Efficient Cache Coherence for processing in 3D Stacked Memory

Amirali Boroumand
amirali@cmu.edu
Carnegie Mellon University

Abstract

DRAM is now a major system bottleneck, and it's latency, bandwidth and power consumption makes it as a key design limiter in almost all computing systems. The magnitude of this problem is growing as well. A growing number of applications requiring low latency and high responsiveness are also adopting an "in-memory" style of processing, where much of the (large) dataset is held in main memory rather than in storage [3] [2] [29], which increase the pressure on main memory system. One potential solution to tackle these challenges is processing in 3D stacked memory, which provides a number of benefits in terms of latency, bandwidth and power consumption. However, lack of efficient cache coherent support overwhelms the benefit of performing the computation near data.

In this paper, we propose an efficient cache coherence mechanism for processing in 3D-stacked memory (PIM). The key observation behind our idea is that maintaining cache coherence for PIM, using conventional directory protocol, makes the directory bandwidth and resources a major bottleneck. It also puts a huge pressure on CPU caches. We propose to re-organize the directory and tag array to enable region-based query as well as block-based query. This simple re-organization allows us to efficiently obtain coherence information and significantly reduce the number of tag-look ups and invalidations.

We evaluate our proposed method assuming that the PIM module is capable of performing bulk copy operation. Our evaluation shows that such simple re-organization reduces the number of accesses to the directory and caches up to 73% and 67%, respectively. The result also indicates that our approach improves total execution time in a multicore system by 15.7% on average.

1. Introduction

Main memory has become the major system bottleneck in all computing devices. The latency of DRAM is now a performance bottleneck in almost all systems. For example, memory accesses contribute an average of 70% to the indexing execution time in conventional databases [14]. This is also extremely critical for latency sensitive applications such Online Transaction Processing applications [1] [10].

DRAM's bandwidth is also a major concern for many systems. With the emergence of more cores and various agents on chip, the memory pin bandwidth has become an increasingly precious resource that determines system performance [11]. This problem is exacerbated for emerging big data and memory-bandwidth-intensive workloads. Conventional and emerging scale-out server workloads, such as OLTP, Web Search, etc., exhibit an abundance of request-level parallelism which requires huge amount of bandwidth.

Power consumption is also now another first-order design constraint in DRAM systems mainly due to the fact that the main memory accounts for significant fraction of total system energy consumption. There are several prior works that attributes 25-40% of data center power to the DRAM [28].

The magnitude of this problem is growing as well. Workloads targeted at the next generation of large-scale compute systems, such as big data analytics, financial applications, recognition/mining/synthesis (RMS), scientific/HPC applications, databases, etc., are expected to be increasingly data-intensive. A growing number of applications requiring low latency and high responsiveness are also adopting an "in-memory" style of processing, where much of the (large) dataset is held in main memory rather than in storage. There is, therefore, increasing pressure on the memory system, and an emergence of datacenters and systems equipped with very large amounts of main memory.

One potential solution to address DRAM's shortcoming is processing in 3D-stacked memory. 3D-stacked memory is a new technology that allows stacking logic on top of DRAM. Hybrid Memory Cube (HMC) [12] is a commercial example of this technology. Stacking logic on top of DRAM enable very fast access to the memory. Furthermore, it provides an abundant bandwidth because the interface is no longer constraint by the processor pin count. Finally, PIM eliminates the huge fraction of data movement over the memory bus, thus leading to substantial reduction in power consumption [8].

However, lack of efficient cache coherence could offset the benefit of processing in the 3D-stacked memory. The PIM functions typically operate on a large amount of data. As a result, using the conventional directory protocol to maintain coherence makes the directory bandwidth and resources a major bottleneck. Our evaluation reveals that enforcing cache coherence for the PIM, using the conventional directory protocol, could increase the number of accesses to the directory up to 5x. It also increases the number of request sent from directory to the caches up to 4.9x. Our evaluation also indicates that, in a multicore system, offloading computation to the PIM module could slow down the execution time of other cores, mainly due to in efficiency of conventional cache coherence.

In this paper, we propose to re-organize directory structure to enable region-based query as well as block-based query. Our

proposed mechanism acquires coherence information through region-based queries. In fact, it issues a single query to get information about the sharers in a region, instead of querying the directory multiple times for each cache line in the region. We also propose to re-organize tag array in such a way that it allows us to perform the tag look-up at a region-based granularity, rather than searching for region's cache blocks at cache-level granularity. Our proposed structure also enables system to perform write-backs/invalidation in an efficient way.

We quantitatively evaluate our proposed method. Assuming the PIM module is capable of performing bulk copy operation, we model the PIM module as an accelerator device and use a cycle accurate, full system simulator to evaluate our mechanism. The result shows that our proposed method could reduce the number of accesses to directory and caches by 73% and 67%, respectively. The result also shows that our mechanism eliminate the slow down of other cores in by improving the total execution time by 15.7%.

