i-MIRROR: A Software Managed Die-Stacked DRAM-Based Memory Subsystem

Jee Ho Ryoo[†], Karthik Ganesan[⋆], Yao-Min Chen[⋆] and Lizy K. John[†]
[†]*The University of Texas at Austin* [⋆]*Oracle Corporation*
jr45842@utexas.edu, {karthik.ganesan, yaomin.chen}@oracle.com, ljohn@ece.utexas.edu

Abstract

This paper presents an operating system managed die-stacked DRAM called i-MIRROR that mirrors high locality pages from the off-chip DRAM. Optimizing the problems of reducing cache tag area, reducing transfer bandwidth and improving hit latency altogether while using the die-stacked DRAM as hardware cache is extremely challenging. In this paper, we show that performance and energy efficiency can be obtained by software management of the die-stacked DRAM, which eliminates the need for tags, the source of aforementioned problems.

In the proposed scheme, the operating system loads pages from disks to the die-stacked DRAM on a page fault at the same time as they are loaded to the off-chip DRAM. Our scheme maintains the pages in the off-chip and the die-stacked DRAM in a synchronized/mirrored state by exploiting the parallel loading capability to the die-stacked and off-chip DRAM from the disk. This eliminates the need for physical page movement to the slower off-chip DRAM upon eviction from the die-stacked DRAM. Requests for pages that got evicted from the die-stacked DRAM are simply serviced by the slower off-chip DRAM to prevent frequent data movements of large pages and thrashing between conflicting pages. The operating system periodically monitors the usage of the pages in the off-chip DRAM and promotes high locality pages to the die-stacked DRAM. Our evaluations show that the proposed hardware-assisted software-managed i-MIRROR scheme achieves an IPC improvement of 13% while consuming 6% less energy than prior state-of-the-art die-stacked caching schemes and 79% improvement in terms of IPC and 72% in terms of energy savings over systems without die-stacked DRAM support.

1. Introduction

Multicore processors have enjoyed performance benefits from exploiting the high level of parallelism, yet having more cores places more pressure on the off-chip memory bus as the bandwidth demand of the system now is much higher with concurrently running applications. The story gets worse with DRAM latency which has improved relatively little over a decade [1]. Die-stacked DRAM (DSD) rises as an emerging solution to address the problems in today's off-chip memory bandwidth. With a large number of banks, this technology can deliver much higher bandwidth than the off-chip DRAM. Since the DSD sits on the same silicon interposer as the cores, the physical proximity to the cores lowers the latency [2]. Therefore, this technology can offer higher bandwidth and lower latency than the off-chip DRAM [3]. One popular approach to leverage the DSD is to use it as a level of storage device between Last Level Cache (LLC) and the off-chip DRAM. But the DSD is still fundamentally DRAM, and thus, it presents numerous challenges in terms of design constraints-Many prior hardware solutions use the DSD as another level of cache [4–12] attempting to achieve the optimal tag management schemes while keeping the latency low. The software solution uses the Operating System (OS) and explicitly remaps pages between two memories [13–15].

Although prior hardware managed caches such as the Alloy Cache [6] and the Footprint Cache [7] devised techniques to reduce the cache tag area, transfer bandwidth and hit latency, they still have to access tags, which involve complex hardware units and prediction mechanisms. In this paper, we propose an OS approach with a minimal amount of architectural support to achieve a tagless read access. The OS initiates filling the DSD by directly controlling the data movement between the disk and two DRAM memories during page fault. Unlike hardware approaches, our approach treats the DSD present at the same level of memory hierarchy as the off-chip DRAM while acting as a mirror of selected regions of the off-chip DRAM. This technique, which we name **i-MIRROR**, ensures that evicted pages are only serviced from the off-chip DRAM and are not moved into the DSD until later by an OS service routine. A small hardware unit, Reverse Promotion Table (**RPT**), constantly monitors the usage statistics of the highly used off-chip DRAM pages. When the OS interrupts periodically, it then promotes highly used pages to the DSD. Unlike hardware schemes that move data on every cache miss, i-MIRROR only performs such operation during the OS intervention, namely a page fault/service interrupt. Similarly, the i-MIRROR Address Translator (**iMAT**) collects statistics of least recently used pages in the DSD as well as keeps track of mirrored pages in the DSD and off-chip DRAM. The two structures help to minimize the detrimental effects of costly page remapping and allow our scheme to overcome the challenges typically associated with software schemes. In summary, our work makes the following contributions:

1. We propose a software-managed DSD solution where the OS is responsible for managing pages from the disk and the off-chip DRAM to the DSD with the assistance of hardware structures that provide fine granularity monitoring statistics. Our scheme provides a tagless access to the DSD solving many problems in prior hardware DSD caching schemes.
2. We present an innovative mechanism to evict pages from

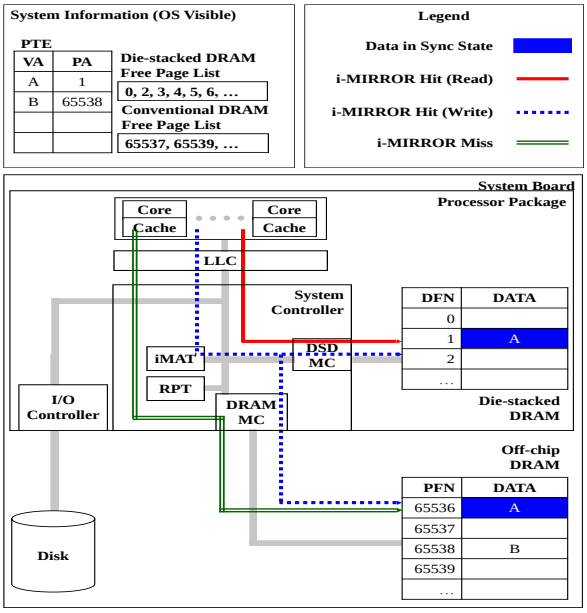

Figure 1: Hit and Miss Path

the DSD to the off-chip DRAM without physically moving pages. By maintaining the DSD in a synchronized state, the i-MIRROR accomplishes **data-movement-free page eviction**.

3. We present an architectural support mechanism, RPT, to assist the OS to perform **periodic selective page promotion** efficiently. This avoids the large page movement at every DSD miss typical of hardware caching schemes. The architectural support from the RPT can enable the OS to identify hot pages and to perform page promotion opportunistically.

We begin with the architectural details of the proposed scheme in Section 2. Our experimental setup and evaluation are presented in Section 3 and Section 4 respectively. Section 5 presents related work done in the field. Finally, we conclude the paper in Section 6.

2. The i-MIRROR Scheme

Figure 1 shows a system overview of our proposed i-MIRROR scheme. DSD and DRAM have its own Memory Controller (DSD MC and DRAM MC). The i-MIRROR Address Translator (iMAT) and Reverse Promotion Table (RPT) are auxiliary structures to guide our scheme. The DSD and off-chip DRAM are part of the same physical address space. In this work, we assign the low physical address space to the DSD, so after a certain page frame number (PFN), the address automatically gets assigned to the off-chip DRAM space. We call page frame numbers assigned to the DSD Die-stacked DRAM Frame Number (DFN). We assume that PFN and DFN are exclusive to each other, and separate free lists are kept for PFN and DFN.

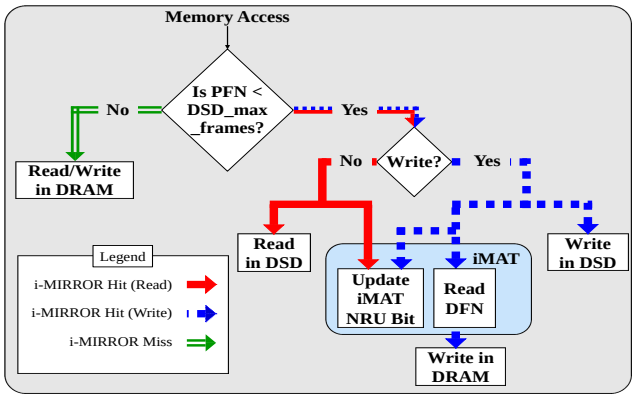

Figure 2: i-Mirror Flow Chart

2.1. Operation of the i-MIRROR

In a read hit scenario for Data A, the Physical Address (PA) read from the Page Table Entry (PTE) is 1, so the request is directly forwarded to the DSD. The request is serviced solely by the DSD with the ideal hit latency. This path is shown by the solid line in Figure 1. In a write hit scenario for Data A, a request reaches the system controller. Since the PTE contains a DFN, the write request is sent to the DSD. However, A's updated data should be synced with the data in the off-chip DRAM in order to keep the mirrored state. Therefore, the corresponding PFN for Data A needs to be found. The iMAT structure keeps track of the DFN and PFN pairing information. In our case, the DFN 1 is a mirrored copy of the page with the PFN 65,536. The number of iMAT entries is equal to the number of page entries in the DSD, so the iMAT is directly indexed with the DFN. For writing Data A to the PFN 65,536, a separate request is generated from the system controller, and is sent to the off-chip DRAM as denoted by a dotted line. This additional request does not have any effect on the correct execution of the program as the processor is oblivious to the the existence of the mirrored page in the off-chip DRAM (only the OS is aware). This redundant request is generated in order to maintain the sync state of the DFN 1 and PFN 65,536. We assume that modern processors have sufficient memory controller write buffers, so that this does not stall the execution.

Since the DSD capacity is limited, not all pages can be resident in the DSD. Pages not in sync state are only resident in the off-chip DRAM. Consequently, their PA entry in the PTE contains a PFN instead of a DFN. Data B in Figure 1 is an example of such a page, which is not resident in DSD. Any request regardless of a read or a write for B performs the identical operation as a system without DSD. Since the PA entry already returns the PFN, the system knows that the data is only resident in the off-chip DRAM. This bypasses the iMAT and achieves the ideal miss latency. We call this path the miss path since the data is serviced solely by the off-chip DRAM. Note that no operation to retrieve the DSD residency information such as the tag lookup is performed

