SOAP mechanisms attempt to infer based on previously observed history. Different from branch or value predictors, there is no correctness issue since the same data referenced by a demand miss is present in a cache line of either orientation. However, both branch and value predictors can subsequently verify the correctness of a prediction, and even prefetchers can get some insight based on whether the prefetched cache line is used. Understanding the *optimal* decision for D-SOAP is more complicated in that there is no authoritative (nor even time-bounded) feedback on whether the predicted orientation was actually better to provide a history of "correct" decisions. With D-SOAP, due to the fact that selecting cache lines to be filled with different orientations changes the content of cache and interacts with replacement policies, there is no clear way of *efficiently* calculating the performance of counterfactual orientation sets.

III. Orientation Dynamism

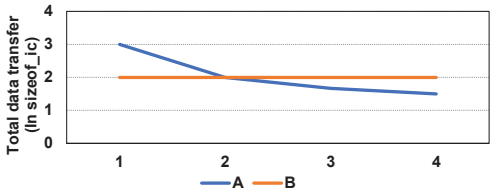
When optimizing for access locality, *logical* orientation preferences arise from underlying program and data interactions. Practical exploitation of these localities is deeply entangled with how data and orders of access (e.g. [56]) are mapped into a specific memory layout and references re-ordered during compilation or by explicit tiling and other loop transformations [1], [7], [31]. Previous works on MOM systems [10]–[12] have shown that it is possible to statically analyze the orientation preference of each load and store for some applications, and these applications receive performance improvements due to reduced memory bandwidth to MOMs when serving oriented data (higher utilization within transfer blocks) and increased hit rates in MOM-caches (higher correlation between spatial and temporal localities).

To convey orientation preference, prior works have extended the ISA with oriented load and store (or scatter-gather) instructions, and bind each load and store with an orientation. However, such approaches are limited. Firstly, not all programs are amenable to the required static analysis. Moreover, binding an orientation to a load or store instruction implicitly assumes that the preferred orientation for this instruction never changes, which is not always the case. We refer to such changes in preference over time as *orientation dynamism*.

In this section, we describe the different sources of orientation dynamism, where the preferred orientation for an access cannot be found through static analysis, which motivates our search for runtime orientation preference prediction schemes. To simplify the discussion, we assume a MOM system with

1: Aligned arrays x, y, indirection array ic of size $sizeof\_ic$
2: **for** $i \leftarrow 0, sizeof\_ic - 1$ **do**
3: $\quad x[ic[i]] \leftarrow x[ic[i]] + y[ic[i]]$

(a)



(b)

Fig. 3. Code fragment (from [57]) with indirect indexing (a), and total data transfer (b) under different density assumptions (# of elements accessed / cacheline) for *ic*

only the two orientations "row" and "column" and that each element/field in the following examples has the size of word, with each cache line holding 4 words.

A. Statically Unanalyzable Access Patterns

An example of code that is hard to analyze statically is accesses through indirection. Even if the runtime access pattern is analyzable, static analysis cannot provide a complete picture because the addresses are unknown at compile time.

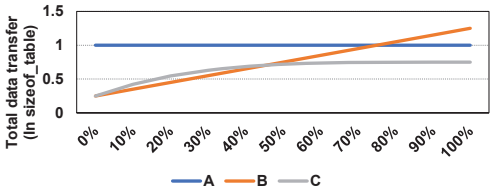
Consider the following code fragment in Fig. 3(a) (from [57]). Assume that arrays *x* and *y* are laid out along the row orientation and are aligned such that *x*[k] and *y*[k] are in the same column orientation. In a MOM system, memory access to *x* and *y* have several options regarding their orientation: A) both in row; B) both in column; C) one in row and one in column. Option A would have been preferable if the array *ic* is dense. For example, if *ic*[i]=i, then it would have been obvious that row is the preferred orientation. Option B is more preferable when *ic*[i] are far apart from each other (sparse). For example, if *ic*[i]=10i, then accessing in row would bring only 1 useful data per access (only 1 useful in 4 consecutive words), while accessing in column would bring 2 (*x*[10i] and *y*[10i]). Option C will always bring redundant data where the row cache line crosses the column, and hence is not an good option. Fig. 3(b) shows how much data need to be transferred for different options, assuming a uniform density on *ic* (how many useful words per cache line). Option B wins when *ic* is sparse and Option A wins when *ic* is dense. If the density of *ic* is not uniform, it is clear that different ranges of *i* will lead to optimal cache line coverings with different mixes of row and column orientations. While any input-dependent values of *ic* may be unknowable at compile time, the entailed preferences arising from its values, if said values are stable, would be equally stable, and therefore potentially learnable at runtime. Note that, unlike prior work on indirect prefetching [58], the goal in the described scenarios is to predict the denser orientation once an uncached element is accessed, not to predict the location of an element in advance.

B. Data-Dependent Orientation Dynamism

Even with direct accesses, codes with data-dependent accesses do not necessarily have static preferred orientation. The code fragment in Fig. 4(a) shows the memory accesses of a typical OLTP-style query. This example assumes that

1: Column aligned table $table$ with $sizeof\_table$ tuples.
2: **for** $i \leftarrow 0, sizeof\_table - 1$ **do**
3: $\quad$ **if** $table[i].f3 > x$ **then**
4: $\quad\quad$ Print $table[i].f1$, $table[i].f2$

(a)

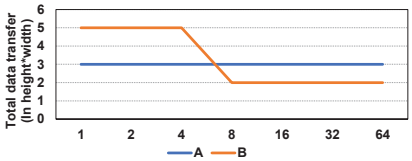


(b)

Fig. 4. Code fragment of data-dependent query "$SELECT\ f1, f2\ FROM\ table\ WHERE\ f3 > x$" (a), and total data transfer (b) over different percentages of tuples with $f3 > x$

1: Matrices X, Y and Z of equal size with $height * width$ number of elements.
2: **for** $i \leftarrow 0, height - 1$ **do**
3: $\quad$ **for** $j \leftarrow 0, (width/2 - 1)$ **do**
4: $\quad\quad Z[i][2j] \leftarrow X[i][2j]$
5: $\quad\quad Z[i][2j+1] \leftarrow Y[i][2j+1]$

(a)



(b)

Fig. 5. Code fragment with orientation-dependent eviction behavior (a), and total data transfer (b) across cache capacities (in increments of $width$ elements)

each tuple has 4 fields ($f1$, $f2$, $f3$, $f4$), each row cache line holds an entire tuple, and each column cache line holds the same field from 4 adjacent tuples (e.g. the layout depicted in Fig. 2). While OLTP-style queries generally benefit from row orientation, there are three reasonable combinations of oriented accesses for this piece of code: A) row load for $f1$, $f2$ and $f3$; B) row load for $f1$, $f2$ and column load for $f3$; C) column load for $f1$, $f2$ and $f3$. Fig. 4(b) shows the total data transfer across these different options over a range of percentages of tuples with $f3 > x$, assuming uniform distribution for probability of $f3 > x$. The best orientation combination varies: When the percentage of $f3 > x$ is below 50%, option B requires the least number of accesses, but option C outperforms B when the percentage of $f3 > x$ is higher. This shows that, even though it is possible to analyze the query statically, determining the best orientation for each memory access requires knowing the content of $f3$ in advance. Moreover, if the distribution of $f3 > x$ is not uniform, the best combination may change dynamically during query execution.

C. Other Sources of Orientation Dynamism

Even for code that is statically analyzable, with known data addresses and regular accesses patterns, without any data dependence, the preferred orientation of a memory access can still be dynamic. The code fragment in Fig. 5(a) shows simple regular accesses over 3 matrices, all matrices are aligned to the MOM system such that an access in row retrieves consecutive elements in $[i][*]$ and an access in column retrieves consecutive elements in $[*][j]$. The loop ordering favors row oriented accesses, but the sparsity in row access for matrices X and

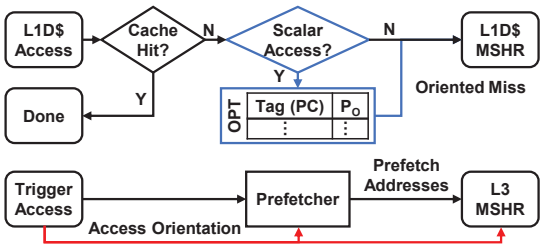


Fig. 6. Overview of the orientation prediction work flow showing demand fills (blue) and prefetches (red)

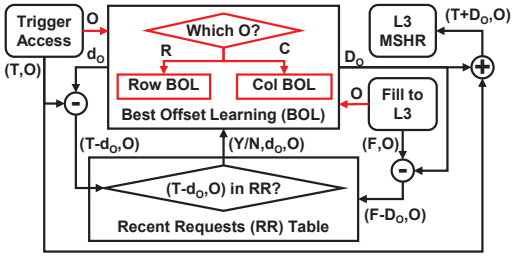


Fig. 7. MOM-aware Best Offset Prefetcher

Y favors column access for these 2 matrices. There are 2 plausible orientation combinations: A) access all 3 matrices in row; B) access matrix Z in row, and matrices X and Y in column. Fig. 5(b) shows the total size of data transfer across different cache sizes (assuming a fully associative LRU cache) with different orientation options. Even though column accesses for matrices X and Y from option B retrieve more data that would have been used by the program, resulting in better performance in large enough caches, the cache line may be evicted before all useful data has been accessed when the cache is small. This shows that preferred orientation can also be influenced by expected cache residency, and be different for different caches. Moreover, in a system with shared caches, the preferred orientation can be dynamic depending on the cache behavior of other applications.

There are several other scenarios where orientation dynamism can arise. These include access preference for iterations over dynamically allocated memory objects with 2D-allocated neighbors, inter-thread cache interference, and multi-programmed cache interference, among others. Our aim in this paper is not to exhaustively explore all sources of orientation affinity dynamism, but, rather, to demonstrate the potential expansion of codes that can benefit from MOMs given hardware-support for orientation dynamism.

D. Runtime Orientation Preference Prediction

In this paper we propose performing runtime orientation preference prediction in hardware. Although it is possible to perform orientation prediction through software methods, such as additional runtime analysis code that generates oriented accesses on the fly, it induces performance overhead and software complexity. Moreover, a hardware approach can suit a wider range of applications regardless of whether SW-runtime analysis is available. We show that simple hardware changes in the caches of MOM systems are enough to provide accurate orientation prediction for both the dynamism described in section III-A and III-B, and all latency overheads are off the critical path with little impact to performance.

IV. HW Support for Oriented Fill Prediction

In this section, we propose hardware mechanisms for runtime orientation prediction to address the sources of orientation dynamism described in Sections III-A and III-B. We identify two sources of cache fills that require orientation affinity: demand-driven fills and oriented cache fills arising from prefetch operations. Fig. 6 shows an overview of our hardware orientation prediction work flow.

From the processor side, a scalar access (e.g. nearly any non-SIMD memory instruction) doesn't need an orientation binding until a decision needs to be made about which spatially local data to bring in on a cache fill; a cache hit to a given word is valid in either orientation. This is consistent with prior MOM-cache approaches where orientation annotations are only consumed on cache misses. Hence, an orientation prediction table (OPT) is introduced to provide an orientation (P_O) on the L1 cache miss path that binds to the fill request as it traverses the memory hierarchy.

From the prefetcher side, prior work [10] has shown effective prefetching results with prefetch fills' orientation following that of the triggering access. While this is straight forward for the PC-based stride prefetcher used in prior work where generated fills are guaranteed to be for the same instruction, it is less so in the case of a PC-less prefetcher where prefetch fills can be for an instruction that has different preferred orientation than the triggering access.

Below, we first briefly discuss unique considerations for prefetching in a MOM system, before detailing orientation prediction for demand-side fills.

A. Prefetching for MOMs

In a traditional memory system, both demand fills and prefetches are row-oriented. In a MOM, prefetches will only be effective when the fill request and prefetch have the same orientation. For prefetchers that do not train prefetching streams per PC, we propose splitting the stream for each orientation for prefetching. Note that this does not necessarily mean duplicating the prefetch hardware for each orientation. By employing a virtual prefetcher per orientation, prefetches can simply follow the orientation of the triggering access.

We use the Best-Offset Prefetcher [52] (BOP) as our prefetching baseline since it provided the best overall performance among the prefetchers we tested (including stride prefetcher [51], VLDP [59], BOP and Bingo prefetcher [53]). Our approach can also be applied to other prefetchers. Fig. 7 shows the work flow of BOP with changes to enable MOM-aware prefetching highlighted in red. BOP works by keeping a list of recent requests in the recent requests (RR) table, testing a list of potential offsets in the best offset learning (BOL) unit, and finds the best offset D in this list to prefetch.

To enable MOM-aware prefetching, we only duplicate the BOL for each orientation (row and column orientations in

the example), and extend the RR with 1-bit orientation for each entry. Each fill and trigger access of orientation O only interacts with the test offset d_O and best offset D_O of that direction, and only look for entries in orientation O in the RR table. This minor change to the BOP ensures that the oriented accesses will only prefetch for cache lines in that orientation with the best offset for that orientation.

B. Demand-side fill orientation

Without a "trigger" access as a guideline, demand-side fill orientation requires more complex analysis. We observe that the most essential advantage of MOM systems is the capability to transform sparsely used cache lines into densely used lines. Hence, dynamic orientation prediction is essentially predicting which orientation is denser upon a miss. This can be broken down into 2 parts: A) predicting how densely accesses will fall in each orientation and B) choosing the preferred orientation, i.e. the denser orientation. We refer to the density of useful data in a cache line as "utilization", which is calculated based on the number of words in a cache line that are used by demand requests. The use of a bitmap, rather than a pair of access counters, is not only a complexity choice. Selecting the more populated bitmap orientation is a heuristic for minimizing cache occupancy and bandwidth. Counters could possibly better optimize for the number of hits supplied by choosing one orientation over the other, but the bitmap approach prioritizes minimizing the number of fills needed to cover the requested data.

As discussed in Section II-C, determining the *optimal* orientation to associate with each request is challenging, but there are already mechanisms in the memory hierarchy that observe partial address streams and can recognize the *possibility* of an orientation affinity. Specifically, both prefetchers and MSHRs naturally observe the access stream and keep a history of that stream in some form for at least some limited period of time. While it is possible to use prefetcher techniques to first predict the future access stream and then predict the utilization based on that, this adds an additional layer of speculation that may not be necessary at this juncture: Near-term MOMs will support only a limited number of spatial localities and, if a repeating pattern exists such that there is a stable preferred orientation, then the utilization should also be repeating. This implies that learning access stream patterns just to map them into utilization is likely redundant. Most importantly, predicting the access stream adds an additional layer of speculation that generally decreases in accuracy with the degree of look-ahead, and orientation preference is a property of a collection of accesses. Instead, we collect utilization directly based on known access stream using the same hardware for MSHR cache line matches, and predict orientation simply based on this stream of previous observed preferences. In the case where the observed history is too short to capture the preferred orientation, prefetcher techniques become relevant in extending the history with predicted accesses.

Note that unstable assignment of orientation preference results in an increased likelihood of issuing fills for intersecting cache lines of different orientations. Depending on the implementation of the MOM cache, this can cause overheads in either or both of management (e.g. eviction of intersecting lines on fill or more collisions in Bloom filters) and cache capacity (duplicate data storage). Hence, we consider mechanisms which, while fully dynamic in nature, employ substantial hysteresis in changing their mind.

TABLE I
D-SOAP SCHEMES

Information source	-U	-S	-US	-P	-USP
Utilization (U)	✓		✓		✓
L1 **S**tride (S)		✓	✓		✓
Lower level **P**refetcher (P)				✓	✓

Table I shows a summary of our D-SOAP schemes, and below, we describe them in more detail. Aside from D-SOAP based on utilization, we also include several variants which are designed primarily to provide comparisons among different options for orientation prediction. First, we introduce the D-SOAP-U scheme that uses utilization to determine orientation and provide a detailed explanation on how utilization information is collected. Then, to compare the effectiveness of utilization against raw prefetcher information, we present the D-SOAP-S schemes that use L1 stride prefetcher information. To investigate the degree to which raw prefetcher information helps with utilization collection, the D-SOAP-US scheme adds D-SOAP-S information to D-SOAP-U. We also explore D-SOAP-P that uses LLC BOP information to expose the prefetcher information from lower level caches. Finally, we examine a scheme that combines all utilization and prefetcher information (D-SOAP-USP). As has been previously noted [60], the fine-grained behaviors, often at the scalar level, that are needed to differentiate between the marginal benefits of two orientation selections may have limited visibility at the outer caches and memory. We therefore expect near-processor data to provide better predictions, on average, but explore multiple data sources and their combination for completeness.

C. D-SOAP

To support our prediction mechanisms, we introduce the orientation prediction table (OPT) to record orientation predictions and associated metadata. The OPT is a small, set-associative cache indexed by a PC-hash to associate orientation predictions with specific instructions. A small counter for each entry guards against unstable predictions (omitted in figures due to space constraint). The OPT is accessed on the L1 miss path to provide an orientation for each miss before arriving at the MSHR, since all inter-cache-level data transfers in a MOM utilize oriented cache lines. Misses with no OPT entry are assigned row orientations. Write upgrade misses (e.g. in MESI, MOESI, etc.) are, unlike other L1 misses, already oriented based on the cache line that is present with read-only permission. Write permission misses are always treated as following their current in-cache orientation, since this introduces substantially less complexity than invalidating the existing line and then bringing in a new cache line in a new orientation. Although, this paper only considers data fills,

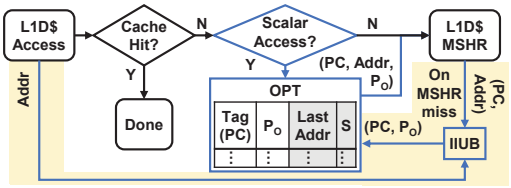


Fig. 8. D-SOAP-S and -U flow example: with unique structures and flows for -S (gray) and -U (yellow)

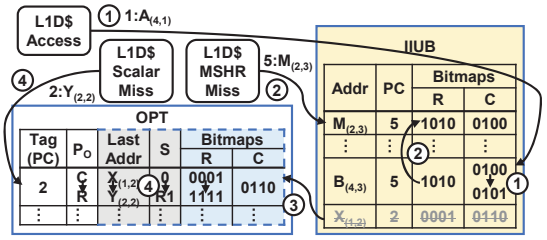


Fig. 9. D-SOAP-U/-US flow example: with common -U / -US structures (yellow), -S / -US common fields (gray), and -US unique fields (blue)

the same approach could be applied to instructions; we treat instruction caches as having uniform row-affinity.

1) Utilization-based Prediction (D-SOAP-U): We propose constructing a predictor, D-SOAP-U, that observes the recent history of scalar accesses at L1, collect utilization of each orientation for each fill, and select the best orientation (e.g. highest utilization) within the observed window as a guideline for next fill from the same instruction.

Fig. 8 illustrates the D-SOAP-U work flow, with the additional prediction flow and structure highlighted in yellow. We introduce an "inter-instruction utilization" buffer (IIUB) that records, for each MSHR miss, the relative utilization, with respect to subsequent accesses, of having served the line in one orientation or another. When an entry is removed from the IIUB, the predicted orientation is stored back to the OPT.

To track the relationship between a miss and subsequent accesses, we introduce utilization bitmaps in the IIUB. Each bit represents an element in a cache line, and each bitmap represents an oriented cache line that can fulfill the miss. Thus, for a cache with 2 access orientations and 8-word cache lines (e.g. 8x8B=64B), the overhead is 16 bits per entry (with the size of an element being a word). The IIUB is a fully associative structure where entries are inserted with the access address and PC on MSHR miss, and victims are selected based on FIFO. Upon entry removal, we perform a popcount on the bitmaps, and the orientation prediction is made following the more populated bitmap's orientation.

Fig. 9 shows an example of IIUB data collection in more detail. The IIUB entries are shown with the accessed address (Addr), instruction PC (PC), and bitmaps for two orientations, row (R) and column (C). The bitmaps are initialized with zeros, and bits representing elements that have been accessed are set to 1, including the access creating an entry. We denote a PC-address pair with its position in its row and column oriented cache lines in subscript. For example, $1 : A_{(4,1)}$ denotes PC 1, address A where A is the 4th element of its row cache line and 1st of its column line. First (see: ① in Fig. 9),

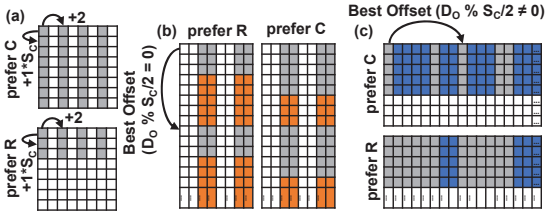


Fig. 10. Potential pathologies arising from (a) stride and (b)(c) biased information in lower-level caches

an L1 access occurs for $1 : A_{(4,1)}$. The IIUB is searched for entries that fall in the same cache line in either orientation, and finds such an entry, $5 : B_{(4,3)}$. In consequence, the 1st bit (from right to left) of the column bitmap for this entry is set to 1. Then (②), an MSHR miss for $5 : M_{(2,3)}$ occurs, and a corresponding entry is created in the IIUB. If an existing entry with the same PC as the new entry records on the same cache line as the new entry, then the bitmap record is forwarded. In this case, existing entry $5 : B_{(4,3)}$ and $5 : M_{(2,3)}$ fall in the same row cache line and have the same PC, hence the row bitmap is forwarded.

Since most current programs are optimized for row accesses, the default prediction orientation is row. The OPT incorporates a 4-bit counter to track indecisive predictions (no utilization outside of serving the initial missed element in any orientation, omitted in figures). If the counter saturates, the entire record is cleared and set to row. If a non-indecisive utilization is recorded, the counter resets to zero.

Although the default prediction is row, it is still important to keep entries that were predicted row in the OPT. This is because OPT entries also provide hysteresis to avoid unhelpful data duplication and associated overheads when there are small changes in the analyzed orientation (e.g. loop edges, or an access reordering that increases temporal distance). While implementation efficiency could presumably be further improved by filtering PCs that **never** exhibit non-row behavior (metapredicting), it is not implemented in this paper.

2) Local Stride-based Prediction (D-SOAP-S): When accesses are a regular stride apart, such sets will have a single dominant orientation, and stride analysis or prefetching feedback should readily detect them. While a dominant stride in one orientation suggests accesses fall in that orientation, it does not exclude the possibility that another orientation remains denser. For example, consider the the access pattern as illustrated in Fig. 10(a) where element 0, 2, 4, 6 in the same row are accessed in that order, and then the next row, etc. The dominant stride for this pattern is +2 in row orientation, however, column orientation is denser. Even if we also identify that the stride in column is +1, this does not mean that the orientation with the smallest stride wins: if only 3 rows are accessed, then row utilization (4) wins against column utilization (3).

To compare the performance of utilization against prefetch analysis information, we construct D-SOAP-S, which collects stride information similar to that from a stride prefetcher [51] to generate orientation predictions. Stride analysis is chosen

because 1) its feasibility at L1 makes all L1 access information available to utilization equally accessible, and 2) it is a natural fit for the evaluated MOM system with orientations where the elements in a cache line are at fixed stride apart (row at unit stride, column at fixed non-unit stride S_C).

Fig. 8 illustrates the D-SOAP-S work flow with each OPT entry extended to keep the last accessed address (Last Addr) and the last observed stride (S) between accesses. A stride is calculated on each scalar miss. A matching stride counter is also kept (omitted in figures), and the value increases when the new stride matches the old and decreases on mismatch. If the counter's value is greater than the confidence threshold, the stride is considered stable, and is used to calculate the orientation. A decision for column orientation is only taken if the stride is a multiple of S_C, with the additional constraint that stride should be smaller than column cache line stride (number of elements per cache line$*S_C$).

3) Utilization+Stride Prediction (D-SOAP-US): By extending the OPT with stride information from D-SOAP-S on top of D-SOAP-U, D-SOAP-US is created to investigate the benefit of adding longer patterns across dynamic instances of the same instruction in addition to the patterns already captured by the bitmaps. To merge stride information with utilization, we forward the bitmaps to OPT upon IIUB entry removal, and we populate the bitmaps at strided locations before performing the popcount. Stride information is used as follows: If the matching stride counter is greater than the confidence threshold, then all entries that are multiples of the stride distance away from the originating miss are marked in the corresponding bitmap as if they had been accessed. The affinity prediction is then updated based on the larger of the two adjusted popcounts.

Fig. 9 shows the D-SOAP-US process in more detail, with the OPT entry extensions shown with blue dotted borders, and additional fields of bitmaps for the row (R) and column (C) cache lines highlighted in blue. As with previous schemes, counters used for stabilizing results are omitted due to figure space constraints. Evicting an IIUB entry (see: ③ in Fig. 9) creates or updates an OPT entry with the corresponding PC. The bitmaps are copied, and, if this is a new OPT entry, e.g. $2 : X_{(1,2)}$, then its last accessed address is set to be the accessed address recorded in the IIUB entry, and its stride is set to 0. When a new miss occurs (④), e.g. $2 : Y_{(2,2)}$, the entry with the corresponding PC is accessed. The stride is calculated based on the new access $Y_{(2,2)}$ and the last address $X_{(1,2)}$. $X_{(1,2)}$ and $Y_{(2,2)}$ fall in the same row cache line at 1 element apart, hence the stride is 1 in row cache line (R1), and the row bitmap is updated accordingly. Finally, the predicted orientation is updated based on new bitmap results and used to provide orientation for the miss to $Y_{(2,2)}$.

4) Prefetch-based Prediction (D-SOAP-P): We also explore how information from prefetchers at lower levels of caches perform in predicting orientation, and expose key challenges due to lack of sub-cacheline level access information. To this end, we use BOP as an example since it's already implemented in our system. Fig. 11 shows the D-SOAP-P work flow, with

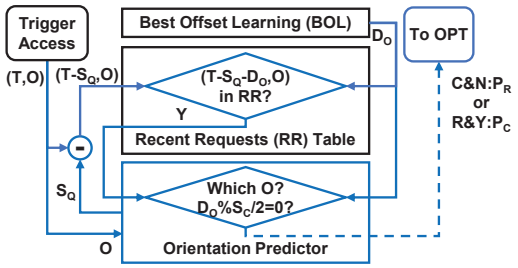


Fig. 11. D-SOAP-P work flow (demand flows only)

the prediction flow highlighted in blue, for a two orientation (row (R) and column (C)) example. The condition to predict an orientation Q different from the accessing one O is that the subsequent accesses are likely to access contiguous elements in Q. While BOP does not detect strides directly, the best offset should be a multiple of access stride for regular stride patterns, and such patterns map to specific orientation affinities.

To predict Q-affinity, consider the following: Offsets of S_Q indicate a dense stride in Q, but offsets evenly dividing S_Q indicate a Q-oriented pattern across multiple Q-oriented cachelines while multiples of S_Q indicate sparsified access in Q. The best offset D_O is tested to see if it is a multiple of $S_Q/2$, the only fraction of S_Q that does not require additional chaining through the RR to discover Q-affinity (access to T with best offset $S_Q/2$ suggest recent request $T - S_Q/2$ and likely future access $T + S_Q/2$ fall in orientation Q). In this example, D_O is only tested against column stride $S_C/2$ since row stride is 1. The prediction checks against the orientation with the largest stride first (the stride of one orientation may be a multiple of another),and defaults to row if the best offset is not a multiple of the other orientations. To check for Q-affinity, we leverage the history in the RR table, and, on each triggering access, check whether an access to $T - S_C - D_O$ exists in RR (access to T suggest recent access to $T - D_O$, which is the next contiguous address in column orientation for $T - S_C - D_O$). If a triggering row-oriented access expects contiguous access in the same column, the predicted direction is column P_C. Conversely, if a triggering access in column expects contiguous access in row, the predicted direction is row P_R. These prediction decisions P_O propagate back to the L1 along with the fill, and are inserted into the OPT.

Due to the fact that prefetchers only see cache line miss patterns and not access patterns within the cache, the information that the prefetcher observes is already biased by prior caching orientation decisions, which can make prefetch-based prediction sensitive to certain simple pathologies [60]. Fig. 10(b)(c) shows two examples of such patterns, where each square represents an element in the cache line (row ones are depicted in blue and column ones are shown in orange, assuming 8 elements per cache line), and gray squares indicate accessed elements. Different from L1 stride analysis, lower level prefetchers have no information about the utilization within a cache line. If the best offset D_O is a multiple of $S_C/2$, the prediction will become column with no way of knowing the utilization of the cache line. Conversely, the pattern in (c)

will never predict column if D_O is not a multiple of $S_C/2$.

5) Fully Ensembled Prediction (D-SOAP-USP): It is possible to simultaneously employ D-SOAP-US and -P to populate OPT entries and perform combined prediction. We keep both counters and predictions for D-SOAP-P and D-SOAP-US separately in OPT, and use the counters as the natural measurement of prediction stability. Since D-SOAP-US generally performs better than D-SOAP-P, the D-SOAP-USP prediction follows that of D-SOAP-US, unless D-SOAP-US is unstable. In that case, if D-SOAP-P is stable, then the prediction from D-SOAP-P is used over D-SOAP-US.

V. Methodology

We evaluate our proposed scheme with the gem5 [61] simulator, in syscall emulation mode, using NVMain [62] for modeling the MOM main memory as per George et al. [12], with two supported orientations: row and column. Note that, despite using an STT-RAM based MOM for evaluation, our D-SOAP schemes are not restricted to any specific MOM implementation (requiring neither STT, nor even NVM), provided that it is of the 2D row-column variety. Logical row length (S_C) is configured to 4KB, meaning that addresses at 4KB stride apart fall into the same logical column. To ensure column accesses do not cross page boundaries, 2MB pages are used. The memory controller for the MOM main memory must handle decoding for different orientations, and we have accounted for the column decoder delay by adding an extra cycle to address translation as per modeling in George, et al. [12].

The system is configured to have SRAM MOM-caches based on the 1P2L model with the "Different set" addressing scheme proposed by George, et al. [12], and employs write-allocate and write-back policies. The caches have 64B cache lines, with 8B per element. We have also added a Bloom filter per cache (128-entry for L1, 512 for each of L2 and L3) to reduce the latency overheads associated with the indexing scheme. Empirically, "Different-set mapping" with the Bloom filters added outperforms the "Same-set mapping" indexing proposed in the same work [12]. This implementation of MOM caching incurs no overhead for L1 hits in the preferred orientation, incurs an additional L1 tag check latency for L1 misses in the preferred orientation (2 cycles in our evaluation) that hit in the other orientation, and, upon completing a cache miss (off critical path), it incurs additional tag check latency to check for the existence of each overlapping cache line in the cache (empirically, the Bloom filter greatly reduces this overhead). Note that, while a hit in the non-preferred orientation incurs an extra access latency to check for its existence, it is not an overhead compared to missing in a conventional cache.

Table II shows the detailed configurations used for NVMain, gem5 and the D-SOAP mechanisms. A MOM-aware Best Offset Prefetcher [52], extended with MOM-awareness, is derived from the baseline BOP implementation in [53], and is used for L3 unless stated otherwise. Note that our MOM-aware prefetcher performs identically to a MOM-agnostic one if only row-oriented accesses exist. NVMain is configured according to the Everspin [63] devices. A 64-entry OPT (similar to L1 prefetcher size) is used where D-SOAP is enabled, with varying entry size for different schemes, and a 16-entry (smallest power of 2 larger than MSHR number) "inter-instruction utilization" buffer (IIUB) is implemented for D-SOAP-U*. The OPT for D-SOAP is only accessed on L1 misses, and is off critical path.

TABLE II
Experimental Setup

Gem5 Configurations			
CPU	X86 architecture, OoO, 3 GHz		
L1 D-/I- cache	32KB, 4 way associative 2-cycle tag lookup , 2-cycle data access Parallel tag/data access		
L2 cache	256KB, 8 way associative 11-cycle tag lookup, 11-cycle data access Parallel tag/data access		
L3 cache	8MB, 16 way associative 11-cycle tag lookup, 23-cycle data access Sequential tag/data access		
L3 prefetcher	Best Offset Prefetcher 256-entry Recent Requests table		
Main memory	4GB, NVMain simulator		
Simulation mode	Syscall Emulation		
NVMain Configurations			
Memory controller	FRFCFS-WQF (separate write queue)		
Device config	STT-RAM		
Memory size	1GB/channel x 4 channels		
Row buffer policy	Open page		
Transfer rate	1600MT/s		
D-SOAP Predictor Configurations			
Structure	Organization	Hardware Overhead	
OPT	16 sets 4 way associative 60-bit tag 1-bit orientation 4-bit counter	D-SOAP-U D-SOAP-S D-SOAP-US D-SOAP-P D-SOAP-USP	552B 1080B 1240B 520B 1280B
IIUB	16 entries, fully associative	288B	

To evaluate the D-SOAP mechanisms, we run benchmarks used in previous MOM works. We also utilize microbenchmarks with parameterized orientation dynamism to highlight the relative sensitivities of static and dynamic approaches. Note that these benchmarks/microbenchmarks are either streaming or LLC-resident, which demonstrates that ***MOMs can provide performance/bandwidth improvement that cannot be achieved simply by increasing on-chip cache***.

IMDB Query Benchmarks Query benchmarks include Q1 to Q13 of the OLXP-style queries evaluated in [11]. We created C programs matching the trace (from https://github.com/RCNVMBenchmark/RCNVMTrace) provided by Wang et al. [11] and annotated the binaries with static orientations following those provided in the trace. Table III shows the SQL statement of each query: Fields that are statically annotated for column orientation are bolded and underscored. Q3, Q5 and Q7 have $f10 > x$ mostly true, while this condition is mostly false for Q2, Q4 and Q6. The data layout of the tables also follows the layout proposed in [11], shown in Fig. 2.

Linear Algebra (LA) Benchmarks LA benchmarks include those evaluated in [12]: *sgemm*, *ssyr2k*, *ssyrk*, and *strmm* benchmarks from LAPACK BLAS [21] and *sobel*, a basic implementation of a Sobel filter. These benchmarks have been compiled with vectorization flag on, but without support of column vector accesses. Static access orientation annotations

TABLE III
BENCHMARK QUERIES

No.	SQL Statement
Q1	SELECT f3, f4 FROM table-a WHERE **f10** $>$ x
Q2	SELECT * FROM table-b WHERE **f10** $>$ x
Q3	SELECT * FROM table-b WHERE f10 $>$ x
Q4	SELECT SUM(**f9**) FROM table-a WHERE **f10** $>$ x
Q5	SELECT SUM(**f9**) FROM table-b WHERE **f10** $>$ x
Q6	SELECT AVG(**f1**) FROM table-a WHERE **f10** $>$ x
Q7	SELECT AVG(**f1**) FROM table-b WHERE **f10** $>$ x
Q8	SELECT table-a.f3, table-b.f4 FROM table-a, table-b WHERE **table-a.f1** $>$ **table-b.f1** AND **table-a.f9** = **table-b.f9**
Q9	SELECT table-a.f3, table-b.f4 FROM table-a, table-b WHERE **table-a.f9** = **table-b.f9**
Q10	SELECT f3, f4 FROM table-a WHERE **f1** $>$ x AND **f9** $<$ y
Q11	SELECT f3, f4 FROM table-a WHERE **f1** $>$ x AND **f2** $<$ y
Q12	UPDATE table-b SET f3 = x, f4 = y WHERE **f10** = z
Q13	UPDATE table-b SET f9 = x WHERE **f10** = y

TABLE IV
DYNAMIC MICRO-BENCHMARKS

ID	Pseudo-code or SQL Statement with Orientations
D1	**for** $i \leftarrow 0, sizeof\_ic - 1$ **do** $\underline{\boldsymbol{A}}[ic[i]] \leftarrow \underline{\boldsymbol{A}}[ic[i]] + \underline{\boldsymbol{B}}[ic[i]] + \underline{\boldsymbol{C}}[ic[i]] + \underline{\boldsymbol{D}}[ic[i]]$
D2a	SELECT f3, f4, f7, f8 FROM table-a WHERE **f10** $>$ x
D2b	SELECT **f3**, **f4**, **f7**, **f8** FROM table-a WHERE **f10** $>$ x

were applied according to the approach described in George, et al [12]. The size of the matrices used in these benchmarks are 512 rows by 512 columns by 64-bit elements, the same as that in [12], and all matrices have data layout that aligns with our MOM configuration.

HTAP Benchmarks We use htap1 and htap2 for hybrid analytic and transactional processing, respectively, from [10]. HTAP benchmarks use a 4096 rows by 512 columns by 64-bit element table, with the columns MOM-aligned.

Dynamic Micro-benchmarks To evaluate how D-SOAP schemes adapt to orientation dynamism, we have created two micro-benchmarks that correspond to the first two orientation dynamism cases presented in section III.

The micro-benchmark *D1*, corresponding to the dynamism presented in section III-A, is similar to the pseudo code presented in Fig. 3(a), but with 4 aligned arrays and 8 elements per cache line. D1 is parameterized on the density of the indirection array ic, varying from accessing 1 element per row cache line to accessing all 8 elements. Table IV shows this benchmark's pseudo-code and static annotations.

The micro-benchmark *D2*, corresponding to the dynamism presented in section III-B, is a query where the static analysis results for access orientation would have varied based on the result of $f10 > x$. Table IV shows the microbenchmark's SQL statement and the different static orientation annotations D2a and D2b. If the values of $f10$ were known in advance, D2a would have been chosen when $f10 > x$ is mostly false, while D2b would have been chosen otherwise. We sweep through the percentage of records with $f10 > x$ from 0% to 100%.

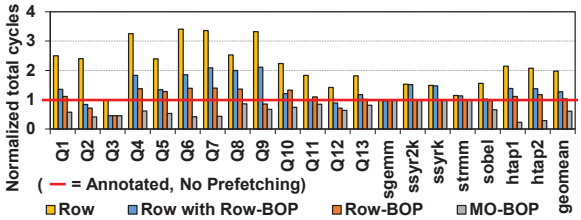


Fig. 12. Latencies of different prefetch schemes, normalized to statically annotated orientations w/o prefetch

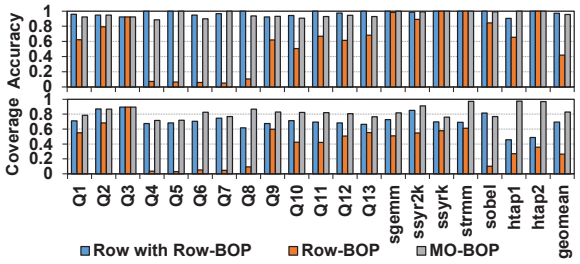


Fig. 13. Prefetch accuracy and coverage

VI. EVALUATION

We evaluate our execution scenarios and associated benchmarks with slightly different goals and success criteria. We first show the impact of MOM-awareness on prefetching. We then apply MOM-aware prefetching, and focus on dynamic benchmarks to quantitatively demonstrate the ability of dynamic prediction to offer superior performance in executions without known fixed orientation preferences. We further assess the competence of our specific prediction approaches via their ability to approach the performance of static annotation on codes with known orientation affinities.

A. Impact of MOM-awareness on Prefetch

Fig. 12 shows the impact of MOM-awareness on performance, comparing MOM-aware BOP with a MOM-agnostic BOP with a fixed row-orientation. We compare the total execution time of accessing in row only without prefetching (Row) and with row only prefetching (Row with Row-BOP); MOM access in preferred orientation without prefetching (Baseline), with a row-only prefetching (Row-BOP), and with MOM-aware prefetching (MO-BOP). All results are normalized to Baseline. The prefetch accuracy and coverage results in Fig. 13 show that MOM-awareness always matches (Q3 only has row accesses) or improves the prefetch accuracy and coverage over row-only prefetch. The data indicate that all considered codes are prefetch-sensitive except for BLAS (which are cache-resident with 8MB L3, limiting the benefit from prefetching), that enabling oriented accesses alone, on average, provides larger benefits than prefetching alone, and that prefetching and orientated accesses do synergize for higher performance. On average, oriented accesses with MOM-aware prefetching (MO-BOP) improve performance by 40% over oriented access without prefetching (Annotated) whereas MOM-agnostic prefetching (Row-BOP) degrades oriented acccess performance by 3%. Note that "Annotated" is about twice as fast as "Row",

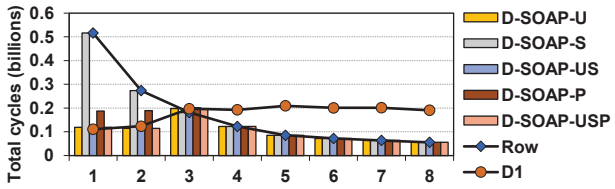


Fig. 14. Latencies for different ic access densities

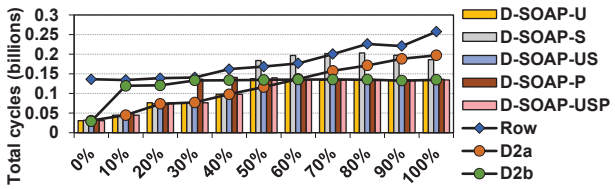


Fig. 15. Latencies for different percentages of $f10 > x$

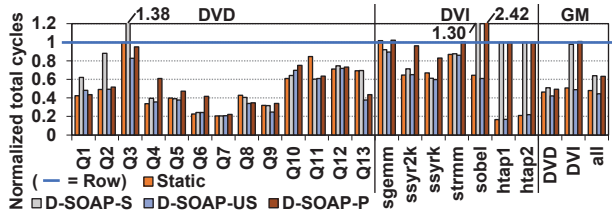


Fig. 16. Latencies for static and D-SOAP schemes, normalized to row-only

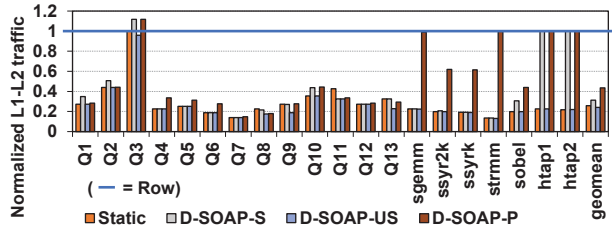


Fig. 17. L1-L2 demand traffic, row-only normalized

and "MO-BOP" is about twice as fast as "Row with Row-BOP", demonstrating the value of MOM both with and without prefetching.

B. Dynamic Orientation Micro-benchmarks

Fig. 14 shows the performance of row-only access (Row), column (annotated) access for the 4 aligned arrays (D1), and our D-SOAP schemes for the indirection microbenchmark (Table IV:D1). Row-only access performs better when the access density is greater than 3 elements in a row cache line, while D1 performs better otherwise. Note that there are substantial slowdowns when annotations do not match the preferred orientation: Row-only access performs up to over 4.6x worse than D1 when density is lower than 3 (density 1), and D1 performs up to over 3.4x worse than Row-only when density is greater than 3 (density 8). Compared to either static annotation, which sees significant overheads for certain densities, D-SOAP-U is able to track the best performing orientation under all densities with less than 1% overhead, on average. Adding stride (D-SOAP-US) or prefetcher (D-SOAP-USP) information does not change the performance. D-SOAP-P exhibits overheads due to a longer warm-up time. D-SOAP-S performs identically to the row-only access because of the lack of regular stride in the access pattern.

For the control-induced data-dependent orientation dynamism case (Table IV:D2), our D-SOAP schemes are evaluated against the two potential static annotations D2a and D2b, with a row-only baseline. Fig. 15 shows the performance of the different orientation assignment schemes. Depending on the percentage of entries with $f10 > x$ either the D2a or D2b annotation results in better performance, as expected. D2a incurs up to about 46% overhead over D2b (100% case), and D2b incurs up to 267% slowdown over D2a (10% case). D-SOAP-P tracks the best performing orientation except for the 30%, 40% and 50% points, where it suffers from the pathology depicted in Fig. 10(b). D-SOAP-U, on the other hand, always tracks the better performing orientation (about 2% overhead on average) except for the 50% case (about 18% overhead), which theoretically has the same utilization in both orientations. D-SOAP-US and D-SOAP-USP perform very similarly to D-SOAP-U with less than 0.2% difference. D-SOAP-S performs relatively well for 0% to 40% cases, but exhibits prediction instability, and significant associated overheads, starting from 50%.

C. Benchmark Performance Analysis

Having shown the benefits of prediction when the best orientation is unknown, we examine D-SOAP performance relative to static annotations describing known affinity targets, and evaluate the "accuracy" of our D-SOAP schemes based on data transfer size between L1 and L2 cache. We compare our D-SOAP schemes on the data-value-dependent (DVD) IMDB query benchmarks [11] (note that static orientation annotation for these benchmarks relies on a priori knowledge of the query criteria's satisfaction distribution at the time of query optimization), and against data-value-independent (DVI) linear algebra benchmarks and HTAP benchmarks using previously proposed static orientation annotations [12]. For performance, we note geomeans (GM) of the DVD and DVI results separately as well as together.

Fig. 16 shows execution latencies for static annotation (Static), D-SOAP-P, D-SOAP-S, and D-SOAP-US normalized to row-only access. D-SOAP-U always performs identically or worse than D-SOAP-US, showing the benefit of adding stride analysis to utilization. D-SOAP-USP performs within 0.1% of D-SOAP-US, and both D-SOAP-U and -USP are omitted for space. D-SOAP-P and D-SOAP-S have a 15-16% slowdown, over static annotation, while D-SOAP-U improve performance by 3.5%. The performance improvement of D-SOAP-U scheme comes from the 4.3% gains on the DVD benchmarks, and the 1.8% gains on the DVI codes.

While it is hard to get the optimal orientation preference for a program, it is still possible to evaluate the effectiveness of a set of orientation decisions since the better set of orientations will result in lower required data traffic. Hence, we use the data traffic as a metric to evaluate how "accurate" our D-SOAP schemes are compared to static annotation schemes.

Fig. 17 shows the amount of data traffic between L1 and L2 for different D-SOAP schemes, normalized to row-only. The orientation predictions provided by D-SOAP-US result in

similar or lower required traffic compared to static annotation. Note that this is also true for cases where D-SOAP-US has visibly lower performance (e.g. Q1 and Q10), and prefetching performance differences appear to be the key limitation.