The paper is organized as follows. Section 2 presents the background and related work. Section 3 describes the motivation and breaks down the problem. Section 4 introduces our proposed method and its various components. Section 5 explains our methodology. Section 6 presents the results. Finally, Section 7 outlines our conclusion.

2. Background

2.1. 3D Stacking Technolgoy

3D stacking is a new technology that enables combining different process technologies in one package. The technology places transistors in both the vertical dimension as well as the traditional horizontal plane. Consequently, It enables a high-bandwidth, low-latency, and low-power interface by a significant reduction of interconnect within a die. Stacking of multiple layers is achieved by using Through Silicon Vias (TSVs) [5].

3D stacking technology provides interesting opportunities. This technology enables integrating processing logic, CMOS, which is mainly optimized for speed, along with memory (DRAM) technology, which is optimized for density, on a same chip. Thus, one can build a high performance and low power in-memory system by shipping data-intensive computations to the memory side and enjoying less latency and abundant bandwidth from computing data near memory.

2.2. PIM

The idea of PIM can be traced back to 1970 [27]. In fact, a large number of prior works attempted to make DRAM intelligent [26] [7] [15] [9] [19], [18] [13], instead of having a dumb device. Unfortunately, due to fabrication limit and lack of coherent programming model, these works had limited adoption in real systems. Prior works on PIM can be classified based on the role of the PIM chip. Some of them attempted to replace memory chips with more intelligent ones while using the same standard memory interface, such as FlexRAM [4], Active Page [6] and DIVA [3]. FlexRAM was developed at UIUC and aimed to replace DRAM chips by FlexRAM processor-memory chips.

The FlexRam processor-memory chip contains many simple compute engines called P.Arrays that are finely interleaved with DRAM macro-cells. To increase the usability of P.Arrays, a low-issue superscalar RISC core is included which is called P.Mem. P.Mem, coordinates the P.Arrays and executes serial tasks. Without the P.Mem, these tasks would need to be performed by the commodity microprocessor in the workstation or server (P.Host) at a much higher cost.

Active Page, developed at UC Davis, was another project that attempted to enable in-memory computing. The idea was to associate simple functions with each page of memory. It considers each subarray as an active page and assigns a small VLIW processor and a cache to that page. Each processor, associated with an active page, can access the memory associated with it, and the central processor can access the complete address space. There is no support for direct communication between Active pages. Central processor reads the data from one and writes to another one. As a result, such approach keeps the system simple and operating system can maintain control of virtual memory.

In addition to these works that attempt to make memory intelligent while using same standard memory interface, there are other works that propose more aggressive change to the traditional system architecture, such as IRAM [19] and Smart Memory [17]. IRAM project was developed at UC Berkeley. They proposed to merge CPU, main memory and cache into on chip, which leads to a number of potential benefits. For example, memory bus width is no longer influenced by Pin constraints and all memory accesses remain on-chip.

All these prior works attempt to enable in-memory processing through proposing a novel architecture. However, none of them address the cache coherence problem.

2.3. Cache Coherence

2.3.1. Coarse-grain Cache Coherence Several prior works proposed to track coherence at coarse granularity, mainly to reduce broadcast bandwidth. Jason F. Cantin et al [6] proposed to employ coarse-grain cache coherence to optimize the performance of conventional snoopy cache coherence mechanism. It tracks the sharing information at coarse granularity to filter unnecessary broadcasts. Each local cache has a Region Coherence Array which maintains coarse-grain coherence state over large, aligned memory regions. When a memory request misses in cache, the RCA is checked. If the RCA already has the permission for the region the memory request belongs to, the broadcast is no longer required and the memory request is forwarded to memory. Otherwise, in the event of RCA misses, the memory request is broadcast to all other caches to obtain the coherence state for the region to which the memory request belongs.

Stream registers [23] had the similar approach. It avoids broadcast to the caches by tracking information about contiguous regions. Instead of having fixed-size regions, Stream

registers enables changing region's size dynamically.

Jason Power et al. [20] aimed to support hardware cache coherence for integrated CPU-GPU systems that share the same address space. GPU applications often exhibit high spatial locality and requires much higher bandwidth than CPU applications. As a result, the directory bandwidth and resources, such as MSHR registers, becomes a major bottleneck because it cannot handle all these outstanding requests. The paper proposed to track sharer information at coarse-grain granularity by replacing conventional directory with region directory which tracks the sharer information at region granularity. The paper also added a buffer, called region buffer, to cache coherence information. When a request misses in the L2 cache, it first consults region buffer. If the request hits, it is forwarded through direct access bus. If the request is a miss in the region buffer, it is forwarded to the region directory.

Although these coarse-grain approaches could potentially avoid unnecessary broadcasts, it leads to significant increase in the number of spurious invalidations [24]. The reason is that accessing each cache block on each region results in invalidating the whole region for the other cores, even if they don't access the same cache block. As a result, it makes the region being ping ponged between the cores and increase the probability of false invalidations. In contrast, our approach, allows system to track coherence at fine granularity.

2.3.2. Reducing directory resource and bandwidth There are a number of prior works which attempted to reduce directory resources and bandwidth. TurboTag [16] aimed to reduce the directory energy consumption. It uses Bloom filters to reduce the number of directory look-ups . Each directory bank has a counting Bloom filter that is accessed prior to the directory look-up.

Multi-Grain Coherence Directory (MGD) [30] reduces the directory entries by refining granularity of the directory entries dynamically. It is based on the observation that while many cores may access a block at different times, at any given moment that block might only be cached by a single core. Based on this observation, MGD proposes a simple dual-grain directory which tracks individual cache blocks and coarse-grain regions of 1KB to 8KB.

Spatiotemporal Coherence Tracking (SCT) [22] aims to provide a scalable directory. It attempts to decrease the number of directory entries by tracking the sharer information at both the region and block levels. It further reduces the directory size by keeping track of private data at region level.

These approaches could be potentially combined with our method to reduce the directory overhead.

3. Motivation

To make the PIM module generally applicable, an efficient cache coherence mechanism is required. The PIM module accesses and modifies data directly on the memory without going through on chip caches. Therefore, other copies of data in system should be kept coherent to ensure cache coherency. In this section, we analyze the bottlenecks of conventional directory protocol. We assume that the PIM module is capable of performing bulk copy operation. Since the PIM module is capable of performing bulk copy operation, it could be used to accelerate many application that use Copy-on-Write technique [25]. Copy-on-Write is a technique used by most modern operating systems (OS) to postpone an expensive copy operation until it is actually needed. For example, one of the most important use cases for Copy-on-Write is fork system call. Fork system call is used to create a new child process with the same memory image and execution state. Invoking fork system call generates many expensive Copy-on-Write operations because the entire address space of parent process should be copied to the child process. As a result, shipping computation to the memory side will substantially improve the performance of the fork system call.

We use a simple micro benchmark that approximately emulates the Copy-on-Write technique used in fork system call. The micro benchmark is a simple program that creates two arrays, source and destination respectively. It initializes the source array with random values, performs some computation with it and then copies it to the destination array. We use the PIM module to perform the expensive copy operation. The PIM module is modeled as an accelerator device in x86 architecture using Gem5 full system simulator [4]. The detail description of our methodology is provided in Section 5.

Figure 1 shows the number of accesses to the directory for this micro-benchmark on a dual core system. We perform the evaluation by varying the size of destination and source array, which represents the number of DRAM pages involve in the copy operation. There are two configurations. First one is the baseline configuration which performs Copy-on-Write without using the PIM module, and the second configuration, which is labeled as PIM, offloads the Copy-on-Write operations to the PIM module.

As it is shown in the Figure 1, offloading bulk copy operation to the PIM module leads to a large number of directory accesses. In fact, there is up to 4.2x increase in the number of accesses to directory. This large number of accesses not only increases power consumption and average directory access latency but also requires infeasible number of MSHRs to handle those outstanding requests.

In addition to the significant increase in the number of accesses to the directory, there is a substantial increase in the number of requests sent from directory to caches, mainly to write-back dirty cache blocks or invalidate stale lines. Figure 2 shows the total number of request received by caches from directory. As for 64 page DRAMs, shipping computation to the PIM module result in 4.9x increase compared to the baseline.

This problem is exacerbated when the number of cores increases to 4. The Figure 3 shows the total number of directory accesses in a system with four cores. The baseline is a system consists of four cores and each of those cores executes the

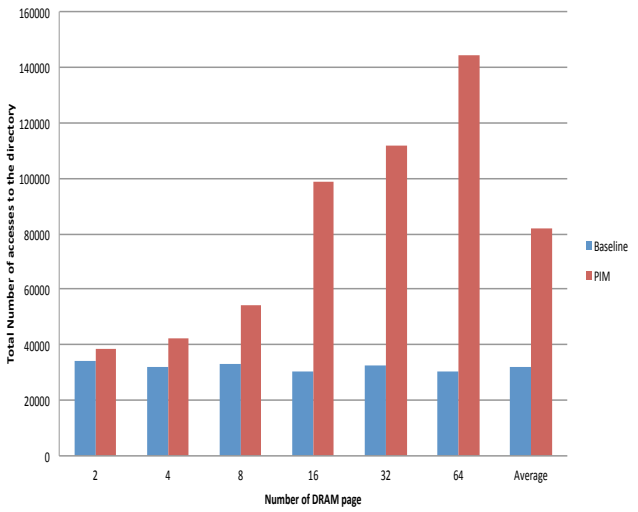

Figure 1: Total number of accesses to the directory in a dual core system

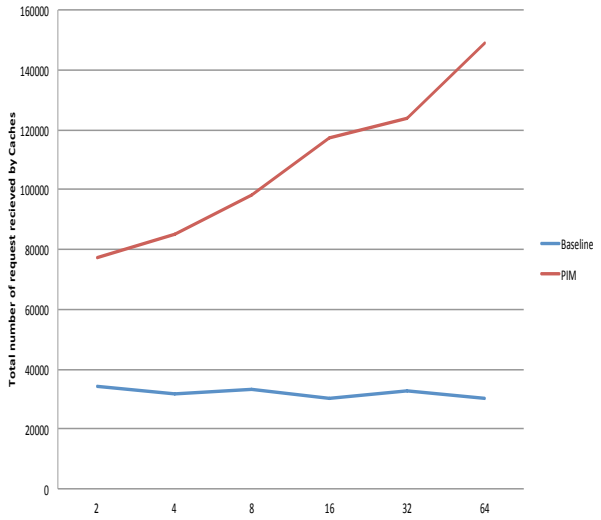

Figure 2: Total number of accesses to the caches

micro-benchmark. In PIM-1 configuration, one core uses the PIM module to perform bulk copy operation, while the other three cores execute the micro benchmark without using the PIM module.

As it is shown in Figure 3, the total number of accesses to the directory is up to 5x higher than the baseline. The same goes for the number of invalidation/write-back to the caches from the directory.

We also evaluate the total execution time for the same multicore (four cores) configuration in the presence of the PIM module. It is normally expected that when one core exploits the PIM module to perform the expensive bulk copy operation on the memory side, the total execution time would decrease, mainly due to smaller latency and abundant available bandwidth for the PIM module. However, as it is plotted in Figure

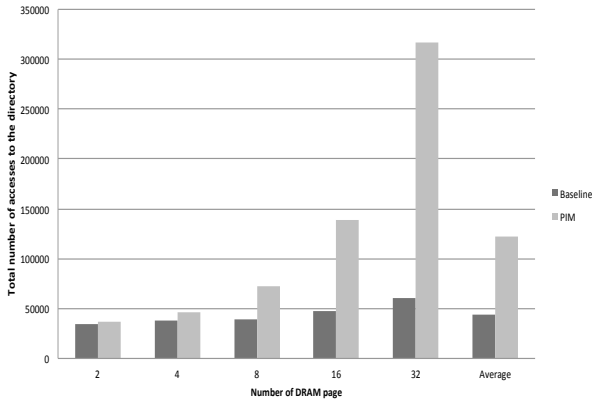

Figure 3: Total number of accesses to the directory in a system with four cores

4, the result indicates that using PIM module leads to an average 9.3% increase in the total execution. Further analysis shows that this performance degradation is because of two main reasons. First, to maintain cache coherence for the PIM module, all cache blocks in the destination array should be invalidated. Therefore, all subsequent accesses to those cache line result in off-chip memory accesses, which significantly increase execution time. Second, the large number of accesses to the directory and caches from core1 delay other cores demand request significantly. As a result, the total execution time slightly increases even though the PIM module significantly accelerates the Copy-on-Write operation on core1.

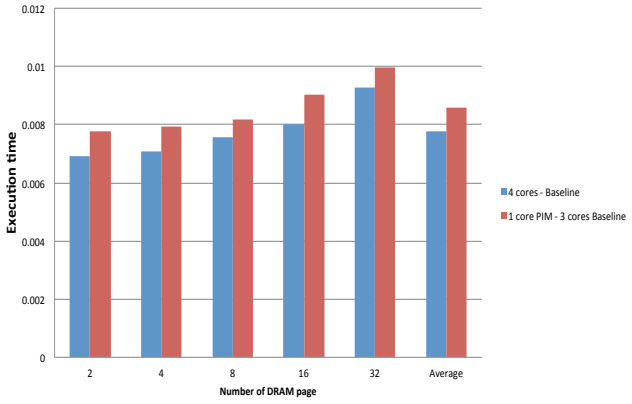

Figure 4: Total execution time

The result reveals that lack of efficient cache coherence support could substantially offset the benefit of processing in memory. Our proposed method aims to reduce this overhead to make PIM generally applicable and enable efficient processing in 3D Stack memory. Particularly, our proposed mechanism should serve three purposes:

- The first goal is to efficiently obtain coherence information. In particular, we aim to improve the directory bandwidth and reduce the required number of MSHRs.

- The second goal is to reduce the pressure on CPU caches. We aim to propose a mechanism that performs the tag look-up and invalidations in a low cost manner.
- Our final goal is to have the best of two worlds. It means that our proposed mechanism allows system to track coherence at fine granularity while enables efficient region-based queries.

4. The Proposal

In this section, we describe our proposed mechanism to maintain cache coherence in the presence of the PIM modules.

4.1. Obtain Coherence Information

The key observation is that the PIM modules operate on a large region of data. Therefore, as our evaluation indicated in the previous section, employing conventional directory protocol leads to several issues. First, to maintain cache coherence, each cache line in the region should send a query to obtain information about other sharers, which leads to excessive directory look-ups and makes directory bandwidth a major bottleneck. Particularly, the result in previous section showed that the number of directory look-ups increased up to 5x. Those queries also require a large number of MSHRs to track all outstanding accesses. However, supporting many MSHRs is not feasible because MSHR is implemented using content-addressable memory (CAM) which is expensive in terms of power and area.

To mitigate these overheads, we propose to acquire coherence information through region-based queries. That is, our proposed mechanism issues a single query to get information about the sharers in a region, instead of querying the directory multiple times for each cache line in the region. For instance, if we assume that each region is a DRAM page and the PIM module operates on an area comprising 4 DRAM pages, our proposed method could get the coherence state via only 4 queries, while the conventional directory protocol requires 256 queries.

The propose mechanism decouples sharer information from directory and stores it in a separate structure, called PIM directory. As it is shown in Figure x, each entry of the PIM directory corresponds to a DRAM region and keeps track of sharers for all cache blocks in that region. Each entry contains a bit vector indicating which core has a copy of the block. There is also a tag associated with each entry that indicates the region's address. This organization allows us to query information about sharers in a region by accessing the bit vector in the corresponding entry, instead of querying the directory multiple times for each cache line in the region.

4.2. Reduce the pressure on CPU Caches

For any block cached on the CPU, the directory should send a request to the sharer(s) to either invalidate or write-back the block. Given that there are many cache lines in the region, it leads to considerable interconnect bandwidth consumption and numerous tag look-ups and write-backs/invalidations. It also could degrade the CPU performance for the access contention

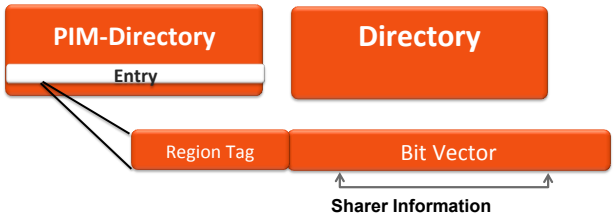

Figure 5: The PIM directory structure

on the tag stores.

To reduce these overheads, we propose to re-organize tag array in such a way that the coherence information is stored in a separate structure, called Coherence Index. As it is shown in Figure 6, the Coherence Index has multiple entries, each keep coherence information associated with cache blocks located in a region. Decoupling coherence information from tag array and storing them in Coherence Index structure allows us to perform the tag look-up at a region-based granularity, rather than searching for region's cache blocks at cache-level granularity. It also enables system to perform write-backs/invalidation in efficient way. The proposed structure, Coherence Index, can be employed to issue such operation efficiently. Since a region can be accessed using one query, instead of invalidating cache blocks located in the region one by one, all blocks can be invalidated by setting invalid bits in the vector bit corresponding to the region.

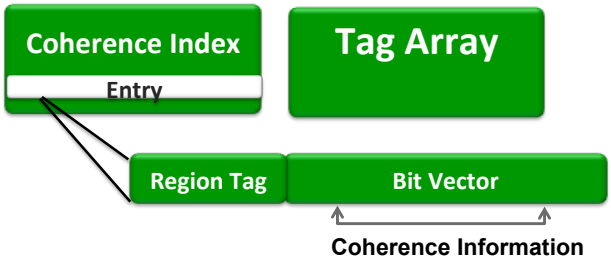

Figure 6: The Coherence Index structure

4.3. Example

In this section, we provide an example to illustrate our proposed method. Assume there is a PIM module which is capable of reading from a source array, do some computation and write-back the result to the destination array. Assume that each array (source and destination) consists of 32 DRAM pages (2048 cache lines) and the baseline system has four cores that share main memory. Assume that core1 wants to offload computation to the PIM. First we consider the baseline configuration. To maintain cache coherency in the baseline, core1 needs to send a query for each cache line to the directory to get information about other sharers, thus leading to 2048 look-ups for the directory. Assuming that 50% of the region is cached, 1024 tag look-ups and write-backs/invalidation should be done. Now, we describe the steps in our proposed method. Assuming that each region consists of one DRAM page, for

each of those 32 DRAM pages, the core1 sends the corresponding region tag to the PIM directory. If it is a miss, it means that none of the cache line in that region (one DRAM page) is cached in the on-chip caches, so the main memory already has the recent copy of blocks. If it is a hit, the corresponding bit vector is accessed to obtain sharers information. The PIM directory then sends request, associated with corresponding region tag, to the sharers to either invalidate or write-back data to the memory. The cache controller forwards the request to the Coherence Index structure. Since each entry in the Coherence Index structure corresponds to a region, it simply invalidates or write-backs all cache blocks belongs to the region.

4.4. Optimizations

The proposed structure, the PIM-directory and the Coherence Index structure could enable a number of optimizations. One potential optimization for write-backs is that the cache controller can generate write-backs for the other dirty-blocks, once a dirty block is evicted from the cache. Since the Coherence Index can list all dirty blocks in that region (using coherence states), this optimization can be simply implemented.

The other potential optimization is that it could be determined whether offloading computation to the PIM module is beneficial. For instance, assume that 60% of blocks in the region are cached and also dirty. Write-back of all those dirty-blocks could overwhelm the benefits of performing the computation on the memory side. Therefore, in that case, it is would be beneficial to cancel PIM's invocation and perform the computation on the CPU side. This optimization can be implemented effectively by extending the PIM directory's entries. Each entry could be associated with a counter that tracks the number of dirty block in that region. For example, assuming each region is equivalent to one DRAM page, a 6-bit counter is required to track the number of dirty blocks.

4.5. Other Use Cases: DMA

Direct Memory Access (DMA) is a hardware mechanism that allows other peripheral components to access memory directly. As a result, CPU is not involved in the operation which improves the system throughput. Since DMA directly modify the data in the memory, there should be a mechanism to maintain cache coherence. One current approach in existing DMA devices is to do explicit write-back/invalidation of cache blocks before performing the operation. As this happens at cache block granularity, it leads to several power and performance issues. The other approach to maintain cache coherence for DMA device is to ask the OS to make the memory range uncacheable. Both of these approaches suffer from several drawbacks in terms of performance and power consumption.

Our proposed method could benefit DMA by providing an efficient cache coherence. DMA could simply act as PIM module and access the PIM directory and the Coherence Index to enforce cache coherence in a low cost and efficient way.

5. Methodology

We use Gem5 [4] full-system simulator to evaluate our proposed method. Gem5 is a modular event-driven full-system simulator which supports various instruction set architecture such as x86, ARM, Alpha, MIPS and PowerPC. We used x86 64-bit architecture for our evaluation mainly because this is the only architecture in Gem5 that fully supports various directory coherence protocols. The DRAM simulator is the DRAMSim2 [21] which comes with Gem5 and provides both timing model and power model of DRAM. As for the operating system, we used Ubuntu 11.04 along with version 2.6.32.3 of the Linux kernel. The version 2.6.22.9 of the Linux kernel is already compiled for Gem5 and it is included in the package in their website. However, as we need to communicate with the PIM module through the Linux driver, we needed kernel header files and source codes to load kernel modules in Gem5. Therefore, we obtain the version 2.6.32.3 of the Linux kernel's source codes and header files and compiled it using Gem5 configuration

We modeled our PIM module by designing an accelerator device in Gem5. The accelerator device is connected to the I/O bus and a PIM cache, which directly reads from memory and writes to it. Since the PIM module is supposed to have very small access latency to memory, we tuned the access latency for the device to accurately model PIM module while keeping other system latency as normal. We also changed bus bandwidth for PIM module accesses to memory to model the abundant available bandwidth in 3D Stacked memory.

In order to ship the computation to the device (PIM module), a driver is required to communicate with the PIM module. We implemented a Linux driver as a kernel module, compiled it for x86 architecture and dynamically load the driver after booting the system in Gem5. The driver performs the necessary step to translate virtual address to physical address, allocates the required contiguous physical pages in memory and it then use the Direct I/O to communicate with the hardware directly. We also provide the required API for user application to invoke the PIM module and communicate with that.

We use a simple micro-benchmark that emulates the Copy-on-Write technique used in fork system call. The micro benchmark is a simple program that creates two arrays, source and destination respectively. It initializes the source array with random values, performs some computation with it and then copies it to the destination array. To offload the expensive copy operation to the PIM module, we send the address of source and destination array to the PIM module using the provided API, and poll for hardware finishing . To accurately perform the evaluation, we start measuring statistics at the beginning of the micro-benchmark execution.

we summarize our configuration parameter in table 1.

6. Evaluation

In this section, we discuss our evaluation result and show the benefit of our proposed mechanism. We first discuss the

Processor	
ISA	x86-64
CMP size and Core Freq.	4-core, OoO, 2 GHz
Re-order Buffer	128 entry
Cache Hierarchy	
L1 I-cache	32KB/2-way, 2-cycle
L1 D-cache	32KB/2-way, 2-cycle
L2 Cache	4MB/8-way, shared, 20-cycle
Coherence Protocol	MOESI Directory
DRAM Parameters	
DRAM device parameters	Micron x8 4Gb chip, 8 chips
Memory Channel	DDR3 800 MHz channel, 1 channel
Number of banks	8 banks/device
DRAM scheduling policy	FR-FCFS
DRAM read/write buffer	Read:32 entries, Write:64 entries
DRAM I/O pins	64 I/O pins for CPU, 256 TSVs for PIM
tRCD	12.5 ns
tRAS	54 ns
tFaW	40 ns
tCL	12.5 ns
tRP	12.5 ns
tCWD	6.5 ns
tRC	50 ns

Table 1: Simulator Parameter

benefit of our mechanism in a dual core system. We show how our proposed mechanism tackles the directory bottlenecks and how it reduces number of accesses to the caches and the total execution time. We also show the result of our proposed mechanism when the number of cores increases to four.

6.1. The Directory

As it is mentioned in section 3, maintaining cache coherence in the presence of the PIM module leads to significant increase in the number coherent requests to the directory, which makes the directory resources and bandwidth a major bottleneck. Figure 7 shows the total number of accesses to the directory. As can be seen from the figure, combining conventional directory with the PIM directory results in substantial reduction in the total number of accesses to the directory. In fact, the total number of accesses to the directory is decreased by 47.1on average. Note that such reduction improves directory bandwidth and energy consumption, and also it allow the directory to track outstanding request with a feasible number of MSHRs.

Figure 8 present the result for the system that has four cores. As it can be seen from the figure, the reduction in the total number of accesses to the directory is even more noticeable when the number of core goes up. Leveraging our proposed method reduces the number of access by 55.7

6.2. The Caches

As we mentioned in the section 3, to make data coherent between memory and CPU caches, the directory broadcasts many invalidations/write-backs to the caches. The figure 9 indicates how Coherence Index can effectively reduce the number of tag look-ups and invalidations which are required to be done by last level cache controller. On average, Coherence Index eliminates 45.48% of these accesses on average.

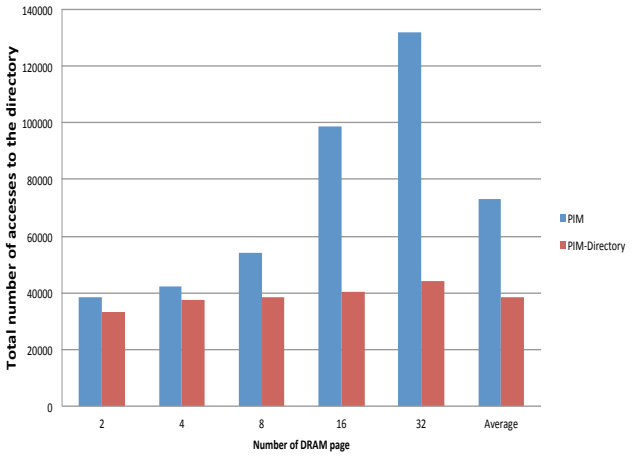

Figure 7: The number of accesses to the directory in a dual core system.

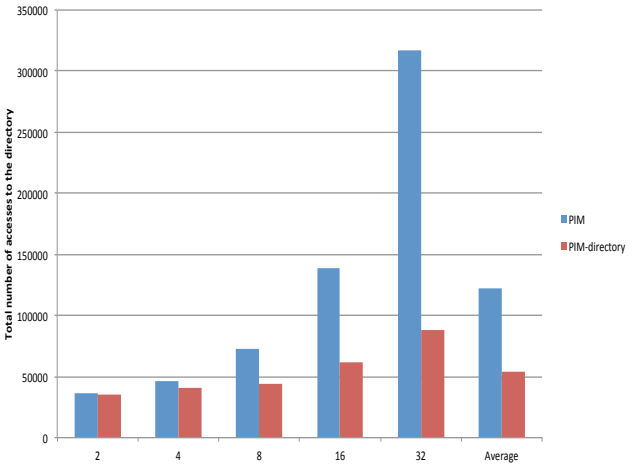

Figure 8: The number of accesses to the directory in a four core system.

6.3. Execution Time

In this section, we discuss our evaluation result for execution time. As we mentioned in the section 3, invoking PIM module to perform copy operation near the data degrades performance in a multicore system due to lack of efficient coherence mechanism. In fact, when one core in a ships computation to the PIM module, it pollutes other cores' caches and delays other cores' demand requests, thus leading to slight increase in the total execution time. Figure 10 presents the execution time for three different configurations. In the baseline configuration, all four cores perform the copy operation without invoking PIM module . In the second configuration, one core offloads copy operation to the PIM module while other three cores execute the copy operation without using the PIM module and last configuration applies our proposed method to the system. As it is shown in the Figure 10, our proposed method improve total execution time by 15.7%. However, as it is seen from the figure, the performance benefit of using PIM module com-

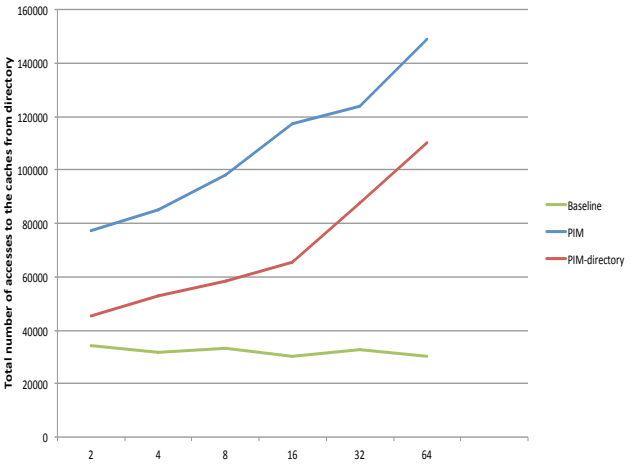

Figure 9: The number of accesses to the caches from directory in a dual core system.

pared to the Baseline is not significant. The reason is that our proposed method does not solve the problem of cache pollution. As a result, subsequent accesses to the data from other cores result in off-chip memory accesses.

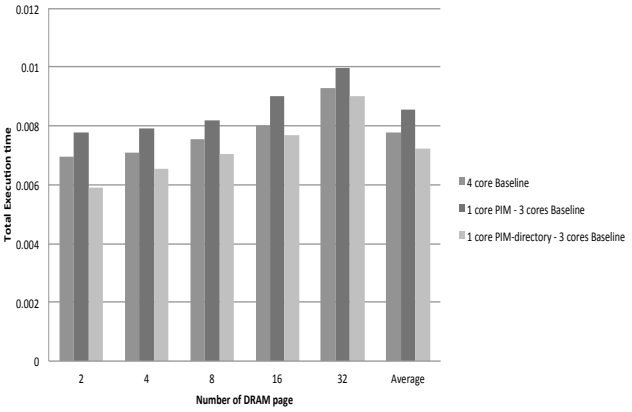

Figure 10: Total execution time.

7. Conclusion

Processing in 3D-Stacked technology is a promising solution to address DRAM's challenges such as latency, bandwidth and power consumption. However, lack of efficient architectural support for cache coherency impede the general applicability of PIM and offset its benefit. Based on the observation that conventional directory protocol is a major bottleneck for PIM, we proposed to re-organize the directory and cache tag array structure to enable region-based query. By leveraging the PIM-directory and the Coherence Index structure, the coherence information is obtained in a more efficient way and pressure on CPU caches is significantly reduced. Our result shows that our propose structure could decrease the number of accesses to the directory up to 73% and improve the total execution time by 15.7% on average.

References

[1] "Online transaction processing." [Online]. Available: http://en.wikipedia.org/wiki/Online_transaction_processing

[2] "Sap database in-memory." [Online]. Available: http://www.sap.com/pc/technology/in-memory-computing-platform.html

[3] "Sas database in-memory." [Online]. Available: http://www.sas.com

[4] N. Binkert, B. Beckman, A. Saidi, G. Black, and A. Basu, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, 2011.

[5] B. e. a. Black, "Die stacking (3D) microarchitecture," in *International Symposium on Microarchitecture*. IEEE, 2009.

[6] J. Cantin, M. Lipasti, and J. Smith, "Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking ," in *Proceedings of the 32nd annual international symposium on Computer Architecture*. IEEE, 2005, pp. 245–257.

[7] D. G. Elliott, W. M. Snelgrove, and M. Stumm, "Computational RAM: A memory-SIMD hybrid and its application to DSP," in *Custom Integrated Circuits Conference*, vol. 30. Citeseer, 1992, pp. 1–30.

[8] M. Facchini, T. Carlson, A. Vignon, M. Palkovic, F. Catthoor, W. Dehaene, L. Benini, and P. Marchal, "System-level power/performance evaluation of 3d stacked drams for mobile applications," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 923–928.

[9] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: The Terasys massively parallel PIM array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[10] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker, "OLTP through the looking glass, and what we found there," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 981–992.

[11] E. Ipek, O. Mutlu, and O. Martinez, "Self-optimizing memory controllers: A reinforcement learning approach," in *Computer Architecture (ISCA), 2008 ACM/IEEE International Symposium on*. IEEE, 2008.

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

[13] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, "FlexRAM: Toward an advanced intelligent memory system," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*. IEEE, 2012, pp. 5–14.

[14] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, "Meet the walkers: Accelerating index traversals for in-memory databases," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013, pp. 468–479.

[15] P. M. Kogge, "EXECUBE-A new architecture for scaleable MPPs," in *Parallel Processing, 1994. Vol. 1. ICPP 1994. International Conference on*, vol. 1. IEEE, 1994, pp. 77–84.

[16] P. Lotfi-Kamran, M. Ferdman, D. Crisan, and B. Falsafi, "TurboTag: Lookup Filtering to Reduce Coherence Directory Power," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*. ACM, 2010, pp. 377–382.

[17] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, *Smart memories: A modular reconfigurable architecture*. ACM, 2000, vol. 28, no. 2.

[18] M. Oskin, F. T. Chong, and T. Sherwood, *Active pages: a computation model for intelligent memory*. IEEE Computer Society, 1998, vol. 26, no. 3.

[19] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent RAM," *Micro, IEEE*, vol. 17, no. 2, pp. 34–44, 1997.

[20] J. Power, A. Basu, J. Gu, M. Hill, and D. Wood, "Heterogeneous system coherence for integrated CPU-GPU systems," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE/ACM, 2013, pp. 457–467.

[21] 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.

[22] A. Safaee, "Spatiotemporal Coherence Tracking," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE/ACM, 2012, pp. 341–350.

[23] V. Salapura, "Improving the accuracy of snoop filtering using stream registers," in *Proceedings of the workshop on MEmory performance: DEaling with Applications, systems and architecture*, 2007.

[24] D. Sanchez and C. Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *Proceedings of the 18th IEEE international symposium on High Performance Computer Architecture*. IEEE, 2012.

[25] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch *et al.*, "Rowclone: Fast and energy-efficient in-DRAM bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013, pp. 185–197.

[26] D. E. