In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2014) [§]

BuMP: Bulk Memory Access Prediction and Streaming

Stavros Volos[†] Javier Picorel[†] Babak Falsafi[†] Boris Grot[‡]

[†]*EcoCloud, EPFL* [‡]*University of Edinburgh*

Abstract—**With the end of Dennard scaling, server power has emerged as the limiting factor in the quest for more capable datacenters. Without the benefit of supply voltage scaling, it is essential to lower the energy per operation to improve server efficiency. As the industry moves to lean-core server processors, the energy bottleneck is shifting toward main memory as a chief source of server energy consumption in modern datacenters. Maximizing the energy efficiency of today's DRAM chips and interfaces requires amortizing the costly DRAM page activations over multiple row buffer accesses.**

This work introduces Bulk Memory Access Prediction and Streaming, or BuMP. We make the observation that a significant fraction (59-79%) of all memory accesses fall into DRAM pages with high access density, meaning that the majority of their cache blocks will be accessed within a modest time frame of the first access. Accesses to high-density DRAM pages include not only memory reads in response to load instructions, but also reads stemming from store instructions as well as memory writes upon a dirty LLC eviction. The remaining accesses go to low-density pages and virtually unpredictable reference patterns (e.g., hashed key lookups). BuMP employs a low-cost predictor to identify high-density pages and triggers bulk transfer operations upon the first read or write to the page. In doing so, BuMP enforces high row buffer locality where it is profitable, thereby increasing row buffer hit ratio by 3x and reducing DRAM energy per access by 23%, and improves server throughput by 11%.

I. Introduction

Today's scale-out datacenters use thousands of servers to host the popular online applications and serve these to a global audience. In order to maintain real-time response latencies, the applications that run in these datacenters rely on large per-server DRAM pools to keep significant fractions of their vast datasets from spilling to disks [40]. Similarly, back-end processes that update the information stores also rely on massive memory capacities to minimize the time it takes for integration of new data with existing content. As a result, the majority of server power in today's datacenters is consumed by the combination of processors and memory [2, 24, 35].

With the slowdown in Dennard scaling [13], server power has become the limiting factor in datacenter expansion [11], thus requiring improvements in energy consumed per server per operation. On the processor side, the industry is transitioning to chip multiprocessors featuring a large number of lean cores [53, 54]. Such designs are well-suited for exploiting the rich request-level parallelism of server applications, while providing greater energy efficiency in the face of frequent long-latency memory stalls [13, 32]. With lean-core designs being effective at minimizing processor energy consumption, the energy-efficiency bottleneck is shifting to DRAM that must serve frequent accesses from many cores.

DRAM memory uses a page-based organization, whereby the first access to a page must *activate* (or open) the page, requiring significant energy. Once a page is open, subsequent accesses to that page are served from the *row buffer*, avoiding the high energy and latency cost of a page activation. Row buffer hits on today's DDR3 memory require 3x less energy than accesses requiring a page activation [37].

Our analysis of server applications shows that a large fraction of the DRAM accesses goes to DRAM pages with a high *access density*, meaning that once cache block *A* on the DRAM page is accessed, the majority of other blocks on that page will be accessed within the lifetime of block *A* in the last-level cache. Critically, this observation applies not only to DRAM reads triggered by load instructions, but also to reads resulting from store instructions (which fetch a block from DRAM to bring it to on-chip caches), as well as DRAM writes (triggered upon a dirty LLC eviction).

In theory, accesses to high-density DRAM pages should be served from the row buffer, thus minimizing the number of costly page activations. In practice, we find that this rarely happens as accesses to any given page are interleaved with a stream of accesses to other pages, hence destroying DRAM row buffer locality [1, 46, 47, 57]. Keeping pages open and prioritizing accesses to an open page helps only to some extent, as accesses to different blocks within a page are often separated by a distance that is beyond the reach of the memory controller's scheduling window or occur in a data-dependent fashion [51]. As row buffer locality is poorly exploited in existing designs, we find activations to be a major contributor to memory energy consumption.

One way to minimize activation energy is through modifications to DRAM chips or interfaces [1, 47, 59]. Historically, such disruptive proposals have failed to gain traction in the DRAM industry, which is focused on commoditization and adheres to rigid standards to ensure broad compatibility. A non-disruptive approach to improving DRAM energy efficiency is to increase the fraction of DRAM accesses that hit in the row buffer, thus eliminating redundant page activations. Advanced prefetchers can enforce row buffer hits by predicting spatial footprints of load-triggered memory reads within a page [4, 25, 44]; however, these proposals carry a high storage overhead (60KB per core) and ignore store-triggered memory reads and LLC writebacks. Eager writeback mechanisms [27, 28, 45] can improve row buffer locality for writebacks, but only to a limited degree as they schedule writebacks of only a few adjacent cache blocks at a time. Maximizing row buffer locality calls for a comprehensive mechanism that targets all types of memory accesses while incurring a nominal area and energy cost.

In this paper, we introduce Bulk Memory Access Prediction and Streaming, or BuMP. BuMP builds on the critical insight that server applications access memory at either coarse (few kilobytes) or fine (several bytes) granularities. The resulting bimodal memory access behavior originates from the way these applications access their datasets. Server applications commonly operate on large objects (e.g., database rows or memory-mapped files) that are accessed through a pointer-intensive indexing data structure (e.g., a hash table or a tree). For instance, a web search engine uses a hash table to quickly retrieve metadata pages relevant to query terms. While metadata pages are accessed at coarse granularity, finding them requires performing a sequence of fine-grained pointer-chasing operations through a hash bucket. As a result, DRAM access density in servers is effectively bimodal, with 59-79% (68% on average) of all DRAM accesses going to high-density pages and the rest going to low-density pages.

To accurately predict the density at which a page will be accessed, BuMP exploits two properties. First, there is high correlation between code and data. As server software utilizes a set of functions to traverse and manipulate data objects, the instruction which triggers the first read access to a page provides information as to whether a coarse-grained data object resides on the page. Second, the first dirty LLC eviction within a coarse-grained data object is a good indicator that the entire data object will be written back to memory. As such, DRAM reads and DRAM writes can be accurately predicted as going to low- or high-density pages.

Building on the properties above, BuMP employs a simple, low-cost, and effective predictor that identifies high-density pages and triggers bulk transfers upon a first read or write access to the page, thus guaranteeing high row buffer locality. The bimodal memory access behavior of server applications allows for a predictor that needs to track a small number of memory instructions (only those that are correlated with high-density pages), thereby affording a design with small cost and low complexity.

We use cycle-accurate full-system simulation of a wide range of contemporary server applications to show that:

- The majority (57-75%) of DRAM reads go to pages with high access density;
- DRAM writes account for a significant share (21-38%) of memory traffic, and the majority of those (62-86%) belong to pages with high access density;
- A simple predictor can identify high-density pages and enforce bulk transfer operations upon the first read or write to a page, with minimal on-chip power (50mW) and low storage (14KB) overhead;

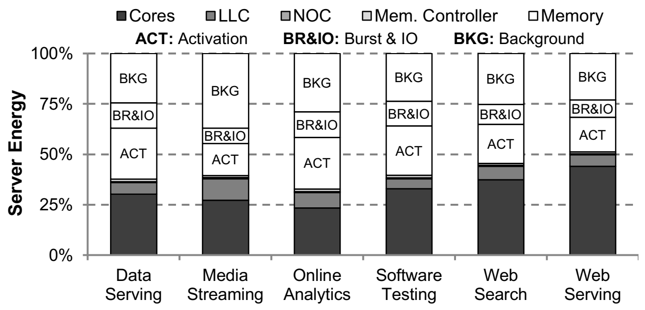


Figure 1. Energy consumption of a many-core server.

- BuMP reduces memory energy per access by 23% over a system that employs a conventional stride prefetcher with FR-FCFS open-row policy while improving server throughput by 11%. Compared to state-of-the-art spatial prefetching [44] and eager writeback [45] mechanisms, BuMP reduces memory energy per access by 20% and 13%, while improving server throughput by 3% and 9%, respectively.

II. Background and Motivation

A. Datacenter Trends

Slowdown in Dennard scaling and the emergence of memory-intensive applications arise as two inflection points, which affect datacenter design. With the semiconductor industry approaching the physical limits of voltage scaling [8, 13, 15], power becomes the main limiting factor to datacenter expansion. Servers are the major contributor to the datacenter power, accounting for up to 85% of the total power draw [24]. Therefore, system architects need to improve server energy efficiency by building servers that match the main characteristics of server applications.