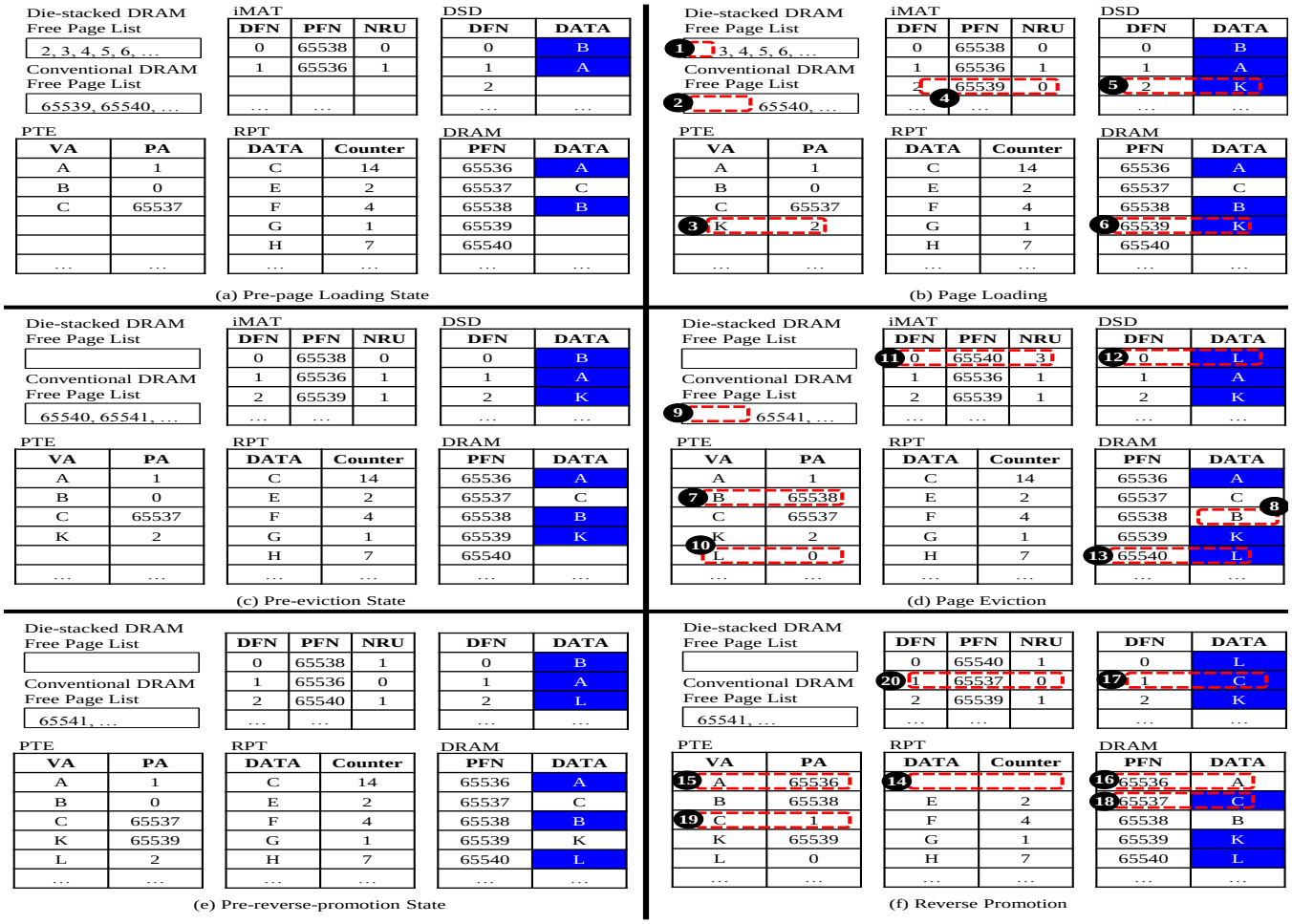

Figure 3: Various i-Mirror Operations

here. However, for such data, which does not reside in the DSD, the access requests are also sent to an RPT structure as in Figure 1. It keeps track of heavily used pages that are not in sync state, which later helps the OS to identify hot pages that are not resident in the DSD. Note that this is not on the critical path, and thus, does not affect performance. A flow chart summary of those three scenarios is presented in Figure 2.

2.2. i-MIRROR During Page Fault

We have so far discussed the operation of the i-MIRROR scheme in relation to normal program execution. The data movement between the two DRAM systems was not discussed as no data is moved without OS intervention in our scheme. Here, we will explain three scenarios where data is moved between the two DRAM devices.

Our system has data in sync state, which requires data to be resident in both the DSD and the off-chip DRAM. This begins during a page fault. Figure 3(a) shows the initial system state. At this time, a request is made for Data K, which invokes the OS to fill a page. Figure 3(b) explains the detailed process. At first in ❶, the OS finds a free page from the DSD free page list. In our example, the DFN 2 is chosen. Since we need a mirrored copy, we also need a PFN from the DRAM free page list. The system selects 65,539 as in ❷. Now, the OS populates the PTE with the selected DFN. In ❸, the corresponding PA for Data K is 2. At the same time, since the mirrored copy for Data K will be resident in the PFN 65,539, we populate an iMAT entry as well in ❹. The iMAT can have dedicated memory addresses just like conventional I/O devices, so that the OS can directly write/retrieve required information. Now, all tables are populated, so the OS fetches Data K into both the DSD and the off-chip DRAM as in ❺ and ❻. Since we have a separate MC for the DSD and the off-chip DRAM, operations ❺ and ❻ are done simultaneously. The DSD page loading finishes earlier than or at the same time as the off-chip DRAM loading due to DSD's lower latency. Hence, this operation has no detrimental effects on overall latency as a system without DSD will incur a page fault latency regardless of our implementation. The page fault operation terminates once the transfer is over. Now, Data K is in sync state as denoted by K. The normal execution resumes as discussed in Section 2.1.

Soon, the system will have the DSD filled with new pages and the DSD free page list will become empty as shown in

Figure 3(c). Here, we will explain what happens when another page fault occurs for Data L as shown in Figure 3(d). We find the eviction candidate (Data B in our example) in the iMAT. Finding an eviction candidate is the same process as the modern OS where the system looks for unreferenced pages. Since the iMAT contains the PFN for Data K, we update the PTE with the PFN 65,538 as in ❼. Although Data B is still resident in the DSD, it lost its only reference, which is the DFN in the iMAT. The only reference corresponding to Data B is PFN 65,538, which is written in the PTE. From this point, all requests for Data K will be serviced by the off-chip DRAM as in ❽. Although our system has not moved any data physically, the operation had the same effects as if Data K moved from the DSD to the off-chip DRAM. In other word, we virtually moved a page from the DSD to the off-chip DRAM without incurring any latency or bandwidth penalty. The reason why data in sync state is enforced in our scheme is to allow this novel eviction technique. At this point, the OS has not finished the service request yet as it still needs to allocate a page for Data L. It is a newly allocated page, so we need another set of the DFN and PFN. Since we just freed one DSD page, the system will reuse the DFN 0. The OS uses another free PFN in ❾. Now, a pair of the DFN and PFN is ready, so the OS populates the PTE with the DFN as shown in ❿. The iMAT is also updated such that the mirrored page of the DFN 0 is now the PFN 65,540 (⓫). Just like in Figure 3(d), we finally move Data L from the disk to both the DSD and the off-chip DRAM as shown by ⓬ and ⓭ simultaneously.

2.3. i-MIRROR During OS Interrupts

The OS interrupts the execution periodically for various reasons such as thread scheduling and memory coalescing [16]. We take advantage of this fact and perform an operation called a reverse promotion during the interrupt period. The motivation behind this technique is that as a program enters different execution phases, what used to be cold pages can become hot pages in a new phase. In this case, we need a mechanism to bring those pages back into DSD as well as to maintain them again data in sync state. The reverse promotion is a technique that achieves this. Unlike other hardware schemes where data movement from the off-chip DRAM to DSD occurs on every DSD miss, our reverse promotion occurs independent of a hit or a miss event.

Figure 3(e) shows the system state in the beginning of the OS interrupt. In Figure 3(f), the OS first looks through the RPT entries as the RPT tracks the access counts of heavily used pages that are not in sync state. The RPT incurs small overheads since it only tracks a small number of pages as our goal is to identify only a few hot pages that are not resident in the DSD. The promotion candidate is chosen based on access count. Similar to the iMAT, the RPT is also memory mapped and can be directly accessed by the OS. In our example, Data C has the highest counter value, so it is chosen as a reverse promotion candidate. First, the OS removes the entry from the RPT as in ⓮. Since our DSD free page list is empty, our system needs to evict a page from DSD. Using the same algorithm as used in Section 2.2, Data A is chosen as an eviction candidate. The system updates the PTE for Data A with the PFN 65,536 as shown in ⓯. ⓰ shows that Data A is not in sync state anymore. Now, we physically copy Data C from the off-chip DRAM to the DSD. Once this process completes, Data C is in sync state as shown by ⓱⓲. At the same time, the OS needs to update the corresponding iMAT and PTE entry for Data C, so the operation is performed by ⓳ and ⓴. Now, the interrupt is over and the execution continues.

Processor	Values
Number of Cores	8
Core Frequency	4.0 GHz
Width	1 IPC
Cache (Block Size: 64B)	**Values**
L1 I-Cache (private)	32KB, 4 way, 2 cycles
L1 D-Cache (private)	32KB, 4 way, 2 cycles
L2 Cache (unified)	4MB, 16 way, 10 cycles
DRAM	**Values**
Bus Frequency	800 MHz (DDR 1.6 GHz)
Channels	2 (64 bits per channel)
Banks	8 Banks per Channel
Maximum Bandwidth / Channel	12.8GB/s
Row Buffer Size	4KB
tCAS-tRCD-tRP-tRAS	12-12-12-45
Die-stacked DRAM	**Values**
Bus Frequency	1600 MHz (DDR 3.2 GHz)
Channels	4 (128 bits per channel)
Banks	16 Banks per Channel
Row Buffer Size	4KB
tCAS-tRCD-tRP-tRAS	8-8-8-11

Table 1: Experimental Parameters

3. Experimental Setup

3.1. Workloads

In this paper, we have used a subset of workloads from SPEC CPU2006 benchmark suite [17] to evaluate the i-MIRROR scheme. Our DSD and off-chip DRAM capacity is 256MB and 8GB respectively. We carefully chose the benchmarks that experience high off-chip bandwidth usage. We run these benchmarks where 8 separate instances of the SPEC CPU2006 workloads execute on each core. We ran the simulation for the total of 1 billion instructions per core.

3.2. Simulation

We use the Flexus architectural timing simulation infrastructure [18], which uses Virtutech SIMICS [19]. It supports the SPARC ISA, and our benchmarks are compiled using compiler suites available in Oracle Solaris Studio 11 [20]. Solaris 10 [21] is the underlying OS platform. Table 1 shows the simulation parameters to model the our simulation [22]. The simulator contains the detailed DRAM timing module based on DRAMSim2 [23]. We modeled the 256MB DSD module

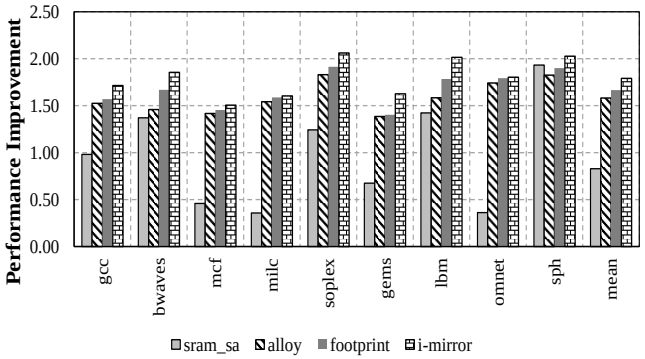

Figure 4: Performance Improvement over no DSD Cache

Figure 5: Hit Rates of Various DSD Caching Schemes

based on publicly available data [24] [25]. Overheads are calculated using CACTI 5.3 [26] and 32nm technology.

4. Evaluation

In the rest of this paper, we compare the proposed i-MIRROR scheme against two state-of-the-art die-stacked DRAM caching schemes, the Alloy Cache [6] and the Footprint Cache [7]. We faithfully modeled these state-of-art schemes to the best of our knowledge. A generic 4-way set-associative with 4KB page size is also presented as sram_sa.

4.1. Performance Evaluation

Figure 4 shows the performance improvement of various DSD schemes over the baseline system without DSD. The performance metric is the IPC speedup over the baseline. The sram_sa scheme suffers in performance as it naively moves a large page. This scheme suffers more for workloads that have lower hit rate such as milc, which is shown in Figure 5. As noticed, sram_sa requires a very high hit rate (close to 1) in order to see performance improvement. Otherwise, it suffers from high miss latency and off-chip bandwidth usage. Figure 6 shows the relative off-chip bandwidth usage of various schemes, and as expected, the sram_sa scheme stands out.

The Alloy Cache performs well overall across all benchmarks. Since a small block does not take advantage of spatial locality of a large page, the hit rate is relatively lower. However, a small block lowers the bandwidth usage as shown in Figure 6. The hit/miss access latency is also low as it fetches/evicts only the 64B worth of data, so the scheme performs well overall by achieving 58% performance improvement over the baseline. We implemented the MAPI scheme presented in [6], the prediction mechanism where a request is sent to either the off-chip DRAM or the DSD only until the tag check result is available. This speculation unit achieves 93% accurancy on average across our benchmarks. Since the Alloy Cache needs to bring less amount of data on a miss, it has the lower miss latency.

The Footprint Cache is a subblocking scheme that takes advantage of a page based design. It achieves a high hit rate just like the page based scheme. Yet, moving only selected subblocks (64B blocks) from the off-chip DRAM to the DSD

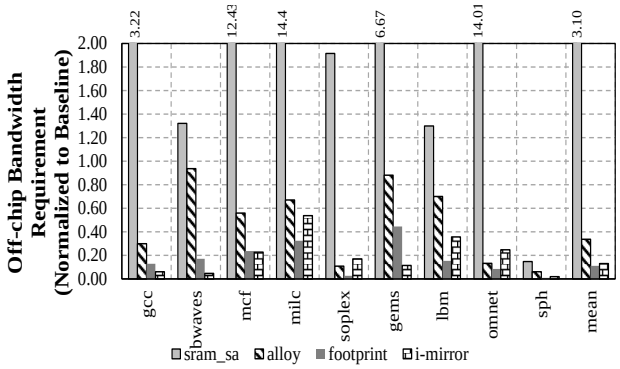

Figure 6: Off-chip Bandwidth Requirement

significantly reduces the overall bandwidth usage unlike other page based schemes. Figure 6 shows that this scheme saves more than 80% of the baseline bandwidth. The Footprint Cache effectively enjoys the high hit rate of a page based design while significantly reducing the off-chip bandwidth usage. This is reflected in Figure 4 as this caching scheme achieves 67% higher performance than the baseline.

Unlike other page based designs such as sram_sa and the Footprint Cache, the i-MIRROR does not always achieve a high hit rate in all benchmarks. A high hit rate in a page based design is achieved by aggressively fetching a large amount of data into the DSD. In hardware based schemes, this movement is triggered by a DSD miss. Yet, in our case, the triggering point is an OS interrupt, so the frequency of this movement is much lower. Not fetching a large page on every miss reduces the hit rate in some benchmarks. Yet, the design choice of not fetching a page on a miss significantly reduces the off-chip bandwidth as shown in Figure 6. The i-MIRROR completely eliminates the SRAM tag lookups on the critical path of a DSD hit or miss, so our scheme benefits significantly from achieving the ideal access latency, compensating for the lower hit rate. For those benchmarks where the i-MIRROR achieves a high hit rate, the improvement is significant as the i-MIRROR provides the ideal hit latency. On the other end where the i-MIRROR hit rate is low, our scheme still performs relatively well by restricting the data movement at the OS interval. This lowers the huge latency/bandwidth overheads associated with moving a large page. Although our miss rate is higher in this

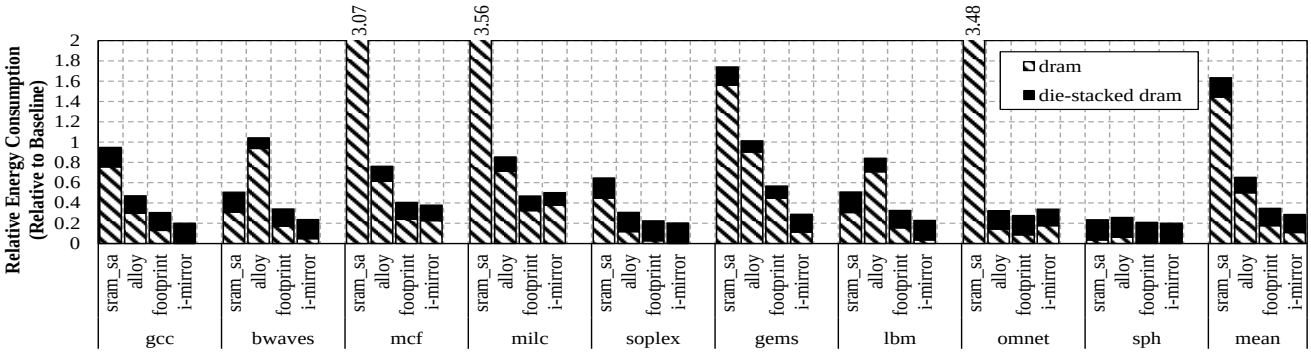

Figure 7: Memory System Energy Breakdown

case, our miss latency is the same as the ideal case, so the miss access latency is not any longer than the baseline case without the DSD. In other words, hits still provide the performance improvement without paying any latency penalty for misses. In other schemes, the miss always deteriorates the performance due to tag checks or a large amount of data transferred. The i-MIRROR improves the performance by 79% over the baseline. Out scheme performs more than 21% and 13% better on average than the Alloy and Footprint Cache respectively.

Until now, we have focused on the performance, bandwidth and latency aspect of the i-MIRROR. However, the DSD also offers much lower energy per access in comparison to the off-chip DRAM [24]. Here, we analyze the total memory system energy consumption and its breakdown in Figure 7. Here, we only show the dynamic energy breakdown. The off-chip DRAM energy consumption in the sram_sa case is dominated by the large amount of data that needs to be fetched on a miss. In some cases such as mcf, the energy consumption is almost 3.07 times higher than the baseline. Even worse, the off-chip DRAM component itself is higher than the baseline without the DSD. In the Alloy case, the higher miss rate makes the system to spend 40% of the baseline energy on the off-chip DRAM accesses. In a page based scheme such as the Footprint Cache, a group of consecutive blocks are read out from the same off-chip DRAM row, so no activation of a new row is necessary. However, this is not necessarily true in the Alloy Cache since consecutive accesses can open two different rows or conflict each other in terms of row buffer hits. Also, the MAPI generates some misspeculated requests to the off-chip DRAM. The Footprint Cache saves energy as it only fetches a small subset of a page, thereby significantly more energy efficient than the sram_sa scheme. On average, it consumes 80% less off-chip DRAM energy than the baseline. Accesses to the energy efficient DSD adds only 18% more energy, so overall this scheme saves more than 60% energy compared to the baseline. However, in some benchmarks that show high spatial locality within a page and a large hot working set such as bwaves, the Footprint Cache energy consumption does not benefit significantly from subblocking. In this case, this scheme has to fetch a large amount of data, or many subblocks, just like the sram_sa case. The i-MIRROR has

	sram_sa	alloy	footprint	i-MIRROR
SRAM Tag	320 KB	–	1368 KB	416 KB
DSD Tag	–	19 MB	–	–
Miss-Predictor	–	768 B	–	–
Footprint History Table	–	–	144 KB	–
Singleton Table	–	–	3 KB	–
RPT	–	–	–	832 B

Table 2: Overheads of Different Schemes (256MB of DSD)

significant energy advantages as it does not move data from the off-chip DRAM to the DSD frequently as in other schemes. In high spatial locality workloads, the i-MIRROR scheme has a very few accesses to the off-chip DRAM. In low spatial locality workloads, the i-MIRROR reduces the number of large data fetching from the off-chip DRAM to the DSD by only performing such operation at the OS interrupt times. Therefore, the DRAM energy consumption alone is saved by more than 90% compared to the baseline. With added energy from the DSD, the i-MIRROR scheme consumes 29% of the baseline energy consumption.

4.2. Hardware Overheads

All schemes we have used throughout the paper use the SRAM/DSD to store bookkeeping information. Table 2 summarizes overheads associated with each scheme. The sram_sa scheme is the simplest with only requiring 320KB of SRAM to store tags. The Alloy Cache uses 19MB of the DSD space in order to store the tag. In addition, additional SRAM storage is used for the predictor. The Footprint Cache uses a larger SRAM overheads as it contains bit vectors required for subblocking, so it uses the largest amount of SRAM among schemes used in this paper. The i-MIRROR's SRAM Tag is the iMAT overheads. It is relatively small, but little larger than the sram_sa since our scheme stores more tag and replacement bits (5 more bits). Scheme specific SRAM overheads such as Singleton Table and RPT use negligible area and do not grow with the DSD capacity. Overall, the i-MIRROR provides performance benefits with smaller overheads than those recently proposed Alloy and Footprint Cache

4.3. Software Overheads

The i-MIRROR requires the OS support and the OS context switch overheads are not negligible. Our system requires the OS in two scenarios. First, when a page gets initially loaded into the DSD, the OS updates the corresponding iMAT and PTE. However, updating the PTE is a required operation during a page fault regardless of our scheme. We are updating one more PTE than what the current OS performs. Since this operation only occurs when the OS is already in the system due to a page fault, we do not pay any extra performance to get the OS into the system. An additional overhead is to modify the iMAT. The latency is much lower as it is an SRAM operation, and also, negligible in comparison to other page fault activities such as accessing the off-chip DRAM to read the PTE. The page transfer from the disk to the DSD occurs at the same time as to the off-chip DRAM. It does not use any extra disk bandwidth as the data is only duplicated at the system controller.

Second, in order to perform the reverse promotion, the OS has to come into the system periodically. The current OS already performs some background jobs periodically [27]. The memory page coalescing is an ideal OS service routine where we can piggyback our reverse promotion scheme to [16, 28]. The memory coalescing service routine walks down the page tables to find the coalescing candidates. Similarly in our scheme, the OS looks through the iMAT and RPT to find the replacement candidate. The memory coalescing service routine moves multiple small size (∼4KB) pages and coalesce them into a super page (∼1MB). Likewise, the reverse promotion performs similar operations as it moves several pages and updates a PTE. Hence, our scheme can be piggybacked to this existing OS service, which already performs similar operations. In both scenarios, the i-MIRROR adds a few inexpensive operations such as updating an iMAT, yet no additional OS context switch overheads are added by appending our proposal to existing OS service routines.

4.4. TLB Shootdown Overheads

The i-MIRROR mechanism involves many PTE updates. Whenever there is a PTE update, the corresponding entry in TLB has to be invalidated. A TLB is private to each core in modern systems, so in multicore platforms, the corresponding entries in TLBs have to be invalidated. During this period, the TLB will be unavailable, the execution is stalled, and this phenomenom is called the TLB shootdown [29]. In modern OS, the TLB shootdown only stalls threads that issue memory requests to the page whose PTE is being updated. This is done by mutexing the PTE being updated and performing the cross-calls to all TLBs for invalidation [28, 30]. Although the likelihood of multiple threads requesting the same page during the PTE update is rare, we have modeled the TLB shootdown faithfully, so whenever there is any memory access to the page whose PTE is being updated, we have stalled the execution fully until the PTE update is done.

5. Related Work

A plenthora of work in hardware DSD caching schemes has been done. CHOP is a large page caching scheme with reasonable SRAM overheads that filters pages from being inserted into the DSD based on access count [4]. MissMap is a proposal that quickly identifies misses by keeping the DSD resident bit vectors [5]. Sim, J. et al. use address based speculation units to eliminate the MissMap's SRAM overheads [8]. The Alloy Cache solves the large tag area overheads associated with small block caches by placing the tag next to the data in the DSD [6]. The Footprint Cache solves the high bandwidth usage associated with page based caches by subblocking [7]. As the DSD capacity increases to multi-gigabytes over years, the SRAM tags face scaling problem. The Unison Cache solves this issue by placing the tag and other bookkeeping information used in the Footprint Cache in the DSD [9]. Sim, J. et al. use a small SRAM cache in the memory controller that temporarily stores the large pages that are being swapped between two DRAM devices, so it does not stall any execution [11]. The Bimodal Cache uses a different approach in that rather than having one fixed cacheline size, the DSD can have a few cacheline sizes, so the capacity utilization and the hit rate can both improve [10]. CAMEO manages a group of blocks together and swaps blocks only within the group. This scheme uses the DSDE to store the block location information within the group [12]. Tag tables scheme manage tags in hierarchical manner as in multilevel page tables and prefetches them to save the tag lookup latency [31].

There also has been a large amount of work done in using software to manage the DSD. Dong, X. et al. propose that the OS constantly remaps multi-megabyte pages between the DSD and the off-chip DRAM, thereby achieving a fully software approach [14]. Loh, G. et al appended additional bits to the TLB in order to better identify hot pages for the OS, such that the OS can remap pages with higher accuracy [13]. Meswani, M. et al. performed an analysis on different page replacement policies to the DSD that can be used by the OS and proposed a history based policy that has minimal hardware changes [15].

6. Conclusion

The die-stacked DRAM is a promising next generation memory technology as an intermediate memory hierarchy layer in modern many core systems, but requires a very careful design and a lot of empirical analysis to get past different challenges including the SRAM tag storage overheads, the excessive off-chip bandwidth usage and the high hit/miss access latency. In this paper, we have proposed the i-MIRROR, a novel hardware assisted software managed DSD that is used as a part of the OS managed physical address space. Our solution enables an access to the DSD without accessing any tags, either in SRAM or DSD by simultaneously loading to

the DSD and the off-chip DRAM at page fault to create initial data in sync state. It maintains the sync state with all pages in the DSD, eliminating the need for costly writing back the pages at eviction time. Our scheme utilizes hardware aids in order to identify high locality pages and to capture the hot pages in the DSD. Thus, it combines the best of the hardware caching and software-only methods in an elegant manner.

The i-MIRROR proposal achieves higher performance than hardware-only mechanisms while providing lower on-chip area overheads, memory bus bandwidth utilization, and energy. On average, the i-MIRROR with a 256MB DSD achieves performance improvement of 13% while consuming 6% less average energy than recent DSD caching schemes. We compare the performance of the i-MIRROR to those of the state-of-the-art techniques such as the Alloy Cache and the Footprint Cache. We show that the i-MIRROR achieves the best improvement in terms of performance and energy while using the least extra hardware.

7. Acknowledgement

The researchers are supported partially by SRC under Task ID 2504, National Science Foundation under grant numbers 1337393 and 1117895, Oracle, AMD and Huawei. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or other sponsors.

References

[1] ITRX, 2011. [Online]. Available: http://www.itrs.net/Links/2011ITRS/Home2011.htm.

[2] G. Loh, "3d-stacked memory architectures for multi-core processors," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, June 2008, pp. 453–464.

[3] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *2012 Symposium on VLSI Technology (VLSIT)*, 2012, pp. 87–88.

[4] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, D. Solihin, and R. Balasubramonian, "CHOP: Adaptive filter-based DRAM caching for CMP server platforms," in *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA),*, 2010, pp. 1–12.

[5] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked DRAM caches," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 454–464.

[6] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting DRAM caches: Outperforming impractical SRAM-Tags with a simple and practical design," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 235–246.

[7] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM caches for servers: hit ratio, latency, or bandwidth? Have it all with footprint cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013, pp. 404–415.

[8] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi, "A mostly-clean DRAM cache for effective hit speculation and self-balancing dispatch," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 247–257.

[9] D. Jevdjic, G. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, Dec 2014, pp. 25–37.

[10] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bi-modal dram cache: Improving hit rate, hit latency and bandwidth," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, Dec 2014, pp. 38–50.

[11] J. Sim, A. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent hardware management of stacked dram as part of memory," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, Dec 2014, pp. 13–24.

[12] C. Chou, A. Jaleel, and M. Qureshi, "Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, Dec 2014, pp. 1–12.

[13] G. H. Loh, N. Jayasena, K. McGrath, M. O'Connor, S. Reinhardt, and J. Chung, "Challenges in heterogeneous die-stacked and off-chip memory systems," in *the 3rd Workshop on SoCs, Heterogeneous Architectures and Workloads (SHAW) held in conjunction with HPCA-18*, 2012.

[14] 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 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[15] 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 *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, Feb 2015, pp. 126–136.

[16] J. W. Davidson and S. Jinturkar, "Memory access coalescing: A technique for eliminating redundant memory accesses," in *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, ser. PLDI '94. ACM, 1994, pp. 186–195.

[17] Standard Performance Evaluation Corporation, "SPEC CPU2006," 2006. [Online]. Available: http://www.spec.org/cpu2006.

[18] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, "Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 4, pp. 31–34, Mar. 2004.

[19] Wind River Systems, "Wind River Simics." [Online]. Available: http://www.windriver.com/products/simics/.

[20] Oracle Corporation, "Oracle Solaris Studio," 2012. [Online]. Available: http://www.oracle.com.

[21] ——, "Solaris 10," 2012. [Online]. Available: http://www.oracle.com.

[22] Micron Technology Inc, "DDR3 SDRAM datasheet." [Online]. Available: http://www.micron.com.

[23] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16 –19, Jan.-June 2011.

[24] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Proceedings of Hot Chips*, vol. 23, 2011.

[25] Hybrid Memory Cube Consortium, "HMC specification 2.0." [Online]. Available: http://www.hybridmemorycube.org/.

[26] S. J. E. Wilton and N. Jouppi, "Cacti: an enhanced cache access and cycle time model," *Solid-State Circuits, IEEE Journal of*, vol. 31, no. 5, pp. 677–688, May 1996.

[27] D. Bovet and M. Cesati, *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005.

[28] J. Mauro and R. McDougall, *Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture*. Pearson Education, 2006.

[29] C. Villavieja, V. Karakostas, L. Vilanova, Y. Etsion, A. Ramirez, A. Mendelson, N. Navarro, A. Cristal, and O. Unsal, "Didi: Mitigating the performance impact of tlb shootdowns using a shared tlb directory," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, Oct 2011, pp. 340–349.

[30] Oracle Corporation, "Oracle VM VirtualBox," 2015. [Online]. Available: http://www.oracle.com.

[31] S. Franey and M. Lipasti, "Tag tables," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, Feb 2015, pp. 514–525.