SELF: A High Performance and Bandwidth Efficient Approach to Exploiting Die-stacked DRAM as Part of Memory

Yuhua Guo[¶], Qing Liu[§], Weijun Xiao[¶], Ping Huang[‡], Norbert Podhorszki[†], Scott Klasky[†], Xubin He[¶‡]
[¶]Virginia Commonwealth University, [§]New Jersey Institute of Technology,
[‡]Temple University, [†]Oak Ridge National Laboratory
[¶]{guoy4,wxiao}@vcu.edu, [§]qing.liu@njit.edu,
[‡]{templestorager,xubin.he}@temple.edu, [†]{pnorbert,klasky}@ornl.gov

***Abstract*—Die-stacked DRAM (a.k.a., on-chip DRAM) provides much higher bandwidth and lower latency than off-chip DRAM. It is a promising technology to break the "memory wall". Die-stacked DRAM can be used either as a cache (i.e., DRAM cache) or as a part of memory (PoM). A DRAM cache design would suffer from more page faults than a PoM design as the DRAM cache cannot contribute towards capacity of main memory. At the same time, obtaining high performance requires PoM systems to swap requested data to the die-stacked DRAM. Existing PoM designs fall into two categories — line-based and page-based. The former ensures low off-chip bandwidth utilization but suffers from a low hit ratio of on-chip memory due to limited temporal locality. In contrast, page-based designs achieve a high hit ratio of on-chip memory albeit at the cost of moving large amounts of data between on-chip and off-chip memories, leading to increased off-chip bandwidth utilization and significant system performance degradation.**

To achieve a similar high hit ratio of on-chip memory as page-based designs, and eliminate excessive off-chip traffic involved, we propose SELF, a high performance and bandwidth efficient approach. The key idea is to SElectively swap Lines in a requested page that are likely to be accessed according to page Footprint, instead of blindly swapping an entire page. In doing so, SELF allows incoming requests to be serviced from the on-chip memory as much as possible, while avoiding swapping unused lines to reduce memory bandwidth consumption. We evaluate a memory system which consists of 4GB on-chip DRAM and 12GB off-chip DRAM. Compared to a baseline system that has the same total capacity of 16GB off-chip DRAM, SELF improves the performance in terms of instructions per cycle by 26.9%, and reduces the energy consumption per memory access by 47.9% on average. In contrast, state-of-the-art line-based and page-based PoM designs can only improve the performance by 9.5% and 9.9%, respectively, against the same baseline system.

I. Introduction

Recent advances in die-stacking technology have made it possible to integrate a large amount of DRAM in the same package of a processor. A processor and on-chip DRAM are interconnected by a high-density, low-latency through-silicon vias (TSVs). This technology has the potential to overcome the memory wall problem [1] by providing an order of magnitude higher bandwidth and much lower latency for on-chip DRAM. Prior work [2], [3], [4], [5], [6], [7] has proposed using die-stacked DRAM as a hardware-managed last-level cache (i.e., DRAM cache). As the technology for manufacturing die-stacked DRAM matures, the size of die-stacked DRAM could be tens of gigabytes by integrating multiple DRAM stacks on a 2.5D interposer [8], [9]. Therefore, using die-stacked DRAM as a DRAM cache would squander a large fraction of total memory space as the DRAM cache is invisible to the OS. Without fully exploiting the memory capacity offered, applications with a large working set would suffer a higher rate of page faults and therefore slowdown due to frequent accesses to backend storage.

An alternative to using die-stacked DRAM as a cache is to use it as part of an OS-visible memory space (i.e., PoM). In such a heterogeneous memory system, data residing in on-chip DRAM is serviced at high bandwidth and low latency, while data residing in off-chip DRAM is serviced at low bandwidth and high latency. However, naively treating on-chip DRAM as a part of memory space renders the PoM design less effective. To obtain high performance, on-chip DRAM needs to play two roles at the same time in a PoM architecture. The on-chip DRAM is not only a part of memory space but also a cache for off-chip DRAM. In other words, requested data is swapped or migrated to on-chip DRAM and victim data is swapped out to off-chip DRAM. This swapping process can be done by either the OS or hardware. For OS-managed approaches, the OS needs to monitor all page usage and migrate hot pages to the on-chip DRAM. OS-invoked page migrations result in page table updates and TLB shoot-downs, which are costly operations. Therefore, the page migrations under the OS control cannot occur frequently so that hot pages in a short period of time could not be migrated to the on-chip DRAM, resulting in performance loss. In contrast, in a hardware-managed PoM architecture, the migration is transparent to OS, and can be initiated at anytime when data is required. Hence, the hardware-managed PoM is a promising design and we only consider hardware-managed PoM in this paper.

Current PoM designs [10], [11] fall into two categories based on the granularity at which they swap data: line-based and page-based (or segment-based). The line-based design uses off-chip bandwidth efficiently as all swapped lines are demanded. However, the line-based design could suffer from

low hit ratio due to poor temporal locality at the main memory layer as highly referenced cache lines have already been filtered out by L1 and L2 caches. The page-based design swaps data at a coarser granularity (typically 1-4KB), thus achieving a higher hit ratio by exploiting spatial locality in the large granularity. However, the page-based design would waste precious off-chip bandwidth as some lines may not be touched before they are swapped out. The inefficient usage of off-chip bandwidth could lead to performance degradation, especially for data-intensive applications.

In this paper, we make the following contributions:

- We propose SELF, a high performance and memory bandwidth efficient approach to using die-stacked DRAM as a part of memory. To take advantage of both line-based and page-based PoM designs while avoiding their respective drawbacks, SELF only swaps those lines in a requested page that are likely to be accessed according to its page footprint. In doing so, SELF enables most incoming requests to be serviced from on-chip memory while avoiding swapping unused lines to save memory bandwidth.
- We redesign TLB to record page footprints and predict the lines in a page that are likely to be accessed again. In order to achieve partial swap of a page, we design two remapping tables with different granularities, remapping page table (RPT) and remapping line table (RLT). RPT is designed to track page locations after swapping and the RLT records every line's physical location within a page. Moreover, RPT is reused to predict line locations. If a requested line is predicted in off-chip memory, SELF will access the predicted location in parallel with the RLT access to hide the latency of RLT. As a result, the RPT predicts the physical location of a line with an 85.5% accuracy.
- We evaluate our system composed of 4GB on-chip DRAM and 12GB off-chip DRAM. Compared to the baseline system of 16GB off-chip DRAM only, the state-of-the-art line-based [11] and page-based [10] PoM designs improve performance by 9.5% and 9.9%, respectively, whereas SELF improves performance by 26.9% and reduces energy per memory access by 47.9% on average.

The rest of the paper is organized as follows. We introduce the background of die-stacked DRAM and analyze advantages and drawbacks of line-based and page-based PoM designs in Section II. We detail our design in Section III. We describe our evaluation methodology and present our experimental results in Section IV. We discuss related work in Section V and conclude in Section VI.

II. Background and Motivation

As more cores are integrated into many-core chips to improve processing capabilities and parallelism, the growth in core count requires a commensurate increase in memory bandwidth. However, memory speeds have not kept pace with CPU performance scaling, which has led to the memory wall problem [1]. Die-stacked DRAM has been advocated as a promising technology to break the memory bandwidth and latency wall. It provides an order of magnitude higher bandwidth and lower access latency than off-chip DRAM due to the dense TSVs buses [12]. However, the capacity of die-stacked DRAM is insufficient to fully replace off-chip DRAM due to technological constraints [4], [11]. Thus, die-stacked DRAM and off-chip DRAM will co-exist in future systems, and die-stacked DRAM can be used either as a cache or as a part of main memory. Most prior work [3], [4], [5], [6], [7], [13], [14], [15] advocates using die-stacked DRAM as a giant cache between the last level cache (LLC) and main memory, and copes with challenges of tag storage overhead, hit ratio, hit/miss latency and off-chip traffic etc. However, DRAM cache is invisible to the OS. In other words, DRAM cache cannot contribute towards the main memory capacity, which could lead to non-negligible performance loss due to increased page faults, especially for modern server applications with a large working set size (WSS). As the technology for manufacturing die-stacked DRAM matures, the size of die-stacked DRAM in each package could be up to tens of gigabytes. In this case, using die-stacked DRAM as a cache could waste a large fraction of total memory space.

Therefore, researchers have proposed using die-stacked DRAM as a part of memory [11], [10], [16], [17] instead of a cache. However, we can only get marginal benefits if the die-stacked DRAM is naively treated as a part of memory [17]. To obtain high performance, highly referenced pages or lines need to be migrated to die-stacked DRAM to take advantage of its high bandwidth and low access latency. This migration process can be performed by the OS or hardware.

A. OS-managed PoM

OS-managed PoM approaches need to track page usage to identify highly referenced pages. For an on-chip DRAM with a capacity of N pages, the OS should choose the top-N most referenced pages and map them into the on-chip memory at run-time. However, the operating system has a limited capability to get such information from the page table as the reference bit in each page table entry (PTE) cannot differentiate which pages are most referenced. A typical solution is to use a counter in each PTE to record the number of LLC misses per page, which would require extra hardware support [18]. At the end of each epoch or interval (e.g. 100K cycles), the OS sorts pages based on the access count and migrates the top-N hottest pages which are resident in off-chip memory to the on-chip memory. At the same time, these pages which are resident in on-chip DRAM but not belonging to the top-N hottest pages are migrated back to off-chip DRAM. Then the OS has to update the page table to reflect new mappings and invalidate corresponding translation lookaside buffer (TLB) entries (i.e., TLB shoot-down) for consistency. Therefore, the data migration under the OS control results in high overhead of sorting, copying pages back and forth between on-chip and off-chip memories, and TLB shoot-downs. As such, OS-managed migration cannot be performed

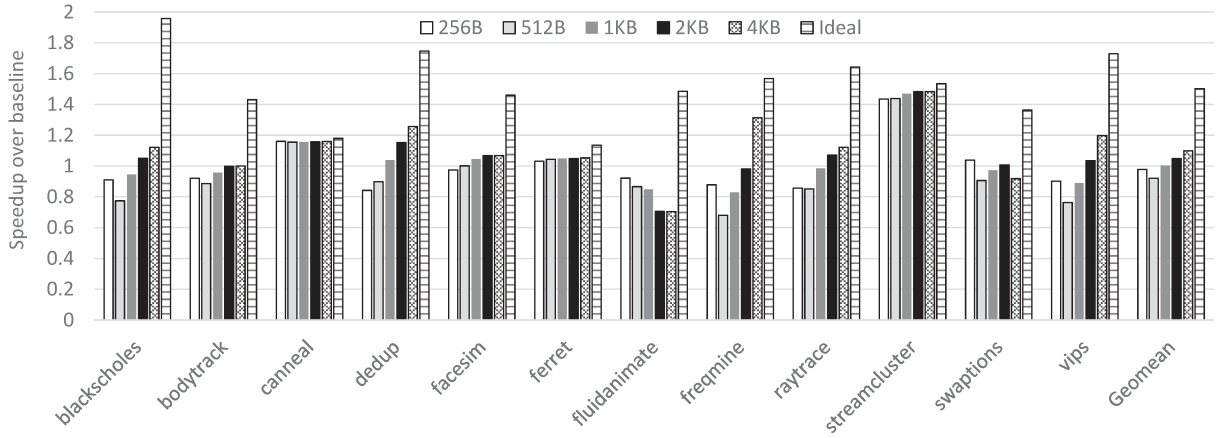


Fig. 1: The performance of a state-of-the-art page-based PoM design [10] with different page sizes. All requests are serviced from on-chip memory in the ideal case. On average, the page-based design performs best at the page size of 4KB.

frequently, which could miss many opportunities to improve performance by migrating pages that are highly referenced in short periods of time. In addition, OS-managed data migration can only occur at a page granularity (typically 4KB). When a significant fraction of data lines are not referenced, such page granularity transfers become very inefficient in terms of off-chip memory bandwidth. In a word, OS-managed PoM approaches could neither exploit the full benefits of on-chip DRAM at a coarse-grained interval nor utilize the off-chip memory bandwidth efficiently at a page granularity.

B. Hardware-managed PoM

Hardware-managed PoM can avoid page table updates, TLB shoot-downs and page sorting by maintaining a hardware-managed remapping table, which records real locations after swapping. The remapping table is updated by hardware without involving the OS after each data migration completes. Hence, the data migrations under hardware control could occur whenever the requested data is not resident in the on-chip memory, which could potentially improve the system performance. According to swapping granularity, hardware-managed PoM designs fall into two categories: line-based and page-based.

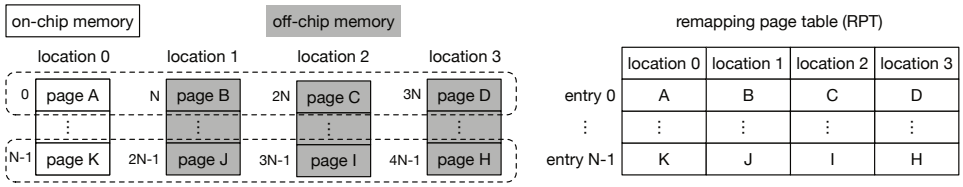
1) Line-based PoM: The line-based design swaps data at a line granularity. The small line granularity ensures a low utilization of off-chip bandwidth, since all lines swapped into the on-chip memory are demanded without wasting off-chip bandwidth. However, the line-based design falls short of exploiting abundant spatial locality, and temporal locality at the main memory layer is usually very poor as it has already been filtered out by the L1 and L2 caches. As a result, the line-based design suffers from a high miss rate of on-chip memory, accessing off-chip memory with low bandwidth and long access latency frequently.

2) Page-based PoM: The page-based design swaps data at a page (1-4KB) granularity. Compared to the line granularity, the large page granularity exploits abundant spatial locality, which could result in a higher hit ratio. Hence, the performance can be potentially improved as most misses in the LLC will likely be serviced from the on-chip memory at high memory bandwidth and low access latency. However, the large swap granularity could increase off-chip traffic as some unneeded lines of a swap-in page are also swapped in on-chip memory. The increase of off-chip traffic prolongs latency of off-chip accesses as the off-chip bandwidth is often overloaded, thus offsetting the benefit of hight hit ratio. Figure 1 shows the performance of state-of-the-art page-based PoM design while varying page size from 256B to 4KB. The results show that there is no one page size that can fit all cases. Smaller page sizes even degrade performance in some applications (e.g., *dedup*). On average, the page-based design performs best at the page size of 4KB. However, there is still a big performance gap between the best page-based design and the ideal case. The root cause is that the coarse page granularity cannot avoid wasting off-chip bandwidth, leading to saturation. In the case of off-chip bandwidth saturation, all requests to off-chip memory need to wait a long time in the transaction queue of memory controller, and are serviced sequentially, significantly degrading system performance.

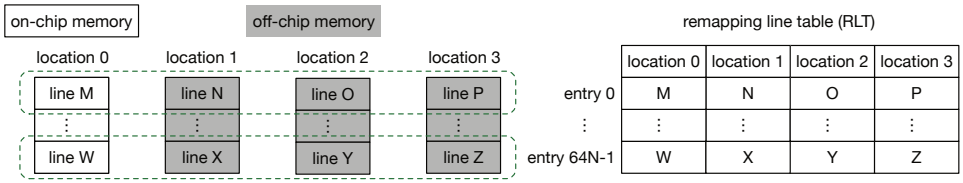
In conclusion, the line-based design uses off-chip memory bandwidth efficiently but suffers from a low hit ratio due to limited temporal locality. In contrast, the page-based design provides a higher hit ratio by exploiting spatial locality, while wasting off-chip bandwidth due to swapping useless data. To take advantages of both line-based and page-based designs while avoiding their drawbacks, we propose SELF, a high performance and bandwidth efficient approach to using on-chip DRAM as a part of memory.

III. SELF: Architecture and Design

In order to gain similar high hit ratio as the page-based design while avoiding unnecessary off-chip traffic due to swapping useless data lines, we propose to selectively swap data lines of a page that are likely accessed during the page's residency in on-chip memory instead of blindly swapping an



(a) Direct page-remapping and remapping page table.



(b) Direct line-remapping and remapping line table.

Fig. 2: Direct remapping and corresponding remapping tables.

entire page. However, current remapping table of the page-based design does not support partial swapping as it cannot differentiate which data line has been swapped due to its page granularity.

A. Remapping Table Design

To achieve partial swapping of a page, we design two remapping tables at the granularity of page and line, called remapping page table (RPT) and remapping line table (RLT), respectively. The RPT is used to track pages' physical locations after swapping while the RLT records all data lines' physical locations in each page. In other words, the RPT tells where the requested page is and the RLT further indicates where the requested line is. With the cooperation of RPT and RLT, SELF can only swap those lines in a page that are likely accessed in the future to save off-chip bandwidth. In the PoM design, each LLC miss must first look up the remapping table to determine the actual physical location of the requested data. Then the memory controller can decide where to fetch the requested data from either on-chip memory or off-chip memory. In theory, data in the off-chip memory can be swapped to any location of on-chip memory in a similar way to a fully associative cache. In this case, we may need to search the entire remapping table in the worst case. As accessing the remapping table is on the critical path, searching the whole remapping table could cause excessive latency. To reduce the remapping table lookup time, we adopt direct-remapping, which is similar to the direct mapped concept in a cache design. In other words, a page or a data line in the off-chip memory can only be swapped to a specific location in the on-chip memory.

Figure 2 shows direct remapping applied in an example of memory system, in which the on-chip memory has a capacity of N pages, and the off-chip memory has 3N pages. Figure 2a shows direct page-remapping and corresponding RPT. Under the direct page-remapping, a page is only allowed to be swapped with another page mapped to the same entry of the RPT. For example, page A, page B, page C, page D are mapped to entry 0 of the RPT, thus they can be swapped with each other. Figure 2b shows direct line-remapping and corresponding RLT. It works in a similar way of the direct page-remapping. Due to the use of direct remapping, every remapping information can be retrieved with a single access to a corresponding entry. The RPT is indexed by the least significant $log_2 N$ bits of the requested physical page number (PPN) and the RLT is indexed by the least significant $log_2 64N$ bits of the requested line address. However, both RPT and RLT are on the critical path, each LLC miss needs to go through them sequentially. How to reduce or hide access latency of these two remapping tables plays an important role to the system performance. The RPT is small due to the use of coarse granularity. For the evaluated memory system consisting of 4GB on-chip DRAM and 12GB off-chip DRAM, the number of the RPTs entries is one million and each entry is a four elements tuple with two bits for each element. Therefore, the size of RPT is 1MB. However, the RPT could be more than ten megabytes as the on-chip DRAM keeps increasing. In order to be scalable and reduce access latency, we store the RPT in the on-chip memory and use a small SRAM (32KB), called RPT cache, to cache it. The RPT cache is indexed by the least significant $log_2 K$ bits of the physical page number, where K is the total number of sets in the RPT cache. And the least significant $log_2 N$ bits of the PPN is used as a tag. The RPT cache is expected to gain a high hit ratio because of a good spatial locality provided by the page granularity. In contrast, the RLT has a poor spatial locality due to the use of fine-grained granularity and it is very large (64MB in our evaluated system). Therefore, we choose to store the RLT in the on-chip memory only without caching it, which causes an extra access to the on-chip memory as each request must first look up the RLT to determine the physical location of the requested line. To hide the access latency of RLT, SELF co-locates each data line with its corresponding RLT entry. This technique is also used in [3], [11]. To implement the co-located RLT, we sacrifice memory space of one data line in each 2KB DRAM row, and use it to store RLT entries for

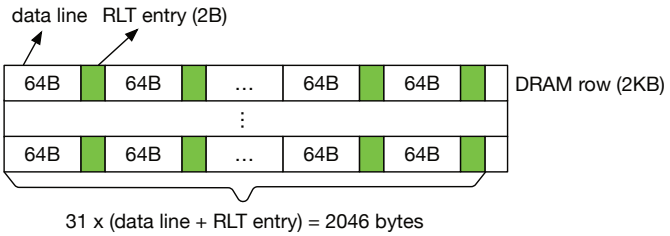


Fig. 3: The data layout of on-chip memory after co-locating each data line with its corresponding RLT entry.

other 31 data lines. Thus, each RLT entry can have up to 2 bytes, leaving 2 bytes unused in each row. Figure 3 shows the data layout of on-chip memory after co-locating each data line with its corresponding RLT entry. In order to support the co-located RLT, we reserve 1/32 off-chip memory space. The reserved space could be used for data that will not be swapped. Therefore, for a requested line address X in on-chip memory, its actual physical address equals to $X + X/31$. In doing so, a data line and its corresponding RLT entry can be streamed out in one access. If the requested line is present in the on-chip memory by checking the RLT entry, we can directly use the data line just read out together with the RLT entry, without any extra access to the on-chip memory. If the RLT entry identifies that the requested line is in the off-chip memory, then a second access for the desired location in off-chip memory is performed.

B. Page Swapping

In the direct page-remapping, some pages (e.g., 4 pages in our system) are mapped to the same entry, and they compete for one location in the on-chip memory. When and which page should be swapped to the on-chip memory depends on the swapping policy. Ideally, the swapping policy should choose the hottest page in a certain period of time to be swapped to the on-chip memory, so that most incoming requests can be serviced from the on-chip memory. The most direct way is to record the number of accesses to each page during an interval by associating a counter with each page, then choose the page with the highest number of accesses to be swapped to the on-chip memory. However, the ideal swapping policy is too costly to implement in hardware. The simplest way is to swap the page to on-chip memory once it is demanded, which could cause frequent page swapping, especially when two pages mapped to the same entry are accessed in an interleaved fashion. As a result, the requested two pages are swapped back and forth, leading to saturating the off-chip bandwidth and wasted energy. In fact, pages residing in off-chip memory, called off-chip pages, are competing with the page residing in on-chip memory, called on-chip page. An off-chip page should be swapped to the on-chip memory as long as it is hotter than an on-chip page. Based on that, we employ a cost effective way by using a competing counter (CC) [10] to record the relative number of accesses. If the requested page is in the on-chip memory, the CC is decreased by 1, otherwise it is increased by 1. Once the CC is larger than the swap threshold, the off-chip page which is being accessed is swapped with the on-chip page. In our system the swap threshold is set to 8 (Section IV-G), so each CC only needs 4 bits. Due to address alignment, each CC is allocated 8 bits, some of which can be reserved for future use. To track page activity, each RPT entry is appended a CC.

C. Page Footprints

To achieve partial swap of a page, we need to predict which data lines in the page will be requested between two consecutive swap-in operations of the page and only swap those lines to reduce memory bandwidth consumption when a page swapping occurs. A lot of previous work [19], [20], [21], [22] demonstrate repetitive access patterns in commercial workloads. In other word, a data line that was accessed in current interval will likely be accessed in next interval. A page footprint records which data lines were accessed between two consecutive swap-in operations of the page. Based on that, we use page footprints to predict which lines in a page are likely accessed, which is similar to the Footprint Cache [4]. However, in main memory there is no tag array that can be used to record page footprints. Therefore, we redesign TLB to add a bit vector in each TLB entry to record a page footprint and also add a bit vector in each page table entry accordingly. The number of bits in a bit vector is equal to the page size divided by the data line size, thus each page footprint is typically 64 bits. If each core has a TLB of 32 entries, the storage cost of page footprints in the TLB is 256 bytes per core. The additional storage cost for page footprints in the page table is negligible since the page table is stored in main memory or disk. As all data requests have to lookup TLB, SELF can set the corresponding bits of the bit vector without extra accesses to the TLB, and the page footprint obtained from TLB can be directly used in the page swapping process without extra accesses to the page table either. Thus, recording page footprints in the TLB is a cost effective way.

However, recording page footprints in the TLB could cause incoherent problem in a multi-core system. As the page footprints' incoherency would not affect programs' correctness, we do not take any coherent action except any of these two cases occurs to reduce maintenance overhead of page footprints. 1) When a TLB entry is evicted, the page footprint from TLB bitwise OR with its corresponding page footprint in the page table and the result is saved in the page table but not synchronized with TLBs to improve prediction accuracy; 2) When a page is swapped to on-chip memory, its page footprints both in TLBs and the page table are reset to store the latest access information to reduce overpredictions (i.e. a data line is not requested but it was predicted). In either case, the related page table entry is updated. We update the page table through a system call to the page table walk. In the second case, the physical address is converted to a virtual address before the page table walk. We maintain a modified inverted page table to translate physical addresses to virtual addresses. Different virtual page mapped to the same physical

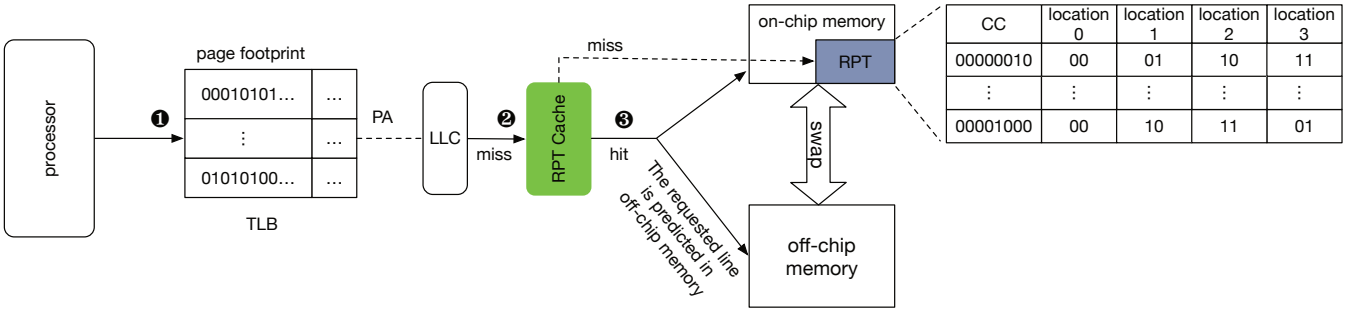


Fig. 4: Overview of SELF architecture. When the competing counter (CC) is larger than the swap threshold, SELF selectively swaps lines in the requested page according to its page footprint. Otherwise, SELF uses page location to predict the requested line location to reduce latency of off-chip accesses.

page are stored in a linked list. Since updating the page table is not on the critical path and the page table can be accessed concurrently, the impacts of updating page table is negligible on the performance.

D. Line Location Prediction

As discussed in Section III-A, we can save one access to the on-chip memory when the requested line is resident in on-chip memory by streaming the data line and RLT entry together. However, for the off-chip access (i.e., the requested line is resident in the off-chip memory), the RLT in the on-chip memory is accessed first to get the physical location of the requested line, then the off-chip memory is accessed according to the physical location. In this case, the off-chip access is serialized and occurs only after accessing on-chip memory. To break the serialized off-chip accesses, we reuse the RPT to predict line locations as the RPT itself has the information about page locations and most data lines in a page are likely to have the same location as its page. We use page locations obtained from the RPT to predict the locations of requested lines. If the line is predicted to be in off-chip memory, the predicted location in the off-chip memory will be accessed in parallel with on-chip access. If the prediction is correct, the line from off-chip location is used and the latency of RLT access is hidden. If the requested line is found in on-chip memory by checking the RLT entry, then the prediction is ignored. In the worst case, the requested line is in off-chip but it is predicted to be in a wrong off-chip location, a second access to the off-chip location still need to be performed.

E. Put Everything Together

The SELF integrates all techniques presented in above sections, as shown in Figure 4, where the on-chip memory accounts for a quarter of the total capacity. For other ratios, SELF works similarly, but the storage overhead of the RPT and RLT may be slightly different. ❶ A request from the processor accesses the TLB to get its physical address (PA) of the requested data, and set corresponding bit in its page footprint at the same time. ❷ The RPT cache is accessed if the request is missed in the LLC. If the request is a cache miss, then a corresponding RPT entry is loaded to the RPT cache. Otherwise, a RPT entry related to the request is accessed. According the real location of the requested page, the CC of the accessed RPT entry is updated. If the CC is larger than the swap threshold, SELF selectively swaps those lines of the requested page to on-chip memory according to its page footprint obtained from step ❶ and resets its CC and page footprint. Otherwise, SELF uses the page location to predict the requested line location. If the requested line is predicted in off-chip memory, the predicted location in off-chip will be accessed in parallel with on-chip access, or only on-chip access will be issued if the requested line is predicted in on-chip memory. ❸ The RLT entry and a data line are returned together from the on-chip memory. According to the RLT entry, if the real location of the requested line is in the on-chip memory, the data line is used to service the request directly and ignore any prediction. If the real location of the requested line is in the off-chip memory and was predicted correctly, memory controller only needs to wait until the requested data returned from the off-chip memory. In this case, latency of retrieving the RLT is avoided as it was issued in parallel with off-chip access to the predicted location in step ❷. However, if the real location of the requested line is in the off-chip memory and was wrongly predicted, then an access to the real location in the off-chip memory is performed.

In a word, all techniques applied in SELF work in concert to enable most incoming requests to be serviced from on-chip memory while avoiding swapping unused lines to save memory bandwidth. SELF also reduces latency of off-chip accesses by smartly reusing the RPT as a line location predictor.

F. Overhead Comparison

We compare the storage overhead of SELF with state-of-the-art line-based and page-based designs, called CAMEO [11] and PoM [10], respectively. Table I shows the storage overhead of these three designs under a memory system composed of 4GB on-chip DRAM and 12GB off-chip DRAM. In such a system, there are 1 million entries in the RPT. Each entry occupies 2 bytes (one byte is allocated to a CC and the other byte is used to store page locations). Thus, the storage overhead of the RPT is 2MB. Moreover, as discussed in Section III-A, each DRAM row needs to sacrifice one data

line out of 32 data lines to implement the co-located RLT. Therefore, the total storage overhead of RPT and RLT is 4GB/32 + 2MB = 130MB. Compared to CAMEO and PoM, first, SELF requires additional storage space in the TLB, 256 bytes per core, to record page footprints. Second, SELF needs more SRAM space than CAMEO. However, SELF could have higher prediction accuracy as CAMEO only uses 512 bytes to record last accessed locations and relies on them to predict requested line locations. Third, SELF consumes more space of on-chip DRAM than PoM and CAMEO, but it is still negligible, only 3.2% of the total capacity. In summary, SELF introduces more storage overhead than CAMEO and PoM, but it achieves two conflicting goals of high hit ratio of on-chip memory and low off-chip traffic.

TABLE I: Storage Overhead Comparison

Storage	CAMEO	PoM	SELF
TLB	N/A	N/A	256B/core
SRAM	512B	32KB	32KB
on-chip DRAM	128MB (3.1%)	2MB (0.05%)	130MB (3.2%)

IV. Evaluation

A. Evaluation Methodology

We use a full system and cycle accurate simulator, MARSSx86 [23], with a detailed DRAM simulator, DRAMSim2 [24], for our evaluations. The DRAMSim2 is modified to support multiple memory instances. We use two instances of DRAMSim2 with different configurations [25] to model both on-chip DRAM and off-chip DRAM. The evaluated memory system consists of 4GB on-chip memory and 12GB off-chip memory. We use a system composed of 16GB off-chip DRAM without on-chip DRAM as our baseline system. Table II shows the system configuration in our study.

TABLE II: System Configuration

CPU	
Core	8 cores, 3.2GHz out-of-order, 4 issue width
L1-D/L1-I cache	8-way, 128KB/128KB, 2 cycles
L2 cache	8-way, private 1MB, 8 cycles
L3	16-way, shared 16MB, 24 cycles
RPT cache	4-way, 32KB, 2 cycles, LRU replacement
Die-stacked DRAM	
Bus frequency	1.6GHz (DDR 3.2GHz)
Channels/Ranks/Banks	8/1/8
Bus Width	128 bits per channel
tCAS-tRCD-tRP-tRAS	11-11-11-28
Off-chip DRAM	
Bus frequency	800MHz (DDR 1.6GHz)
Channels/Ranks/Banks	2/1/8
Bus Width	64 bits per channel
tCAS-tRCD-tRP-tRAS	11-11-11-28

We use the PARSEC 2.1 [26] benchmark suite to evaluate our design. PARSEC 2.1 includes emerging applications ranging from computer vision to financial analytics. And it is a multi-threaded and memory-sharing benchmark suite, thus it is suitable for evaluating memory system. Each benchmark of PARSEC has defined a range of interest (ROI) to represent the workload and we checkpoint each benchmark at the beginning of the ROI. We launch simulations from checkpoints with warmed caches and page footprints to achieve a steady state. Each benchmark runs for 500 million instructions with simlarge input dataset to collect statistical data.

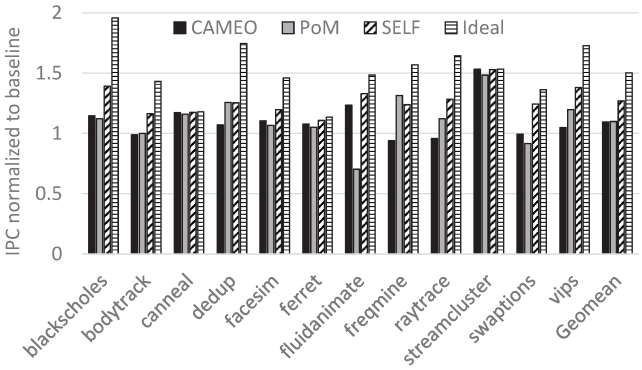
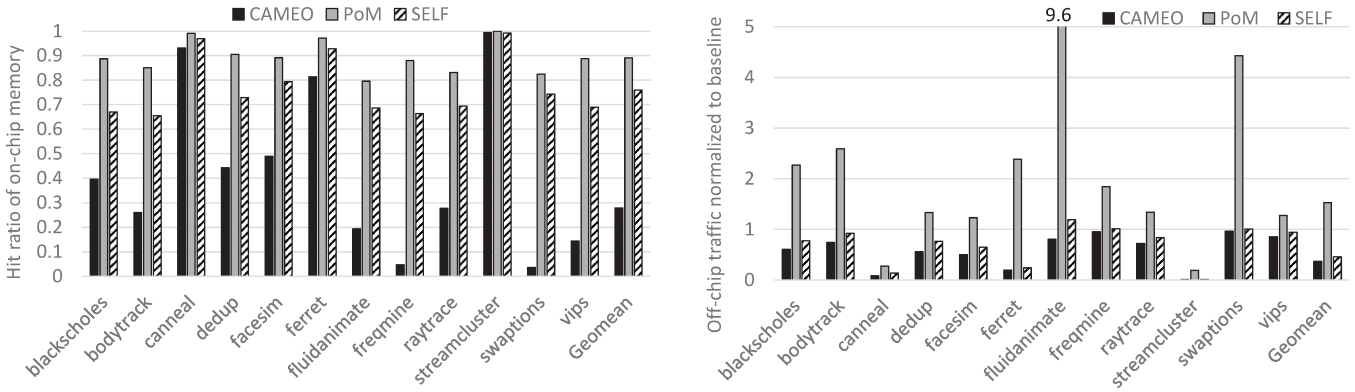


Fig. 5: Performance comparisons. On average, CAMEO and PoM improve performance by 9.5% and 9.9%, respectively, while SELF improves performance by 26.9%, which is 85% of the ideal case.

B. Performance Results

We compare SELF with CAMEO [11] and PoM [10]. We also compare these three designs against an ideal memory system composed of all on-chip DRAM without off-chip DRAM. The ideal memory system is also set to 16GB for fair comparison. We use instructions per cycle (IPC) as our performance metric. Figure 5 shows the performance results of various designs, which are normalized to the baseline system. On average, SELF improves performance by 26.9%, which is 85% of the ideal case. However, CAMEO and PoM improve performance by 9.5% and 9.9%, respectively. From the figure we can see CAMEO in some benchmarks, e.g., *freqmine* and *raytrace*, is even worse than the baseline system. There are two possible reasons for the surprising results. On the one hand, the temporal locality of these workloads is very poor, thus most requests are serviced from the off-chip memory. On the other hand, the line location predictor (LLP) used in CAMEO cannot work well with these workloads as the LLP simply uses last accessed location to predict the requested line location. The two aspects together cause the worse performance than the baseline system. PoM in some workloads, e.g., *fluidanimate*, also performs worse than the baseline system. The main reason is that these workloads have a poor spatial locality which causes PoM to exhibit a low hit ratio of on-chip memory although it swaps at a page granularity. In this case, the coarse swap granularity could easily saturate the off-chip bandwidth, leading to a long latency for off-chip accesses. However, SELF performs steadily in all benchmarks by combining all benefits from CAMEO and PoM while avoiding their shortcomings.



(a) Percentage of requests serviced from the on-chip memory.

(b) The total data read from the off-chip meory.

Fig. 6: Two import performance metrics (a) hit ratio of on-chip memory and (b) off-chip traffic. All results are normalized to the baseline. SELF achieves an average hit ratio of 76% while reducing off-chip traffic to 46% of the baseline system. Although PoM obtains the highest hit ratio, 89% on average, it also causes the highest off-chip traffic, 153% on average.

C. Hit Ratio and Off-chip Traffic

To further understand the above performance results, we collect data about two important performance metrics, hit ratio of on-chip memory and off-chip traffic, as shown in Figure 6. Figure 6a shows the percentage of requests serviced from the on-chip memory. Figure 6b shows how much data is read from the off-chip memory. We simulate write requests in the evaluation but write traffic is not calculated as write requests are not on the critical path. We compare SELF with CAMEO and PoM, all results are normalized to the baseline. For simplicity, we analyze these three designs respectively.

First, CAMEO gains the lowest hit ratio of on-chip memory among these three designs, only 28% on average. As CAMEO can only capture temporal locality due to the fine-grained line granularity used to swap data from off-chip memory to on-chip memory. Therefore, the strength of workloads' temporal locality decides the hit ratio of on-chip memory. The *freqmine* and *swaptions* have a very low hit ratio, less than 5%, thus most requests are serviced from off-chip memory. That is why their off-chip traffic is almost the same as the baseline. In general, CAMEO produces the least off-chip traffic, 37% of the baseline, as every data line read from the off-chip memory are demanded although it gains the lowest hit ratio of on-chip memory.

Second, PoM design obtains the highest hit ratio of the fast memory, 89% on average. However, it also causes the highest off-chip traffic, 153% on average. Both high hit ratio and heavy off-chip traffic are attributed to the large page granularity used to swap data between on-chip and off-chip memories. Figure 6b shows *swaptions* and *fluidanimate* workloads have a very high off-chip traffic, especially for the *fluidanimate* workload, which causes almost an order of magnitude higher off-chip traffic than the baseline system. The high off-chip traffic could easily lead to saturating the off-chip bandwidth. As a result, requests missed in the on-chip memory need to wait for a very long time in the transaction queue of memory controller, which explains why *fluidanimate* gets the worst performance with PoM design, as shown in Figure 5.

Last, our proposed SELF achieves an average hit ratio of 76%, which is close to PoM design, and meanwhile reduces the off-chip traffic to 46% of the baseline system. The high hit ratio indicates page footprint has a low rate of underprediction, i.e., a data line is demanded but it was not predicted. In contrast, the low off-chip traffic indicates page footprint has a low rate of overprediction, i.e., a data line is not demanded but it was predicted. Therefore, the page footprint is a good predictor for a page's spatial pattern. SELF takes advantage of page footprint to do partial swapping of a page to achieve both high hit ratio of on-chip memory and low off-chip traffic. Moreover, SELF reuses the RPT to predict line locations of off-chip accesses to shorten access latency. These techniques together ensures that SELF outperforms CAMEO and PoM designs on average.

D. Prediction Accuracy

In SELF, we reuse the RPT to predict the requested line location based on the location of page which the requested line belongs to. For assessing the accuracy of RPT, we first describe five possible cases that can occur: 1) the requested line is in the on-chip memory and the RPT predicted correctly; 2) the requested line is in the on-chip memory but the RPT predicted wrong; 3) the requested line is in the off-chip memory but it was predicted in the on-chip memory; 4) the requested line is resident in the off-chip memory and the RPT gave a right off-chip location; 5) the requested line is resident in the off-chip memory but the RPT gave a wrong off-chip location. Case 2, 3 and 5 are mispredicted cases, but have different misprediction penalties. In case 2, the request can still be serviced from the on-chip memory quickly, but energy and off-chip bandwidth consumed by off-chip access is wasted. In case 3, off-chip access is performed after on-chip access. Thus, the latency

TABLE III: Cost analysis under different scenarios

Served by	Prediction	Percentage	Latency	Energy	Off-chip traffic
On-chip	On-chip	75.5%	$L_{on-chip}$	$E_{on-chip}$	0
	Off-chip	4.1%	$L_{on-chip}$	$E_{on-chip} + E_{off-chip}$	64B
Off-chip	On-chip	4.8%	$L_{on-chip} + L_{off-chip}$	$E_{on-chip} + E_{off-chip}$	64B
	Off-chip (right)	10%	$L_{off-chip}$	$E_{on-chip} + E_{off-chip}$	64B
	Off-chip (wrong)	5.6%	$L_{on-chip} + L_{off-chip}$	$E_{on-chip} + 2 * E_{off-chip}$	128B

of RLT access cannot be hidden. The penalty of case 5 is the sum of case 2 and 3. Table III summaries the cost under five cases in terms of latency, energy and off-chip traffic. The $L_{on-chip}$ and $L_{off-chip}$ denote on-chip access latency and off-chip access latency, respectively. Similarly, the $E_{on-chip}$ and $E_{off-chip}$ denote energy consumed by one on-chip access and one off-chip access, respectively. In summary, the RPT achieves an average accuracy of 85.5% across all workloads.

E. Energy Analysis

Figure 7 compares various designs in terms of energy per access. The energy per access is defined as total energy consumed by on-chip and off-chip memories without considering peripheral wires divided by the number of read and write requests. The results are normalized to the baseline system. We calculate power consumption based on the Micron Power Calculator [27] and the Micron DDR3 data sheet [25]. We modify power parameters according to the power number reported in [28] for the on-chip memory. The results show that CAMEO, PoM and SELF reduce energy consumption by 31.3%, 27.6% and 47.9%, respectively. Comparing these designs, CAMEO has zero overprediction while PoM has the most overpredictions among these three designs. Each overprediction wastes energy and increases off-chip traffic. The increased off-chip traffic may prolong the execution time, resulting in more static energy consumption. Therefore, PoM consumes more energy than CAMEO on average, especially for the *fluidanimate* workload as it causes the highest off-chip traffic, as shown in Figure 6b. Running the *freqmine* workload, CAMEO even consumes more energy than the baseline system. As this workload gets a very low hit rate of on-chip memory, most requests still need to access the off-chip memory to get the requested data after first accessing the on-chip memory, which causes a lot of extra on-chip accesses, compared to the baseline system where each request only requires one off-chip access. These extra on-chip accesses cause CAMEO to consume more energy than the baseline system. However, SELF reduces energy consumption across all workloads due to its high hit ratio of on-chip memory and low off-chip traffic.

F. Sensitivity to RPT Cache Size

We adopt a RPT cache to shorten access latency of the RPT. If a request hit in the RPT cache, the requested page location can be obtained quickly. Otherwise, the request needs to access the on-chip memory instead to get a corresponding RPT entry, which causes a much longer latency. Therefore,

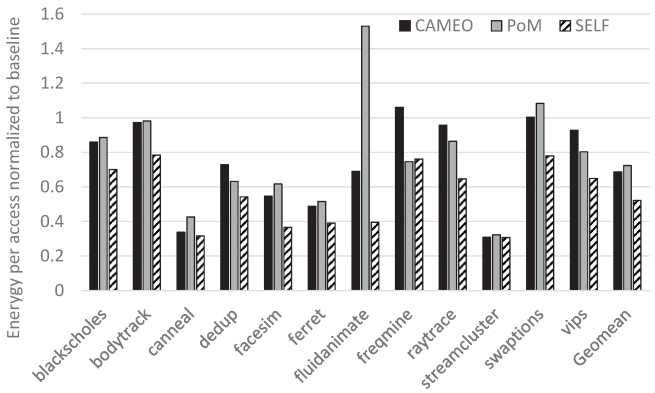


Fig. 7: Energy consumption. On average, CAMEO, PoM and SELF reduce energy per access by 31.3%, 27.6% and 47.9%, respectively.

the effectiveness of the RPT cache is crucial to the system performance. Figure 8 shows the hit ratio of the RPT cache when we change its size from 8KB to 64KB. From the figure, all workloads can get a high hit ratio even with a 8KB RPT cache. On average, the hit ratio is 77.5%, 83.5%, 85.8% and 87%, respectively. The hit ratio of the RPT cache can be improved marginally by increasing the cache size after the RPT cache reaches 32KB. Thus, we choose 32KB as the RPT cache size.

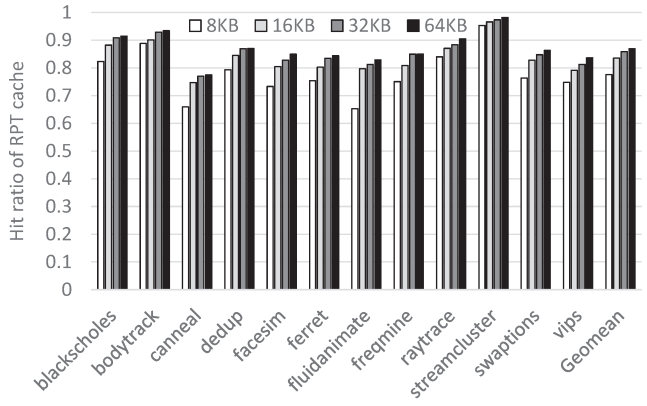


Fig. 8: Hit ratio of RPT cache across different cache size.

G. Sensitivity to Swap Threshold

In SELF system, page swapping occurs when the CC is larger than the swap threshold. Thus, the swap threshold is

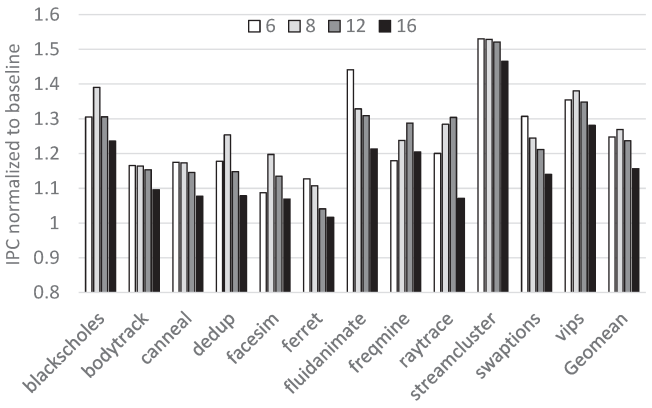


Fig. 9: Performance sensitivity to the swap threshold.

closely related to the system performance. We perform a sensitivity study of swap threshold on the system performance and Figure 9 shows the results. There is no one swap threshold that can fit all workloads. For example, *ferret* and *swaptions* prefer small swap threshold as their spatial localities are strong. The earlier requested page is swapped to the on-chip memory, the better performance SELF can achieve. In contrast, for *freqmine* and *raytrace* benchmarks, the performance is improved as the swap threshold increases. However, the performance becomes worse in most workloads when the swap threshold is set to 16. The high swap threshold could make most pages have no chance to be swapped to on-chip memory, causing most requests to be serviced from off-chip memory. In this case, SELF could be degraded to the baseline system. For example, the performance of *ferret* is close to the baseline system when the swap threshold is set 16. Therefore, the ideal value of swap threshold should be configured according to the access pattern of each workload. However, we set it to 8 as a good trade-off since SELF achieves the best performance on average in this case. To improve the adaptability of swap threshold in the future, we are trying to use several small regions in on-chip memory as sampling regions and apply different swap thresholds for them. The swap threshold which can produce the highest benefit in current interval is adopted for the next interval. In doing this, the swap threshold is changed dynamically to adapt to the access pattern.

V. Related Work

DRAM Cache. A large body of previous work [2], [3], [4], [5], [6], [7], [13], [14], [15], [29] has proposed using on-chip DRAM as a hardware-managed cache between the LLC and main memory. DRAM caches can also be classified into two categories by caching granularity: line-based and page-based. These two categories have the same problems stated in this paper. To alleviate the over-fetching problem of the page-based design, Footprint Cache [4] and Unison Cache [5] use a footprint predictor to identify and fetch only those lines within a page that will be requested during the page's residency in the DRAM cache. In doing so, they eliminate the excessive off-chip traffic associated with page-based cache designs, while preserving their high hit ratio. They are similar to our work but the on-chip DRAM is used as a cache and the tag array provides sufficient information about page footprints while this kind of information is missing in the main memory layer.

Part-of-Memory (PoM). DRAM cache has the advantage of being transparent to the OS. However, DRAM cache cannot contribute towards capacity of main memory, which could lead to non-negligible performance loss. Thus, many researchers advocate using die-stacked DRAM as a part of memory. Some hybrid approaches managed by both software and hardware have been proposed besides the hardware-managed PoM designs. Meswani et al. [16] propose the *first-touch hot-page (FTHP)* approach to managing a heterogeneous memory architecture (HMA). This approach needs support from both hardware and software. An access count is added to each TLB and page table entry to track the number of page accesses. At the end of an epoch, all pages whose access count is larger than the hotness threshold θ are treated as hot pages. The OS selects first N (N is the size of the die-stacked DRAM) hot pages to place in the stacked memory and updates corresponding PTEs. If the number of hot pages is more than N, the OS increases the hotness threshold, otherwise decreases it. In the case when the size of hot pages is less than N, the OS adopts first-touch policy to allocate requested pages in the stacked memory until it is used up. Although this approach makes use of hardware to fasten page profiling, the page table updates and TLB shoot-downs handled by the OS are still very costly, as discussed in Section II-A. Thereby, page migrations cannot happen so frequently that many opportunities to improve performance could be missed.

Oskin et al. [17] propose a software-managed and hardware-assisted approach to use die-stacked DRAM as a part of memory. This approach leverages two techniques to make it be feasible. The first is a hardware-assisted TLB shoot-down to accelerate this process; the second is a software-implemented prefetcher that extends classic hardware prefetching algorithms to the page level. This approach requires simpler hardware than our approach, however, it performs data migration between on-chip and off-chip memories at a granularity of page, resulting in waste of the off-chip memory bandwidth.

VI. Conclusion

In this paper, we propose a high performance and bandwidth efficient approach, called SELF, to exploit on-chip DRAM as a part of memory. SELF selectively swaps lines in a requested page according to its page footprint instead of swapping an entire page blindly. In doing so, SELF enables most incoming requests to be serviced in on-chip memory while avoiding swapping unnecessary lines to reduce memory bandwidth consumption. Moreover, SELF reuses the RPT to predict line location to reduce latency of off-chip accesses. As a result, SELF improves performance by 26.9% while reducing energy per access by 47.9% on average, compared to the baseline system of the same capacity.

VII. Acknowledgments

We would like to thank our shepherd, Djordje Jevdjic, and the anonymous reviewers for their insightful feedback and comments. This work is sponsored in part by U.S. National Science Foundation grants CCF-1547804, CNS-1702474 and CNS-1700719.

References

[1] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, no. 1, pp. 20–24, March 1995.

[2] G. H. Loh and M. D. Hill, "Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches," in *Proceedings of the* 44^{th} *Annual IEEE/ACM International Symposium on Microarchitecture(MICRO'11)*, 2011.

[3] M. K. Qureshi and G. H. Loh, "Fundamental Latency Trade-offs in Architecting DRAM Caches," in *Proceedings of the* 45^{th} *Annual IEEE/ACM International Symposium on Microarchitecture(MICRO'12)*, 2012.

[4] 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* 40^{th} *International Symposium on Computer Architecture(ISCA'13)*, 2013.

[5] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *Proceedings of the* 47^{th} *Annual IEEE/ACM International Symposium on Microarchitecture(MICRO'14)*, 2014.

[6] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bi-Modal DRAM Cache: Improving Hit Rate, Hit Latency and Bandwidth," in *Proceedings of the* 47^{th} *Annual IEEE/ACM International Symposium on Microarchitecture(MICRO'14)*, 2014.

[7] C.-C. Huang and V. Nagarajan, "ATCache: Reducing DRAM cache Latency via a Small SRAM Tag Cache," in *Proceedings of the* 23^{rd} *International Conference on Parallel Architectures and Compilation Techniques(PACT'14)*, 2014.

[8] Micron Technology, "Hybrid Memory Cube," https://www.micron.com/products/hybrid-memory-cube/short-reach-hmc/4GB.

[9] JEDEC STANDARD, "High Bandwidth Memory (HBM) DRAM," https://www.jedec.org/standards-documents/results/jesd235.

[10] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent Hardware Management of Stacked DRAM as Part of Memory," in *Proceedings of the* 47^{th} *Annual IEEE/ACM International Symposium on Microarchitecture(MICRO'14)*, 2014.

[11] C. Chou, A. Jaleel, and M. K. Qureshi, "CAMEO:A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache," in *Proceedings of the* 47^{th} *Annual IEEE/ACM International Symposium on Microarchitecture(MICRO'14)*, 2014.

[12] B. Akin, F. Franchetti, and J. C. Hoe, "Data Reorganization in Memory Using 3D-stacked DRAM," in *Proceedings of the* 42^{nd} *International Symposium on Computer Architecture(ISCA'15)*, 2015.

[13] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms," in *Proceedings of the* 16^{th} *IEEE International Symposium on High Performance Computer Architecture(HPCA'10)*, 2010.

[14] S. Franey and M. Lipasti, "Tag Table," in *Proceedings of the* 21^{st} *IEEE International Symposium on High Performance Computer Architecture(HPCA'15)*, 2015.

[15] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A Fully Associative, Tagless DRAM Cache," in *Proceedings of the* 42^{nd} *International Symposium on Computer Architecture(ISCA'15)*, 2015.

[16] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous Memory Architectures: A HW/SW Approach for Mixing Die-stacked and Off-package Memories," in *Proceedings of the* 21^{st} *IEEE International Symposium on High Performance Computer Architecture(HPCA'15)*, 2015.

[17] M. Oskin and G. H. Loh, "A Software-managed Approach to Die-stacked DRAM," in *Proceedings of the* 24^{th} *International Conference on Parallel Architectures and Compilation Techniques(PACT'15)*, 2015.

[18] G. H. Loh, N. Jayasena, J. Chung, S. K. Reinhardt, J. M. OConnor, and K. McGrath, "Challenges in Heterogeneous Die-Stacked and Off-Chip Memory Systems," in *Proceedings of* 3^{rd} *Workshop on SoCs, Heterogeneous Architectures and Workloads (SHAW'12)*, 2012.

[19] K. Chen, S. Li, J. H. Ahn, N. Muralimanohar, J. Zhao, C. Xu, S. O, Y. Xie, J. B. Brockman, and N. P. Jouppi, "History-Assisted Adaptive-Granularity Caches (HAAG$) for High Performance 3D DRAM Architectures," in *Proceedings of the* 29^{th} *ACM International Conference on Supercomputing(ICS'15)*, 2015.

[20] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial Memory Streaming," in *Proceedings of the* 33^{rd} *International Symposium on Computer Architecture(ISCA'06)*, 2006.

[21] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos, "Accurate and Complexity-Effective Spatial Pattern Prediction," in *Proceedings of the* 10^{th} *IEEE International Symposium on High Performance Computer Architecture(HPCA'04)*, 2004.

[22] S. Kumar and C. Wilkerson, "Exploiting Spatial Locality in Data Caches using Spatial Footprints," in *Proceedings of the* 25^{th} *International Symposium on Computer Architecture(ISCA'98)*, 1998.

[23] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSSx86: A Full System Simulator for x86 CPUs," in *Proceedings of the* 48^{th} *ACM/EDAC/IEEE Design Automation Conference(DAC'11)*, 2011.

[24] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, January 2011.

[25] Micron Technology, "8Gb: x4, x8 1.5V TwinDie DDR3 SDRAM," 2011.

[26] C. Bienia and K. Li, "PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors," in *Proceedings of the* 5^{th} *Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.

[27] Micron Technology, "Calculating Memory System Power for DDR3," Tech. Rep. TN-41-01, 2007.

[28] B. Giridhar, M. Cieslak, D. Dugga, R. Dreslinski, H. M. Chen, R. Patti, B. Hold, C. Chakrabarti, T. Mudge, and D. Blaauw, "Exploring DRAM Organizations for Energy-Efficient and Resilient Exascale Memories," in *Proceedings of the IEEE International Conference for High Performance Computing, Networking, Storage and Analysis(SC'13)*, 2013.

[29] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support," in *Proceedings of the IEEE International Conference for High Performance Computing, Networking, Storage and Analysis(SC'10)*, 2010.