While D-SOAP-P and D-SOAP-S can exploit MOM benefits, outperforming row-only, they are generally less accurate than static annotation. D-SOAP-P on Q3 suffers from access patterns similar to those depicted in Fig. 10(b), resulting in full column access where row would have had more utilization. The HTAP benchmarks have access patterns similar to that depicted in Fig. 10(c), resulting in D-SOAP-P's failing to produce column predictions. Q4, Q6 and Q10 suffer from limited contiguous run lengths within column-oriented data that reduce orientation confidence, and the BLAS benchmarks suffer from a similar problem after some of the loop-unrolled instructions have been predicted column and removed from the row stream. D-SOAP-S suffers from a lack of inter-instruction information that prevents it from finding the stride for the better orientation on Q3 and the HTAP benchmarks. Q2 and sobel suffer from unstable predictions and Q10 suffer from the same problems seen in D-SOAP-P.

Sensitivity Study We have also evaluated the impact of OPT and IIUB sizes, and found only small performance impacts. For OPT with 16, 32 and 64 entries, the average performance difference is less than 3% of 64-entry OPT across all D-SOAP schemes. For IIUB with 4, 8 and 16 entries, the average performance difference is less than 4% of 16-entry IIUB for D-SOAP-U, and less than 2% for D-SOAP-US. Note that the insensitivity to OPT and IIUB sizes does not indicate a lack of variation in access orientation across instructions. Rather, the insensitivity to OPT stems from evaluated benchmarks spending most of their time in tight loops with a small number of distinct PCs; the insensitivity to IIUB size shows that the benchmarks have accesses that fall in the preferred orientation that are temporally close. We also observed that, while D-SOAP-U generally performs well with 4-entry IIUB, on rare occasions, the IIUB contention results in up to 69% slowdown. D-SOAP-US, however, does not exhibit such problems, which shows the advantage of combining utilization and stride information.

Insights While D-SOAP-U was generally effective, utilization estimation is sometimes enhanced by predicting whether or not the current pattern extends. Thus, the combination of both local and larger scale stride information in D-SOAP-US / D-SOAP-USP still offers some improvements over D-SOAP-U despite both -S and -P having limited effectiveness on their own, especially for cases with limited OPT/IIUB size. The performance of D-SOAP-S and -P indicates that stride analysis is a limited proxy for orientation preference, and applications targeting MOMs without a regular stride, or with intra-cache line ordering unaligned with the stride order may not be well-served by existing feedback mechanisms.

VII. Conclusions

This paper presents an initial study of the potential benefits and core research challenges in performing orientation prediction for MOM/MOM-caching systems. It develops an intra-line utilization-based approach to accurately predict the orientation preference of cache fills, shows that utilization information is a more suitable foundation than address pattern information for orientation prediction, although the latter can complement the former, and provides a template for modifying existing prefetchers to be MOM-aware. The relative performance of D-SOAP-U versus the -S and -P variants, alongside constructed examples, such as the indirection microbenchmark, confirms the distinction between orientation prediction and stride prediction as separate, if related, efforts. Our evaluation demonstrates that the proposed D-SOAP-U* mechanisms are highly effective at predicting performance-beneficial data-value-dependent orientation preferences, avoiding the potentially enormous (267%) slowdowns of misaligned static annotations, and closely tracking (within 2%, on average) the best known static annotation for any given data distribution. This work motivates further study in both the formal measurement of optimal orientation selection for arbitrary access sequences and extends MOM benefits to a broader set of applications.

References

[1] S. Rubin, R. Bodík, and T. Chilimbi, "An efficient profile-analysis framework for data-layout optimizations," *SIGPLAN Not.*, vol. 37, no. 1, pp. 140–153, Jan. 2002. [Online]. Available: http://doi.acm.org/10.1145/565816.503287

[2] E. Z. Zhang, H. Li, and X. Shen, "A study towards optimal data layout for gpu computing," in *Proceedings of the 2012 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, ser. MSPC '12. New York, NY, USA: ACM, 2012, pp. 72–73. [Online]. Available: http://doi.acm.org/10.1145/2247684.2247699

[3] M. T. Kandemir, A. N. Choudhary, J. Ramanujam, N. Shenoy, and P. Banerjee, "Enhancing spatial locality via data layout optimizations," in *Proceedings of the 4th International Euro-Par Conference on Parallel Processing*, ser. Euro-Par '98. London, UK, UK: Springer-Verlag, 1998, pp. 422–434. [Online]. Available: http://dl.acm.org/citation.cfm?id=646663.700129

[4] D. Cho, S. Pasricha, I. Issenin, N. Dutt, Y. Paek, and S. Ko, "Compiler driven data layout optimization for regular/irregular array access patterns," in *Proceedings of the 2008 ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, ser. LCTES '08. New York, NY, USA: ACM, 2008, pp. 41–50. [Online]. Available: http://doi.acm.org/10.1145/1375657.1375664

[5] Y. Zhang, W. Ding, M. Kandemir, J. Liu, and O. Jang, "A data layout optimization framework for nuca-based multicores," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 489–500. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155677

[6] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi, "Nonlinear array layouts for hierarchical memory systems," in *Proceedings of the 13th International Conference on Supercomputing*, ser. ICS '99. New York, NY, USA: ACM, 1999, pp. 444–453. [Online]. Available: http://doi.acm.org/10.1145/305138.305231

[7] M. Wolfe, "More iteration space tiling," in *Supercomputing '89:Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, Nov 1989, pp. 655–664.

[8] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," in *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '08. New York, NY, USA: ACM, 2008, pp. 101–113. [Online]. Available: http://doi.acm.org/10.1145/1375581.1375595

[9] B. Bao and C. Ding, "Defensive loop tiling for shared cache," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Feb 2013, pp. 1–11.

[10] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Gather-scatter dram: In-dram address translation to improve the spatial locality of non-unit strided accesses," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015, pp. 267–280.

[11] P. Wang, S. Li, G. Sun, X. Wang, Y. Chen, H. Li, J. Cong, N. Xiao, and T. Zhang, "Rc-nvm: Enabling symmetric row and column memory accesses for in-memory databases," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018, pp. 518–530.

[12] S. George, M. J. Liao, H. Jiang, J. Kotra, M. Kandemir, J. Sampson, and V. Narayanan, "Mdacache:caching for multi-dimensional-access memories," in *51st IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2018, pp. 842–855.

[13] C. J. Lin, S. H. Kang, Y. J. Wang, K. Lee, X. Zhu, W. C. Chen, X. Li, W. N. Hsu, Y. C. Kao, M. T. Liu, W. C. Chen, Y. Lin, M. Nowak, N. Yu, and L. Tran, "45nm low power cmos logic compatible embedded stt mram utilizing a reverse-connection 1t/1mtj cell," in *Electron Devices Meeting (IEDM), 2009 IEEE International*. IEEE, 2009, pp. 1–4.

[14] W. Zhao, S. Chaudhuri, C. Accoto, J. Klein, C. Chappert, and P. Mazoyer, "Cross-point architecture for spin-transfer torque magnetic random access memory," *IEEE Transactions on Nanotechnology*, vol. 11, no. 5, pp. 907–917, Sep. 2012.

[15] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, 2009.

[16] D. Kau, S. Tang, I. Karpov, R. Dodge, B. Klehn, J. Kalb, J. Strand, A. Diaz, N. Leung, J. Wu, S. Lee, T. Langtry, K. wei Chang, C. Papagianni, J. Lee, J. Hirst, S. Erra, E. Flores, N. Righos, H. Castro, and G. Spadini, "A stackable cross point phase change memory," in *2009 IEEE International Electron Devices Meeting (IEDM)*, Dec 2009, pp. 1–4.

[17] A. Kawahara, R. Azuma, Y. Ikeda, K. Kawai, Y. Katoh, Y. Hayakawa, K. Tsuji, S. Yoneda, A. Himeno, K. Shimakawa, T. Takagi, T. Mikawa, and K. Aono, "An 8 mb multi-layered cross-point reram macro with 443 mb/s write throughput," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 178–185, Jan 2013.

[18] H. G. Lee, S. Baek, C. Nicopoulos, and J. Kim, "An energy- and performance-aware dram cache architecture for hybrid dram/pcm main memory systems," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, Oct 2011, pp. 381–387.

[19] A. Aziz, N. Jao, S. Datta, and S. K. Gupta, "Analysis of functional oxide based selectors for cross-point memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2222–2235, 2016.

[20] S. George, X. Li, M. J. Liao, K. Ma, S. Srinivasa, K. Mohan, A. Aziz, J. Sampson, S. K. Gupta, and V. Narayanan, "Symmetric 2-d-memory access to multidimensional data," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 6, pp. 1040–1050, 2018.

[21] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[22] Z. S. Hakura and A. Gupta, "The design and analysis of a cache architecture for texture mapping," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, ser. ISCA '97. New York, NY, USA: ACM, 1997, pp. 108–120. [Online]. Available: http://doi.acm.org/10.1145/264107.264152

[23] H. Wong, M. M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying gpu microarchitecture through microbenchmarking," in *2010 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*, March 2010, pp. 235–246.

[24] S. R. Srinivasa, X. Li, M. Chang, J. Sampson, S. K. Gupta, and V. Narayanan, "Compact 3-d-sram memory with concurrent row and column data access capability using sequential monolithic 3-d integration," *IEEE Trans. VLSI Syst.*, vol. 26, no. 4, pp. 671–683, 2018.

[25] H. Jiang, X. Peng, S. Huang, and S. Yu, "Cimat: A compute-in-memory architecture for on-chip training based on transpose sram arrays," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[26] J. Su, X. Si, Y. Chou, T. Chang, W. Huang, Y. Tu, R. Liu, P. Lu, T. Liu, J. Wang, Z. Zhang, H. Jiang, S. Huang, C. Lo, R. Liu, C. Hsieh, K. Tang, S. Sheu, S. Li, H. Lee, S. Chang, S. Yu, and M. Chang, "15.2 a 28nm 64kb inference-training two-way transpose multibit 6t sram compute-in-memory macro for ai edge chips," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 240–242.

[27] S. Salahuddin and S. Datta, "Use of negative capacitance to provide voltage amplification for low power nanoscale devices," *Nano Letters*, vol. 8, no. 2, pp. 405–410, 2008, pMID: 18052402. [Online]. Available: https://doi.org/10.1021/nl071804g

[28] S. George, K. Ma, A. Aziz, X. Li, A. Khan, S. Salahuddin, M.-F. Chang, S. Datta, J. Sampson, S. Gupta, and V. Narayanan, "Nonvolatile memory design based on ferroelectric fets," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 118.

[29] Y. Chen and Y. Liu, "Dual-addressing memory architecture for two-dimensional memory access patterns," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2013, pp. 71–76.

[30] K. Bong, S. Choi, C. Kim, S. Kang, Y. Kim, and H. Yoo, "14.6 a 0.62mw ultra-low-power convolutional-neural-network face-recognition processor and a cis integrated with always-on haar-like face detector," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2017, pp. 248–249.

[31] J. R. Allen and K. Kennedy, "Automatic loop interchange," in *Proceedings of the 1984 SIGPLAN Symposium on Compiler Construction*, ser. SIGPLAN '84. New York, NY, USA: ACM, 1984, pp. 233–246. [Online]. Available: http://doi.acm.org/10.1145/502874.502897

[32] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 404–415. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485957

[33] S. Baek, H. G. Lee, C. Nicopoulos, J. Lee, and J. Kim, "Ecm: Effective capacity maximizer for high-performance compressed caching," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013, pp. 131–142.

[34] ——, "Size-aware cache management for compressed cache architectures," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2337–2352, Aug 2015.

[35] J. Park, S. Baek, H. G. Lee, C. Nicopoulos, V. Young, J. Lee, and J. Kim, "Hope: Hot-cacheline prediction for dynamic early decompression in compressed llcs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 3, pp. 40:1–40:25, Apr. 2017. [Online]. Available: http://doi.acm.org/10.1145/2999538

[36] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12. New York, NY, USA: ACM, 2012, pp. 377–388. [Online]. Available: http://doi.acm.org/10.1145/2370816.2370870

[37] A. Seznec, "Decoupled sectored caches: conciliating low tag implementation cost," in *Proceedings of the 21st annual international symposium on Computer architecture*, 1994, pp. 384–393.

[38] D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, Jan 2001, pp. 197–206.

[39] T.-Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *Proceedings of the 24th Annual International Symposium on Microarchitecture*, ser. MICRO 24. New York, NY, USA: ACM, 1991, pp. 51–61. [Online]. Available: http://doi.acm.org/10.1145/123465.123475

[40] A. Seznec, "A new case for the tage branch predictor," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 117–127. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155635

[41] A. Seznec, J. San Miguel, and J. Albericio, "Practical multidimensional branch prediction," *IEEE Micro*, vol. 36, no. 3, pp. 10–19, May 2016.

[42] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS VII. New York, NY, USA: ACM, 1996, pp. 138–147. [Online]. Available: http://doi.acm.org/10.1145/237090.237173

[43] M. H. Lipasti and J. P. Shen, "Exceeding the dataflow limit via value

prediction," in *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO 29*, Dec 1996, pp. 226–237.
[44] A. Perais and A. Seznec, "Practical data value speculation for future high-end processors," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2014, pp. 428–439.
[45] E. Tune, , D. M. Tullsen, and B. Calder, "Dynamic prediction of critical path instructions," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, Jan 2001, pp. 185–195.
[46] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proceedings of the 17th Annual International Symposium on Computer Architecture*, ser. ISCA '90. New York, NY, USA: ACM, 1990, pp. 364–373. [Online]. Available: http://doi.acm.org/10.1145/325164.325162
[47] D. Joseph and D. Grunwald, "Prefetching using markov predictors," *IEEE Transactions on Computers*, vol. 48, no. 2, pp. 121–133, Feb 1999.
[48] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *33rd International Symposium on Computer Architecture (ISCA'06)*, 2006, pp. 252–263.
[49] K. J. Nesbit and J. E. Smith, "Data cache prefetching using a global history buffer," in *Software, IEE Proceedings-*, Feb 2004, pp. 96–96.
[50] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.
[51] J. W. C. Fu, J. H. Patel, and B. L. Janssens, "Stride directed prefetching in scalar processors," in *[1992] Proceedings the 25th Annual International Symposium on Microarchitecture MICRO 25*, Dec 1992, pp. 102–110.
[52] P. Michaud, "Best-offset hardware prefetching," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 469–480.
[53] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2019, pp. 399–411.
[54] D. H. Yoon, M. K. Jeong, M. Sullivan, and M. Erez, "The dynamic granularity memory system," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 548–560.
[55] M. Rhu, M. Sullivan, J. Leng, and M. Erez, "A locality-aware memory hierarchy for energy-efficient gpu architectures," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013, pp. 86–98.
[56] W. Kelly and W. Pugh, "A unifying framework for iteration reordering transformations," in *Proceedings 1st International Conference on Algorithms and Architectures for Parallel Processing*, vol. 1, April 1995, pp. 153–162 vol.1.
[57] S. D. Sharma, R. Ponnusamy, B. Moon, , R. Das, and J. Saltz, "Run-time and compile-time support for adaptive irregular problems," in *Supercomputing '94:Proceedings of the 1994 ACM/IEEE Conference on Supercomputing*, Nov 1994, pp. 97–106.
[58] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "Imp: Indirect memory prefetcher," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 178–190.
[59] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2015, pp. 141–152.
[60] M. Ferdman, C. Kaynak, and B. Falsafi, "Proactive instruction fetch," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 152–162. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155638
[61] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718
[62] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, July 2015.
[63] "Spin-transfer torque mram technology," https://www.everspin.com/spin-transfer-torque-mram-technology, accessed: 2017-11-11.