Server applications have two chief characteristics which directly affect server design [9, 10]. First, these applications exhibit abundant request-level parallelism with low instruction- and memory-level parallelism within each thread. As a result, server designers are turning to many-core processors based on lean-core microarchitectures [53, 54]. Second, server applications operate on vast DRAM-resident datasets, thus requiring efficient memory systems [40].

To uncover opportunities for improving server energy efficiency, we examine chief sources of server energy consumption when running server applications [9] on a lean-core CMP with 16 cores and 16 GB of memory (Section V.A details the chip organization, methodology, and applications). We focus our analysis on processors and memory as these components dominate server power [33, 35].

Figure 1 shows the relative energy consumption of cores, network-on-chip (NOC), caches, memory controllers, and main memory. We account for both dynamic and static components of energy. Overall, memory energy is the single largest energy consumer, responsible for 48-62% of total energy. Because server applications operate on vast memory-resident datasets with poor temporal locality, (a) servers require large amounts of memory, which comes at a high static (or *Background*) DRAM energy cost (up to 37% of the total), and (b) frequent memory accesses by many cores consume significant dynamic DRAM energy (up to 38% of the total). Together, these trends indicate that the memory system is key to server energy efficiency.

B. DRAM Basics

As memory systems are central to improving server energy efficiency, we briefly review the basics of today's main memory architecture.

A DRAM chip houses multiple memory arrays organized in a set of banks. Each bank operates at the granularity of a row, also referred to as a DRAM page. Today's DDR3 memory chips employ a row size of 1024 bytes. A set of DRAM chips, called a rank, is activated together upon a memory access to achieve high bandwidth through parallelism given pin-limited DRAM chips. Multiple DRAM chips are packaged on a DIMM, and multiple DIMMs comprise a memory channel managed by a processor-side memory controller.

Memory operations comprise *DRAM reads*, triggered by processor load or store requests, and *DRAM writes*, which occur upon a dirty eviction from the last-level cache (LLC). A typical memory access consists of two operations. The first is a DRAM *page activation*, which involves copying an entire DRAM page into the so-called *row buffer*. The second is the transfer of data to/from the row buffer, consisting of two sub-operations, a *burst* and the actual I/O activity. Because a DRAM page activation operates at a granularity of a row while a transfer is per cache block, a page activation consumes 3x more energy than a transfer [37].

One way to lower the relative energy expense of activating a page is by amortizing the activation energy over multiple row buffer accesses. For instance, assuming that a processor will access 16 cache blocks within a DRAM row, up to 65% of the memory energy can be saved by fetching all cache blocks at once with a single row activation.

C. DRAM Row Buffer Locality in Servers

The challenge in maximizing row buffer hits is in discovering accesses to the open DRAM page before that page is closed and a new one is opened. Part of the challenge stems from the over-subscribed memory subsystems in many-core server processors, whereby memory requests from different cores contend for resources and destroy whatever row buffer locality may be present in another thread's request stream [46]. Another challenge is the frequent occurrence of data-dependent accesses in server workloads [9], which delays requests to a given DRAM page. Finally, massive data working sets of server workloads further compromise row buffer locality by minimizing the likelihood of temporal reuse in the row buffer across application threads.

We quantify the ability of today's servers to exploit DRAM row buffer locality in Figure 2. The baseline system integrates a conventional stride prefetcher and employs FR-FCFS open-row policy to exploit row buffer locality by

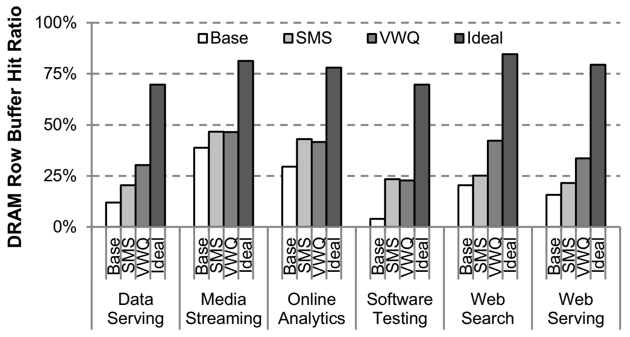


Figure 2. DRAM row buffer hit ratio of various systems.

keeping pages open and prioritizing requests to already open pages [41]. Moreover, we include a system that integrates Spatial Memory Streaming [44], a state-of-the-art spatial prefetcher, referred to as SMS. We also include a system that augments the baseline system with Virtual Write Queue [45], a state-of-the art eager writeback mechanism [27, 28], referred to as VWQ. Finally, we include an ideal system that exhibits maximal row buffer locality — i.e., it exploits all row buffer locality that exists in an access stream of a thread.

Across all workloads, the baseline system achieves a row buffer hit ratio of 21% as compared to 77% of the ideal system. As a result, DRAM page activation energy accounts for a significant fraction of dynamic memory energy (Figure 1), corroborating prior work [1, 46, 57] and demonstrating that row buffer locality is not fully exploited in server CMPs and calling for techniques to maximize row buffer hits.

SMS utilizes spatial footprint prediction [25, 44] to identify repetitive access patterns and to fetch only the predicted useful cache blocks within a page. As SMS is effective in capturing regular and irregular access patterns, row buffer hit ratio increases to 30%. However, its row buffer hit ratio is limited as it targets only memory accesses that are critical to performance (i.e., load-related traffic), ignoring store-triggered memory reads and memory writes (store misses are hidden through store buffers whereas LLC writebacks are not on the critical path of the processor). As store-triggered reads and memory writes compromise a significant share of memory activity, SMS's energy-efficiency gains are small.

VWQ generates writebacks of adjacent cache blocks upon every dirty LLC eviction, thereby exploiting writeback locality and increasing row buffer hit ratio to 36%. However, the row buffer hit ratio is still low as VWQ (a) exploits low degree of read row buffer locality (similar to the baseline system), and (b) performs lookups for a small number of cache blocks within an open row so as to minimize increase in LLC traffic.

D. Summary

Server applications require large memory capacities, resulting in high memory energy consumption. As dynamic memory energy is dominated by row activations, it is essential to increase the fraction of accesses served by the row buffer to improve average memory energy per access.

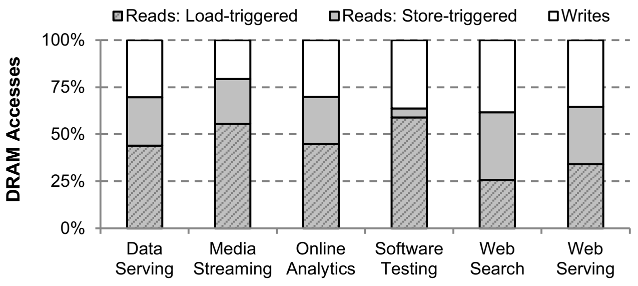


Figure 3. DRAM accesses broken down into reads (load- and store-triggered) and writes (LLC writebacks).

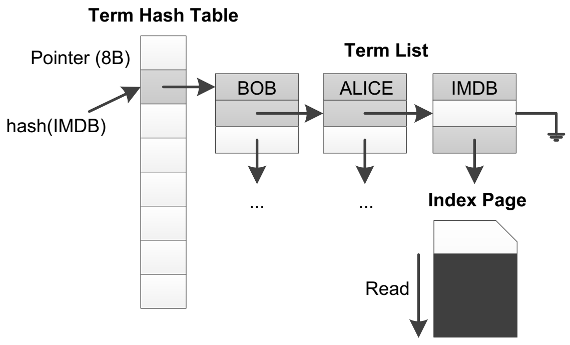


Figure 4. The inverted index in Web Search. Terms are associated with index pages and rank metadata.

III. Memory Access Characterization of Server Applications

Memory energy efficiency can be improved by exploiting row buffer locality. To uncover opportunities for increasing the row buffer hit ratio, we define *region access density* as the fraction of cache blocks in a memory region accessed between the first access to the region and the first LLC eviction of a block belonging to the region. The first eviction is a good indicator that the coarse-grained software object is dead with regard to its on-chip usefulness. Intuitively, high-density regions see most of their cache blocks accessed within a modest time window of the first access. Conversely, low-density regions have just one or a few blocks accessed.

We study both DRAM reads (triggered by load and store instructions) and DRAM writes (triggered by LLC writebacks) as both components are important in servers. Figure 3 breaks down memory traffic into reads and writes, illustrating that writes account for 21-38% of memory accesses.

A. DRAM Reads

Server applications organize and access data at either fine (several bytes) or coarse (few kilobytes) granularities. Examples of fine-grained operations are key lookups in key-value data stores and file systems, hash table walks in data stores [23], and pointer-chasing across software objects and operating system structures. Examples of coarse-grained operations include index page traversals in web search, data copying from media files into network packets in streaming servers, row (column) accesses in NoSQL (column-oriented) data stores, and object accesses in object caching systems.

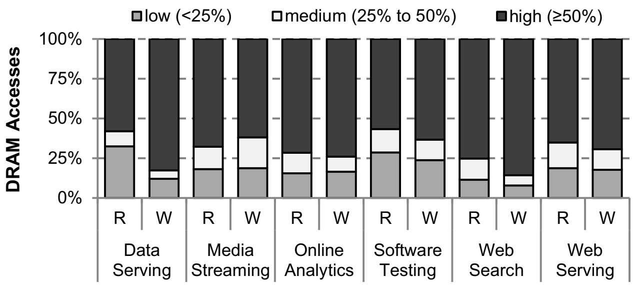


Figure 5. Region access density for a region size of 1KB. R: DRAM reads, W: DRAM writes.

An example that illustrates the phenomenon of operations on fine- and coarse-grained objects is demonstrated in Figure 4, which presents the organization of the inverted index in Web Search. The inverted index keeps query terms in a hash table and associates each term with a set of index pages. Index pages store all web pages that contain the term along with rank metadata (e.g., term frequency in the web page). Upon a search query, web pages that contain the term are ranked based on their relevance to the term. First, the term of interest (e.g., IMDB) is looked up in the hash table to find the index pages that contain the term. The hash table walk requires traversing a pointer chain over a large memory space, resulting in low DRAM page access density. Once the matching term is found, the rank metadata for all web pages containing the term is extracted from the index page and used for computing the relevance of each web page. The read of the index page leads to high DRAM page access density due to the contiguous layout of the rank metadata in application memory.

Intuitively, operations on coarse-grained software objects have high DRAM page access density. However, accesses to fine-grained software objects with good spatial locality also result in high DRAM page access density, offering opportunity for exploiting row buffer locality.

Figure 5 shows the region access density of the examined applications. We use a region of 1KB as larger regions do not provide much opportunity in amortizing the energy cost of DRAM page activations (Section V.E). Each segment represents the percentage of cache blocks touched within the region prior to the first LLC eviction of one of its cache blocks. The three segments correspond to high (≥50%), medium (25-50%), and low (<25%) access density.

We observe that memory reads to high-density regions account for a significant fraction of memory accesses, ranging from 57% to 75% (66% on average). Due to their high spatial locality, these regions can be fetched in bulk to maximize row buffer locality. Low-density regions account for most of the remaining accesses, ranging from 17% to 36% (25% on average). Accesses to these regions, such as hashed key look-ups, are difficult to predict and offer little opportunity for exploiting row buffer locality. Finally, a small fraction of accesses go to medium-density regions. Of these, a considerable fraction is attributed to coarse-grained software objects unaligned to region boundaries.

Table I. Fraction of cache blocks of a high-density region that are modified after its first LLC eviction.

Workload	Value	Workload	Value
Data Serving	8%	Software Testing	3%
Media Streaming	11%	Web Search	6%
Online Analytics	6%	Web Serving	9%

Exploiting row buffer locality on DRAM reads requires identifying accesses to high-density regions upon the first read to the region. Due to high correlation between code and data, the instruction triggering the first access to a region provides information whether the region belongs to a coarse-grained software object. For instance, software developers use a set of functions to access software objects. As such, the first call of those functions can be used to identify accesses to software objects of the same type.

Implications. The majority of DRAM reads fall into high-density regions. By identifying such regions using code correlation and by fetching them in bulk, row buffer locality can be improved. The rest of the reads offer little opportunity for exploiting row buffer locality.

B. DRAM Writes

Server applications have a high incidence of store-related DRAM traffic from high-density regions. Often, spatially clustered stores arise when coarse-grained software objects are manipulated. For instance, web servers and NoSQL data stores allocate web pages and frequently-used rows in software caches. Other examples include per-client buffers in a media streaming server to store media packets, and sockets for inter-process communication. While the store instructions themselves result in DRAM reads that allocate blocks in the cache, the subsequent writeback of modified blocks evicted from the LLC trigger DRAM writes.

As noted earlier, DRAM writes account for 21-38% of memory traffic. As Figure 5 shows, these writes share similar region density characteristics with DRAM reads. We quantify the fraction of DRAM writes going to high-density regions by measuring the number of modified blocks in a kilobyte-sized region. Across our applications, 62-86% (73% on average) of writes go to high-density regions.

Exploiting row buffer locality on DRAM writes requires eagerly writing back to DRAM all modified cache blocks of a memory region, upon the region's first dirty LLC eviction. Such optimization is effective only if the set of stores defining a memory region as high density have actually completed, meaning that their respective cache blocks have been modified by the time the first block is evicted. We quantify this phenomenon and summarize our results in Table I. On average, only 8% of the blocks of a high-density region are modified after the first dirty LLC eviction within that region. As such, the first dirty LLC eviction is a good indicator that the coarse-grained software object will not be modified in the future.

Implications. DRAM writes offer significant opportunity for exploiting row buffer locality in the context of high-density modified regions. The first dirty eviction is a good indicator that the entire region can be written back.

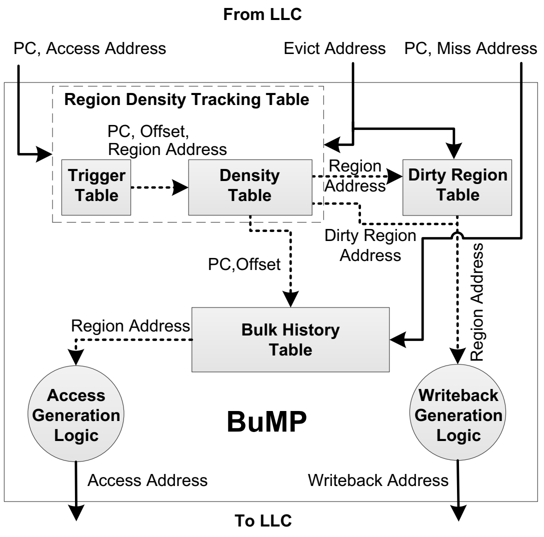


Figure 6. BuMP design overview.

IV. Bulk Memory Access Prediction and Streaming

As shown in Section III, 59-79% (68% on average) of all memory accesses go to high-density regions. As memory energy per access can be improved by exploiting row buffer locality upon accesses to high-density regions, we propose Bulk Memory Access Prediction and Streaming, or BuMP. We identify DRAM accesses (both reads and writes) to high-density regions and trigger bulk transfers upon a first access to the region, thus eliminating redundant row activations.

A. BuMP Design Overview

Figure 6 illustrates an overview of the BuMP design. BuMP employs three microarchitectural components shared by all the cores: (a) the *region density tracking table* to identify high-density regions, (b) the *bulk history table* to keep prediction metadata for high-density regions, and (c) the *dirty region table* to keep track of cache-resident high-density modified regions. As BuMP monitors LLC activity, it benefits from being physically near the LLC but is a standalone component. Therefore, BuMP is not on the critical path of the processor.

BuMP uses the region density tracking table (RDTT) to identify high-density regions. The RDTT monitors the LLC access and eviction streams to track accessed and modified cache blocks within active memory regions. A memory region is considered active in the time interval between its first access (triggering) and its first LLC eviction. Upon an eviction in an active region, the region is terminated.

For terminated regions identified as having high access density, the RDTT informs the bulk history table with prediction metadata associated with high-density regions. For high-density modified regions terminated due to a clean LLC eviction, the RDTT also informs the dirty region table.

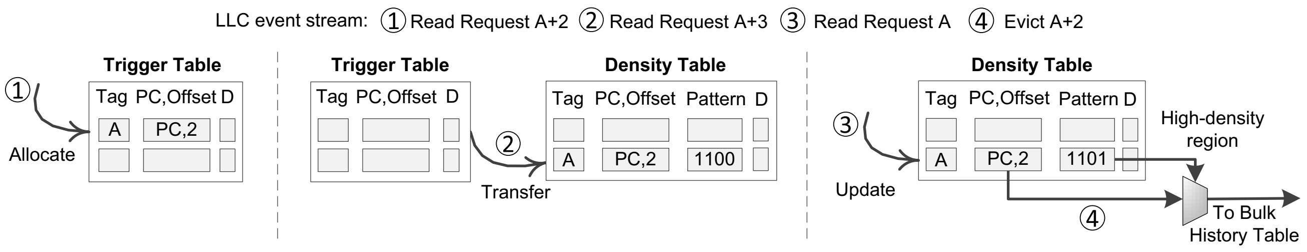


Figure 7. Example for region density tracking and identifying instructions that access high-density regions.

The bulk history table (BHT) monitors LLC misses and uses its prediction metadata to predict if an LLC miss falls into a high-density region. For an access predicted as going to a high-density region, the BHT informs the access generation logic to launch access requests for the region's blocks.

The dirty region table (DRT) monitors LLC evictions. Upon identifying a dirty LLC eviction falling into a high-density modified region, the DRT utilizes the writeback generation logic to trigger eager writeback requests for each of the cache blocks in the region.

B. Bulk DRAM Read Prediction and Streaming

BuMP employs the RDTT to track the access density of cache-resident regions. Internally, the RDTT is comprised of two tables: the *trigger* table and the *density* table. The trigger table tracks regions with a single accessed block whereas the density table tracks the density of regions with more than one accessed block. Splitting the RDTT into two tables allows for (a) reducing interference between regions with a single access from high-density regions, and (b) reducing energy per access over a unified and larger table as most of the RDTT accesses hit in the density table — most of accesses go to high-density regions (Section III).

The trigger and density tables are built as set-associative structures to minimize conflicts and are accessed using the region address. The region address is calculated by shifting the physical address by the number of region offset bits to the right.

The benefit of BuMP hinges on its ability to accurately identify DRAM accesses to high-density regions. To do so, BuMP's prediction mechanism relies on the observation that there is a high correlation between code and software objects that lead to high-density regions (Section III). To account for misalignment of a software object with the beginning of a region, the BuMP predictor augments the PC of the instruction triggering the memory access with the *offset*, defined as the distance between the triggering cache block and the beginning of the region. For a kilobyte-sized region, the offset is 4 bits. The PC and offset combination is similar to that used in spatial footprint prediction [4, 44].

The RDTT associates each entry with a *PC,offset* tuple. The PC is carried with LLC requests. When an RDTT region identified as having high access density is terminated due to an eviction, an entry is allocated in the bulk history table. Doing so simply requires indexing the bulk history table (BHT) with the *PC,offset* tuple and setting a valid bit.

On an LLC miss, the BHT is probed using *PC,Offset*. In case of a BHT hit (identified through a tag match), BuMP generates cache block requests for the entire region (except for the triggering block) and forwards them to the LLC.

Bulk DRAM read tracking example. Figure 7 illustrates the operation of the RDTT in detail. In event 1, the triggering instruction requests the block *A+2*, missing in both the density and the trigger tables. This results in an allocation in the trigger table. The allocated entry consists of the region address *A*, and the *PC,Offset*.

In event 2, the second access to region *A, A + 3,* hits in the trigger table, thus requiring the transfer of the entry to the density table. The density table entry enhances the trigger table entry with a bit vector (noted as *pattern*) that summarizes the cache blocks that have been accessed within a region. In our example, the third and fourth bits are set to one to indicate that the corresponding blocks of the region have been accessed. Subsequent access to region *A* in event 3 hits in the density table and updates the pattern accordingly.

Upon an LLC eviction within an active region or upon a table conflict, the corresponding entry in the density table is invalidated and a simple logic checks if the corresponding region has a high access density, by (a) counting the number of bits set in the pattern, and (b) comparing the resulting density to a pre-defined *threshold* (event 4 in Figure 7). If the resulting density is high, the *PC,Offset* is inserted into the bulk history table; otherwise, it is only invalidated.

C. Bulk DRAM Write Prediction and Streaming

BuMP relies on LLC dirty evictions to enforce bulk DRAM writes. In particular, BuMP leverages the observation that the first LLC dirty eviction is a good indicator that a high-density modified region will not be modified prior to its eviction from the last-level cache (Section III).

BuMP uses the region density tracking mechanism to identify modified high-density regions by extending the density and trigger tables with a dirty bit which is set upon receiving a write/writeback request from the L1 data cache. Upon an LLC dirty eviction within an RDTT region identified as modified high-density region, BuMP triggers bulk writeback requests to the LLC for the entire region. This operation is denoted by the dashed line labelled as *Dirty Region Address* in Figure 6.

In practice, we find that most region terminations in the density table occur due to a table conflict — before the first

Table II. Architectural parameters.

Parameter	Value
Technology	22nm, 2.5 GHz
CMP size	16 cores
Core	3-way OoO, 48-entry ROB and LSQ
L1-I/D caches	32KB, 2-way, 64B blocks 2-cycle load-to-use, 10 MSHRs
LLC	Unified, 4MB, 16-way, 64B blocks, 8 banks, 8-cycle hit latency Stride prefetcher with degree of four
NOC	16x8 crossbar, 5 cycles
Main Memory	16GB, 4 ranks per memory channel 2Gbit, x8, 8 banks per rank, 8KB row buffer 2 DDR3-1600 channels (Max. BW: 25.6GB/s) close- and open-row FR-FCFS policy [41] 64-entry transaction and command queues
t_{CAS}-t_{RCD}-t_{RP}-t_{RAS} t_{RC}-t_{WR}-t_{WTR}-t_{RTP} t_{RRD}-t_{FAW}	11-11-11-28 39-12-6-6 5-24

dirty LLC eviction within the region. Therefore, to minimize premature writebacks, BuMP employs the dirty region table (DRT) to keep track of cache-resident high-density modified regions that are evicted from the density table. The DRT is set associative and is accessed using the region address.

Upon an LLC dirty eviction, BuMP probes the DRT to check whether the cache block belongs to a high-density region. In case of a DRT hit, BuMP generates bulk write-backs (except for the already evicted block), forwards them to the LLC, and invalidates the corresponding DRT entry.

D. BuMP Configuration and Hardware Cost

BuMP employs a region size of 1KB, which corresponds to 16 cache blocks and allows for amortization of the DRAM page activation energy. The threshold for labeling regions as having high access density is eight cache blocks. This configuration balances the opportunity for energy savings by targeting a large fraction of DRAM accesses with limited tolerance for overfetch (Section V.E).

Memory controller. BuMP employs FR-FCFS open-row policy [41] with region-level address interleaving. BuMP maps an entire region to one DRAM row by using the addressing scheme *Row:ColumnHigh:Rank:Bank:Channel: ColumnLow:ByteOffset,* where ColumnHigh is 3 bits and ColumnLow is 7 bits (complements the region offset). Although BuMP's addressing scheme diminishes parallelism (accesses to consecutive cache blocks within a kilobyte-sized region are serialized) accesses to a high-density region serve as prefetches that counter-act the serialization delay.

Hardware cost. For our server workloads, an RDTT with 256-entry trigger table (2.5KB) and 256-entry (3KB) density table provides almost the same accuracy as a predictor with unlimited storage (results not shown) except for Software Testing. Both DRT and BHT employ 1024 entries each to minimize premature writebacks and to maximize prediction accuracy, for 4.25KB and 4.5KB of storage, respectively. All structures are 16-way set-associative. In total, BuMP requires ~14KB.

Table III. Power and energy for system components.

Parameter	Value
Core	Peak Dynamic Power: 700mW Leakage Power: 70mW
LLC	Read/Write Energy: 0.63nJ/0.70nJ Leakage Power: 750mW
NOC	Peak Dynamic Power: 55mW Leakage Power: 30mW
Memory Controller	Dynamic Power @ 12.8GB/s: 250mW
DRAM (per 2GB rank and 64-byte transfer)	Background Power: 540-770mW Activation Energy: 29.7nJ Read/Write Energy: 8.1nJ/8.4nJ I/O Termination (Read/RRead):1.5nJ/3.8nJ I/O Termination (Write/RWrite):4.6nJ/4.6nJ

V. EVALUATION

We first evaluate BuMP's accuracy in identifying high-density regions. Then, we examine the energy and performance implications of bulk memory streaming including BuMP's on-chip energy and bandwidth overheads. Finally, we include a comparison between BuMP and prior proposals on prefetching and eager writeback mechanisms.

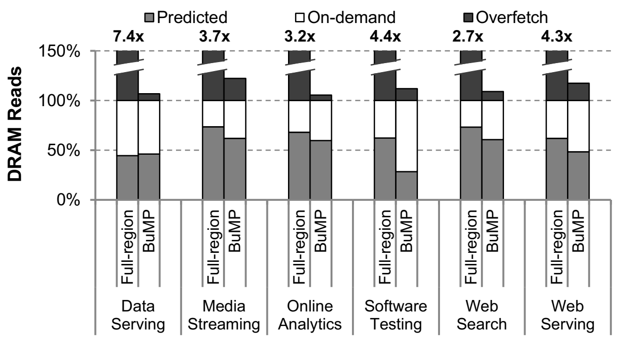
A. Methodology

Baseline systems. We evaluate BuMP in the context of a lean-core CMP with 16 cores, a modestly sized last-level cache, and a crossbar-based NOC that minimizes the delay to the LLC. Prior research has shown that large LLC capacities are counter-productive for server workloads and that a fast path to the LLC is critical to server processor performance [9, 10, 32, 43]. The chip is modelled in 22nm technology with high-performance process for all the components except the LLC which uses low-leakage process. The chip features two DDR3-1600 channels. Cores are modelled after a high-end mobile-class three-way out-of-order core [12]. Table II details the parameters for the processor.

Our baseline systems employ a stride prefetcher that predicts strided accesses if two consecutive addresses accessed are separated by the same stride, and prefetches the subsequent four cache blocks into the last-level cache.

We consider two memory controller configurations for our baseline systems: (a) FR-FCFS close-row policy with block-level address interleaving, referred to as *Base-close*, and (b) FR-FCFS open-row policy with region-level address interleaving, referred to as *Base-open*, same as BuMP's. Base-close maximizes channel-/rank-/bank-level parallelism by distributing consecutive cache blocks across channels/ranks/banks, thereby reducing serialization delays upon sequential accesses. This is accomplished by using the addressing scheme *Row:ColumnHigh:Rank:Bank:Channel: ColumnLow:ByteOffset,* where ColumnHigh is 7 bits and ColumnLow is 3 bits (complements the cache block offset).

We also evaluate SMS, a state-of-the-art spatial prefetcher [44]. SMS was originally designed for performance and was integrated next to the core at a storage cost of 60KB per core. In this work, we incorporate SMS next to the LLC. This optimization allows for higher energy-efficiency gains due to



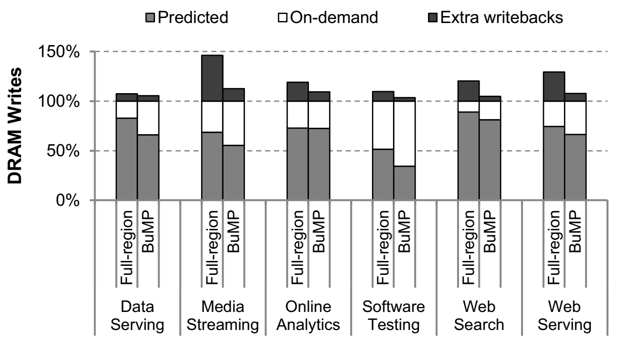


Figure 8. BuMP's prediction accuracy for DRAM reads (left) and DRAM writes (right).

higher prediction accuracy while reducing the storage costs as prediction metadata can be shared across cores. The memory controller is configured the same way as BuMP's.

Finally, we consider a state-of-the-art eager writeback mechanism [45], referred to as VWQ. The writeback mechanism generates eager writeback requests for three adjacent cache blocks upon a dirty LLC eviction. The memory controller is configured the same way as BuMP's.

Energy modeling framework. We use energy consumed per instruction as our energy-efficiency metric. To measure energy per instruction, we develop a custom energy modeling framework to include various system components, such as cores, network-on-chip (NOC), caches, memory controllers, and main memory. Our framework, summarized in Table III, draws on several specialized tools to maximize fidelity through detailed parameter control.

We estimate dynamic core power by scaling published measurements of dynamic power on a high-IPC workload by the ratio of the actual workload IPC and the reference IPC [1, 16, 29]. We measure core leakage power using McPAT [30]. We use CACTI's models to obtain LLC read and write energy estimates and we account for advanced LLC leakage reduction techniques [29, 38]. We obtain NOC power from McPAT, finding it to be negligible compared to other system components; hence, we assume constant NOC power. We use McPAT to measure memory controller's dynamic power at peak memory bandwidth. We estimate DRAM background power and energy per operation based on Micron models as implemented in DRAMSim2 and Micron data sheets [36, 37].

Simulation infrastructure. We evaluate BuMP using full-system simulation using *Flexus* [52]. Flexus models the SPARC v9 ISA and runs unmodified operating systems and applications. Flexus extends the *Simics* functional simulator with timing models of out-of-order cores, caches, on-chip protocol controllers and interconnect, and DRAM. DRAM is modeled by integrating DRAMSim2 [42] directly into Flexus. DRAMSim2 is instantiated with data borrowed from commercial DDR3 device specifications [36]. Table II details the simulated architecture.

We evaluate BuMP using contemporary server workloads. The workloads, taken from CloudSuite 2.0 [5, 9], include Data Serving, Media Streaming, Web Search, and Web Serving (frontend). We evaluate online analytics on a commercial database. In particular, we run a mix of queries 1, 6, 13, and 16 from the TPC-H benchmark on IBM DB2. Queries 1 and 6 are scan-bound, Query 16 is join-bound, and Query 13 exhibits a mixed behavior [14]. We also consider a large-scale scientific task that might run in cloud datacenters. In particular, we benchmark one instance per core of the Klee SAT Solver, an important component of CloudSuite 2.0's Software Testing workload.

For energy and performance evaluation of BuMP, we run cycle-accurate simulations using the SMARTS sampling methodology [56]. Our samples are drawn over the entire query execution for Online Analytics, and over an interval of 10-30 seconds of simulated time for the rest of the workloads. For each measurement, we launch simulations from checkpoints with warmed caches and branch predictors, and run 800K cycles (2M cycles for Data Serving) to achieve a steady state of detailed cycle-accurate simulation prior to collecting measurements for the subsequent 400K cycles. To measure performance, we use the ratio of the aggregate number of application instructions committed to the total number of cycles (including cycles spent executing operating system code); this metric has been shown to reflect system throughput [52]. Performance is measured at a 95% confidence level and an average error below 2%.

B. BuMP Prediction Accuracy

We evaluate BuMP's ability to exploit reads to high-density regions by measuring the number of cache blocks that are correctly fetched prior the processor's request, referred to as *predicted DRAM reads.* Similarly, for writebacks we measure the fraction of blocks that are timely written back to memory, referred to as *predicted DRAM writes*. To show the importance of triggering bulk transfers only upon accesses to high-density regions, we include a system that always triggers bulk transfers [31, 55], referred to as *Full-region*.

DRAM reads. Figure 8 (left) plots the fraction of predicted DRAM reads as well as the overfetch rate. Across all applications except for Software Testing, BuMP predicts 45-55% of DRAM reads at the cost of a small overfetch rate (5-22%) due to its ability to identify accesses to high-density regions and to enforce bulk transfers upon an LLC miss.

Table IV. BuMP's DRAM row buffer hit ratio.

Workload	Value	Workload	Value
Data Serving	54%	Software Testing	34%
Media Streaming	64%	Web Search	62%
Online Analytics	57%	Web Serving	56%

For Software Testing, BuMP predicts only 28% of DRAM reads, despite having a high fraction of memory accesses to high-density regions (52%). This disparity is attributed to the large number of active regions that compete for space in the region density tracking table (RDTT). Our analysis (not presented due to space constraints) of an RDTT with 256-entry and 2048-entry trigger and density tables showed that BuMP can predict up to 44% of DRAM reads.

Figure 8 also plots the coverage and overfetch rate of Full-region, a design that fetches the entire region upon the first access to the region. The coverage increases from 50% (BuMP) to 63% (Full-region), on average. However, the small increase in coverage comes at the cost of prohibitive overfetch (4.3x, on average) and cache thrashing. In case of workloads with a high number of accesses to low-density regions (Data Serving), Full-region loses coverage compared to BuMP due to excessive cache thrashing.

DRAM writes. Figure 8 (right) plots the BuMP coverage (i.e., fraction of predicted DRAM writes) as well as the extra writebacks that result from premature writebacks and pressure to the last-level cache due to overfetched cache blocks.

On average, BuMP predicts 63% of DRAM writes, matching our observation that most of the writeback traffic comes from high-density regions. Similar to DRAM reads, BuMP achieves the lowest coverage for Software Testing.

The increase in writeback traffic due to BuMP is low, less than 10% across all applications, as the first LLC dirty eviction within a high-density region is a good indicator that the region will not be modified (Section III.B).

The figure also plots the coverage of DRAM writes for Full-region. Full-region triggers bulk writeback requests to the last-level cache upon every dirty LLC eviction. As a result, the DRAM write coverage increases from 63% (BuMP) to 73% (Full-region). However, this increase comes at the cost of an increase in writeback traffic — from 7% to 22%, on average, due to premature writebacks. This is because the first eviction within a low-density region is not a good indicator as cache blocks within low-density regions belong to different software objects.

Overall. BuMP predicts 55% of all memory accesses at a small increase in memory traffic (11%). By doing so, BuMP recovers 87% of the locality that exists in high-density regions, thus achieving high row buffer hit ratio (Table IV).

C. Energy Efficiency

We examine the implications of bulk memory streaming on memory system energy efficiency. Figure 9 illustrates the memory energy consumed per access for BuMP normalized to our baseline systems: Base-close and Base-open.

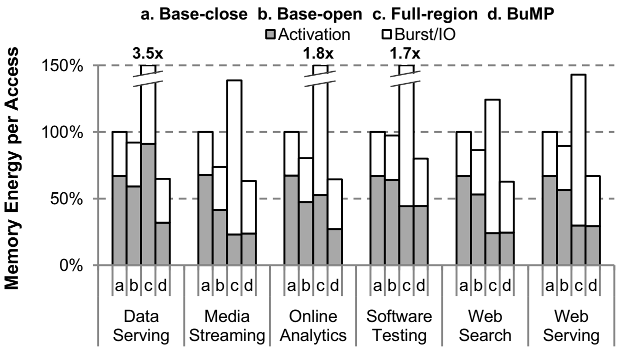


Figure 9. Memory energy per access for various systems.

Compared to Base-close, Base-open increases row buffer hit ratio to 21%, as it keeps pages open and prioritizes requests to already open pages (Figure 2). The increase in row buffer hit ratio allows for reducing memory energy by 14%. However, the row buffer hit ratio is low due to interleaving of accesses among cores, preventing the memory controller from exploiting high row buffer locality.

BuMP achieves high gains in energy efficiency compared to the baseline systems. On average, BuMP reduces energy per access by 34% and 23% compared to the close- and open-mode baseline systems, respectively, thanks to its ability to exploit high row buffer locality upon an access to a high-density region.

Figure 9 also plots the memory energy consumption of Full-region, showing that Full-region achieves the worst energy efficiency due to excessive overfetch. Furthermore, we find that for bandwidth-intensive workloads (Data Serving, Online Analytics, and Software Testing), contention in queuing structures in the LLC and the memory controller prevents requests within the same region to arrive within the memory controller's scheduling window. Thus, the memory controller cannot fully exploit the open-mode benefits of DRAM. Furthermore, this phenomenon results in a high number of additional row activations even for blocks that will not be accessed by the processor. In the extreme case (Data Serving), the combination of additional row activations and extreme LLC thrashing results in higher activation energy for Full-region than for the baseline systems.

D. Performance Implications

While our primary motivation in this work is to improve energy efficiency, we find that BuMP also improves system performance. Figure 10 plots the performance improvement of various systems over *Base-close,* a system that employs a stride prefetching scheme and maximizes DRAM parallelism. Base-open delivers 1-2% lower performance than Base-close as it delays precharging open pages, penalizing future accesses going to a different page in the bank. BuMP outperforms Base-close by 9% and Base-open by 11%, as bulk transfers from memory allow for fetching cache blocks prior to the processor's demand, thus hiding a fraction of off-chip memory stalls. Media Streaming exhibits the smallest performance improvement as most of the predicted memory

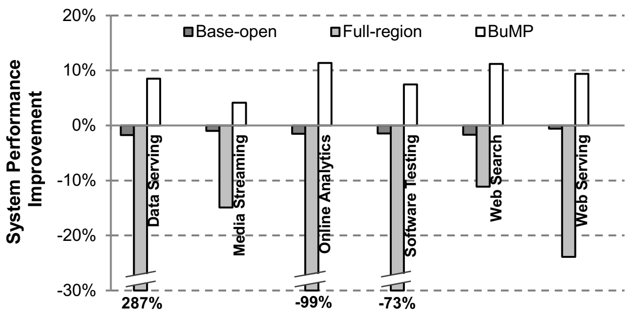


Figure 10. Performance improvement over Base-close.

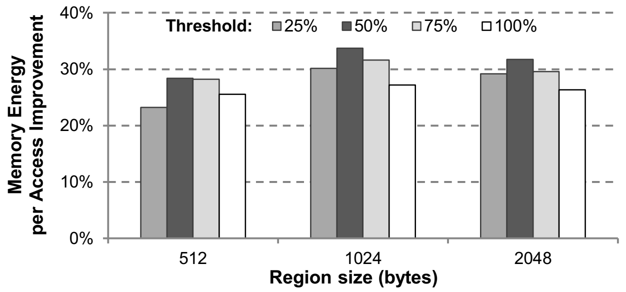


Figure 11. Memory energy efficiency sensitivity to BuMP's threshold and region size.

accesses exhibit high memory-level parallelism, and hence the out-of-order processor is able to overlap a significant fraction of off-chip memory stalls. Compared to SMS and VWQ (results not shown), BuMP improves performance by 3% and 9%, on average, due to higher read coverage.

On the other hand, Full-region hurts performance by 67% (up to 4x for Data Serving), due to oversaturation of memory bandwidth, highlighting the importance of avoiding bulk transfer operations upon accesses to low-density regions.

E. BuMP Design Space Exploration

Figure 11 illustrates the improvement in memory energy per access for various BuMP configurations. We vary both the region size and threshold for labeling regions as high-density. BuMP with region size of 1KB and threshold of eight cache blocks (noted as 50%) maximizes energy improvements as it targets a higher fraction of DRAM accesses, thus amortizing a higher degree of row activations compared to configurations with smaller regions while exhibiting lower overfetch rate than configurations with larger regions.

F. On-chip Bandwidth and Energy Overheads

We examine BuMP's on-chip bandwidth and energy overheads focusing on the LLC, NOC, and BuMP's structures.

Last-level cache. The LLC traffic increases due to (a) data overfetch (b) bulk read and writeback requests, and (c) extra writeback traffic. Figure 12 plots the LLC traffic of BuMP normalized to the baseline system. BuMP increases LLC traffic by 10%, on average. The increase in LLC bandwidth utilization has small impact on system performance as such a small traffic overhead is easily absorbed by the LLC.

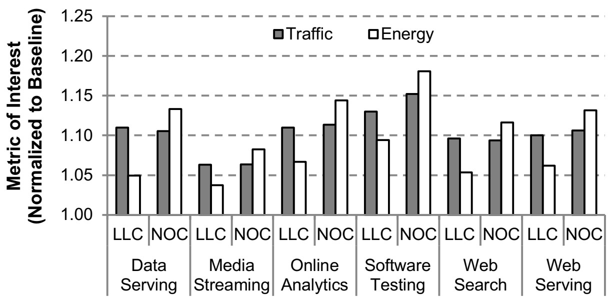


Figure 12. BuMP's LLC and NOC overheads.

The increase in LLC traffic comes at the cost of modest LLC energy overheads (7% on average). As the LLC energy accounts for only 3% of server energy, on-chip LLC energy overheads contribute to less than 0.3% of server energy while LLC power overheads are small (on average, 32mW).

Network-on-chip. The NOC traffic increases due to (a) L1-D requests to the LLC are augmented with the virtual address of the associated instruction (PC), (b) LLC data accesses and evictions have to be forwarded to BuMP, (c) BuMP-generated access and writeback requests have to be forwarded to LLC, and (d) data overfetch and extra write-back traffic. As shown in Figure 12, network traffic increases by 11%, on average. Similar to traditional server workloads [48], the NOC bandwidth utilization is low. Thus, the extra traffic is absorbed by the NOC, and consequently does not affect the system performance.

The increase in NOC traffic comes at the cost of 13% higher NOC energy — half of which is due to the transfer of the PC. As NOC accounts for a small fraction of server energy and power, BuMP's NOC energy and power overheads (~8mW) are negligible.

BuMP structures. BuMP introduces multiple structures for region density tracking, summarizing high-density modified regions, and bookkeeping the virtual address of instructions that trigger access to high-density regions. We model these structures using CACTI, and find energy per access to be ~2pJ for the region-density tracking tables and ~4pJ for the bulk history and dirty region tables. In total, energy consumed to access the structures accounts for 0.1% of server energy. On-chip power consumption is 10mW.

Summary. BuMP on-chip energy overheads account for less than 0.4% of server energy, negligible compared to the registered memory energy gains. On-chip power overheads are smaller than 50mW. Finally, on-chip bandwidth overheads are small, and do not impact performance.

G. Summary

In Figure 13, we plot the DRAM row buffer hit ratio and energy per access of BuMP, averaged across all workloads, compared to the close- and open-mode baselines (both with a stride prefetcher), a system with SMS, a system with VWQ, a system with SMS and VWQ, and an ideal system that exhibits maximal row buffer locality — i.e., it exploits all row buffer locality that exists in an access stream of a thread.

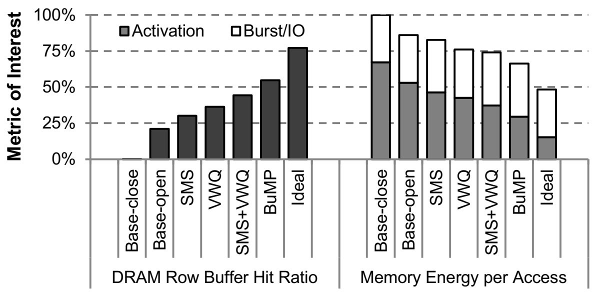


Figure 13. Comparison between BuMP and other systems.

The open-mode baseline system exploits a low degree of DRAM row buffer locality, thus achieving a row buffer hit ratio of 21%. The SMS-enabled system provides a row buffer hit ratio of 30% as it is able to predict irregular access patterns. However, the row buffer locality improvement is small as it targets only load-triggered reads. The VWQ-enabled system improves row buffer hit ratio to 36%, as it exploits a high degree of writeback locality. When SMS and VWQ are combined, row buffer hit ratio increases to 44%.

BuMP improves row buffer hit ratio to 55%, outperforming systems with SMS and/or VWQ. Compared to SMS, BuMP increases row buffer hit ratio by ~2x at 3x lower storage requirements. BuMP reduces storage cost as (a) it correlates triggering instructions with coarse-grained regions instead of individual cache blocks, and (b) few instructions trigger accesses to high-density regions. Compared to VWQ, BuMP increases row buffer hit ratio by an additional 19% as VWQ (a) exhibits poor read row buffer locality, and (b) performs lookups for a small number of cache blocks within an open row, thus failing to maximize open-mode gains. Finally, BuMP improves row buffer hit ratio by an additional 11%, compared to the SMS+VWQ system, as it maximizes open-mode gains upon accessing a high-density region.[1]

BuMP achieves high row buffer locality, improving memory energy efficiency by 23-34%. BuMP improves memory energy efficiency by 20% and 13% over SMS- and VWQ-enabled systems. Compared to SMS+VWQ, BuMP reduces memory energy per access by 10%. Finally, BuMP's memory energy efficiency is within 73% of the ideal system.

VI. Discussion

Memory access scheduling policy. FR-FCFS open-row policy [41] employed by BuMP can hurt system fairness in systems running multiprogrammed and parallel applications due to memory behavior variation [22, 39]. Server applications execute almost identical instruction sequences across requests and cores [20], and hence cores exhibit almost same memory behavior. Thus, row buffer locality can be maximized with small impact on system fairness. Nevertheless, scheduling policies that trade off row buffer hit ratio for system fairness [18] can be used complementary with BuMP.

1. BuMP can be combined with VWQ to exploit higher writeback locality by targeting memory writes to non-high-density memory regions.

Design scalability. BuMP can scale well with the CMP size, requiring the following modifications for larger configurations: (a) the region density tracking table needs to increase linearly with the number of cores to support a higher degree of interleaving among cores, and (b) the dirty region table needs to increase linearly with the LLC size to support a higher number of active modified regions.

Server virtualization. BuMP is applicable to server applications running in a virtualized environment. In doing so, the bulk history table needs to increase to accommodate the triggering instructions of all active workloads. Even with extreme heterogeneity (one workload per core), the storage requirement of the bulk history table is 72KB in a 16-core system, for total storage requirement of 5KB per core.

VII. Related work

Prior research has identified DRAM to be a significant contributor to server power. Based on the observation that off-chip memory bandwidth is often not utilized, prior work has either applied voltage frequency scaling to the memory controller and frequency scaling to the memory interface and devices [6, 7], or proposed using low-power, low-speed memory interfaces [34, 57]. However, emerging many-core processors saturate memory bandwidth [13, 17, 32], thereby mandating high-speed memory interfaces to maximize per-pin available bandwidth. Nevertheless, memory frequency scaling can be employed for server applications that exhibit lower off-chip memory bandwidth consumption.

Leveraging the observation that applications access only a few words within a cache block, researchers have proposed re-engineering the processor/memory interface to allow for activating only the DRAM chips at which the requested words are stored [1, 16, 58, 59]. While potentially effective in the server context as well, these proposals require disruptive changes to commodity memory technology.

Sudan et al. [46] have observed that desktop and parallel applications consist of a small number of OS pages (~100) that account for a high fraction of memory accesses (~25%) and that these accesses are centered around only a few cache blocks. To improve row buffer locality, the authors have proposed mechanisms that identify and merge such pages. However, server applications operate on vast datasets and a large number of accesses go to pages that were not previously visited. Any energy gains of this technique would come at the cost of high storage requirements as it has to track an enormous number of pages.

Prior work has proposed hardware and software mechanisms to guide page-based prefetching [3, 49]. Stealth prefetching requires keeping address-related metadata (hundreds of KBs per core) to decide what fraction of a page should be fetched after a certain number of cache blocks are accessed within the page [3]. Our work makes the observation that page density is code-correlated, thus allowing for fetching high-density pages with the first read to the page as opposed to multiple reads (Stealth prefetching) while introducing much lower storage overhead.

Guided region prefetching [49] uses compiler hints to direct both spatial and non-spatial (e.g., pointer-chasing) prefetches. However, rapid dataset changes in server applications present a challenge for static and profile-based approaches. Moreover, these approaches require application changes or recompilation, while our work is software transparent and adapts to changing application behavior.

Instruction-based predictors have been extensively used in the context of data prefetching [25, 44], cache-coherence action prediction [19], NOC power reduction [21], last-write prediction [50], on-chip granularity prediction [26], and off-chip bandwidth reduction [17, 58]. However, none of these works has targeted improving row buffer locality by predicting the memory page access density.

VIII. Conclusion

With lean-core servers being effective at minimizing processor energy, the energy-efficiency bottleneck is shifting to DRAM that must serve frequent memory accesses from many cores. Our analysis illustrated that server applications frequently access memory in bulk and that this behavior is not fully exploited by today's DRAM, resulting in high page activation energy. In response, we introduced BuMP, a predictor that identifies accesses to high-density pages and triggers bulk transfers upon the first read or write to a page. Using cycle-accurate full-system simulation of a 16-core server, we demonstrated that BuMP reduces memory energy per access by 23% and improves server throughput by 11%.

Acknowledgments

The authors would like to thank Haris Volos, Onur Kocberber, Djordje Jevdjic, Alexandros Daglis, Sotiria Fytraki, Cansu Kaynak, and the anonymous reviewers for the insightful feedback on earlier drafts of this paper. The work was partially supported by the Swiss National Science Foundation projects 200021_127021 and 200021_140051, the NanoTera project “YINS”, and the DeSyRe project.

References

[1] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi. Multicore DIMM: An energy efficient memory module with independently controlled DRAMs. *Computer Architecture Letters*, 8(1):5–8, Jan.-June 2009.

[2] L. A. Barroso and U. Holzle. 2009. The datacenter as a computer: An introduction to the design of warehouse-scale machine. 1st Edition. Madison: Morgan & Claypool.

[3] J. F. Cantin, M. H. Lipasti, and J. E. Smith. Stealth prefetching. In *International Conference on Architectural Support for Operating Systems and Programming Languages*, Oct. 2006.

[4] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos. Accurate and complexity-effective spatial pattern prediction. In *International Symposium on High Performance Computer Architecture*, Feb. 2004.

[5] CloudSuite 2.0. [Online]. http://parsa.epfl.ch/cloudsuite.

[6] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/ frequency scaling. In *International Conference on Autonomic Computing*, June 2011.

[7] Q. Deng, D. Meisner, L. E. Ramos, T. F. Wenisch, and R. Bianchini. MemScale: Active low-power modes for main memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2011.

[8] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *International Symposium on Computer Architecture*, June 2011.

[9] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic et al. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2012.

[10] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic et al. A case for specialized processors for scale-out workloads. *IEEE Micro*, 34(3):31–42, May-June 2014.

[11] B. Grot, D. Hardy, P. Lotfi-Lamran, C. Nicopoulos, Y. Sazeides, and B. Falsafi. Optimizing datacenter TCO with scale-out processors. *IEEE Micro*, 32(5):52–63, Sep.-Oct. 2012.

[12] L. Gwennap. Qualcomm Krait 400 hits 2.3GHz. *Microprocessor Report*, 27(1):1, 6–8, Jan. 2013.

[13] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, Jul.-Aug. 2011.

[14] N. Hardavellas, I. Pandis, R. Johnson, N. Mancheril, A. Ailamaki, and B. Falsafi. Database servers on chip multiprocessors: Limitations and opportunities. In *Biennial Conference on Innovative Data Systems Research*, Jan. 2007.

[15] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein. Scaling, power, and the future of CMOS. In *Electron Devices Meeting. IEDM Technical Digest. IEEE International*, Dec. 2005.

[16] M. K. Jeong, D. H. Yoon, D. Sunwoo, M. Sullivan, I. Lee, and M. Erez. Balancing DRAM locality and parallelism in shared memory CMP systems. In *International Symposium on High-Performance Computer Architecture*, Feb. 2012.

[17] 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 *International Symposium on Computer Architecture*, June 2013.

[18] D. Kaseridis, J. Stuecheli, and L. K. John. Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era. In *International Symposium on Microarchitecture*, Dec. 2011.

[19] S. Kaxiras and J. R. Goodman. Improving CC-NUMA performance using instruction-based prediction. In *International Symposium on High-Performance Computer Architecture*, Jan. 1999.

[20] C. Kaynak, B. Grot, and B. Falsafi. SHIFT: Shared history instruction fetch for lean-core server processors. In *International Symposium on Microarchitecture*, Dec. 2013.

[21] H. Kim, P. Ghoshal, B. Grot, P. V. Gratz, and D. A. Jimenez. Reducing network-on-chip energy consumption through spatial locality speculation. In *International Symposium on Networks-on-Chip*, May 2011.

[22] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Blter. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *International Symposium on Microarchitecture*, Dec. 2010.

[23] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan. Meet the walkers: Accelerating index traversals for in-memory databases. In *International Symposium on Microarchitecture*, Dec. 2013.

[24] J. G. Koomey. Growth in data center electricity use 2005 to 2010.

[25] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *International Symposium on Computer Architecture*, June 1998.

[26] S. Kumar, H. Zhao, A. Shriraman, E. Matthews, S. Dwarkadas, and L. Shannon. Amoeba-cache: Adaptive blocks for eliminating waste in the memory hierarchy. In *International Symposium on Microarchitecture*, Dec. 2012.

[27] C. J. Lee, V. Narasiman, E. Ebrahimi, O. Mutlu, and Y. N. Patt. DRAM-aware last-level cache writeback: Reducing write-caused interference in memory systems. *Technical Report*, Apr. 2010.

[28] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens. Eager writeback: A technique for improving bandwidth utilization. In *International Symposium on Microarchitecture*, Dec. 2000.

[29] S. Li, K. Chen, J. Ho Ahn, J. B. Brockman, and N. P. Jouppi. CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In *International Conference on Computer-Aided Design*, Nov. 2011.

[30] S. Li, J. Ho Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *International Symposium on Microarchitecture*, Dec. 2009.

[31] W.-F. Lin, S. K. Reinhardt, and D. Burger. Reducing DRAM latencies with an integrated memory hierarchy design. In *International Symposium on High-Performance Computer Architecture*, Jan. 2001.

[32] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi. Scale-out processors. In *International Symposium on Computer Architecture*, June 2012.

[33] S.-L. Lu, T. Karnik, G. Srinivasa, K.-Y. Chao, D. Carmean, and J. Held. Scaling the "memory wall". In *International Conference on Computer-Aided Design*, Nov. 2012.

[34] K. T. Malladi, F. A. Nothaft, K. Periyathambi, B. C. Lee, C. Kozyrakis, and M. Horowitz. Towards energy-proportional datacenter memory with mobile DRAM. In *International Symposium on Computer Architecture*, June 2012.

[35] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating server idle power. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2009.

[36] Micron 2Gb: x4, x8, x16 DDR3 SDRAM. [Online]. http://www.micron.com/-/media/documents/products/data sheet/dram/ddr3/2gb_ddr3_sdram.pdf.

[37] Micron DDR3 SDRAM Power Calculation. [Online]. http://www.micron.com/products/support/power-calc.

[38] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *International Symposium on Microarchitecture*, Dec. 2007.

[39] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair queuing memory systems. In *International Symposium on Microarchitecture*, Dec. 2006.

[40] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazieres et al. The case for RAMClouds: Scalable high-performance storage entirely in DRAM. *SIGOPS Operating Systems Review*, 43(4):92–105, Dec. 2009.

[41] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *International Symposium on Computer Architecture*, June 2000.

[42] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A cycle accurate memory system simulator. *Computer Architecture Letters*, 10(1):16 –19, Jan.-June 2011.

[43] J. L. Shin, H. Park, H. Li, A. Smith, Y. Choi, H. Sathianathan, et al. The next-generation 64b SPARC core in a T4 SoC processor. In *International Conference on Solid-State Circuits*, Feb. 2012.

[44] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Spatial memory streaming. In *International Symposium on Computer Architecture*, June 2006.

[45] J. Stuecheli, D. Kaseridis, D. Daly, H. C. Hunter, and L. K. John. The virtual write queue: Coordinating DRAM and last-level cache policies. In *International Symposium on Computer Architecture*, June 2010.

[46] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis. Micro-Pages: Increasing DRAM efficiency with locality-aware data placement. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2010.

[47] N. A. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. Rethinking DRAM design and organization for energy-constrained multi-cores. In *International Symposium on Computer Architecture*, June 2010.

[48] S. Volos, C. Seiculescu, B. Grot, N. Khosro Pour, B. Falsafi, and G. De Micheli. CCNoC: Specializing on-chip interconnects for energy efficiency in cache-coherent servers. In *International Symposium on Networks-on-Chip*, May 2012.

[49] Z. Wang, D. Burger, K. S. McKinley, S. K. Reinhardt, and C. C. Weems. Guided region prefetching: A cooperative hardware/software approach. In *International Symposium on Computer Architecture*, June 2003.

[50] Z. Wang, S. M. Khan, and D. A. Jimenez. Improving writeback efficiency with decoupled last write prediction. In *International Symposium on Computer Architecture*, June 2012.

[51] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos. Temporal streams in commercial server applications. In *International Symposium on Workload Characterization*, Sept. 2008.

[52] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, July-Aug. 2006.

[53] B. Wheeler. Tilera sees opening in clouds. *Microprocessor Report*, 25(7):13–16, July 2011.

[54] B. Wheeler. Cavium joins ARM-server fray. *Microprocessor Report*, 2r65(8):1,5–6,16, Aug. 2012.

[55] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. S. Lee. An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth. In *International Symposium on High Performance Computer Architecture*, Jan. 2010.

[56] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *International Symposium on Computer Architecture*, June 2003.

[57] D. H. Yoon, J. Chang, N. Muralimanohar, and P. Ranganathan. BOOM: Enabling mobile memory based low-power server DIMMs. In *International Symposium on Computer Architecture*, June 2012.

[58] D. H. Yoon, M. K. Jeong, M. Sullivan, and M. Erez. The dynamic granularity memory system. In *International Symposium on Computer Architecture*, June 2012.

[59] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-Rank: Adaptive DRAM architecture for improving memory power efficiency. In *International Symposium on Microarchitecture*, Dec. 2008.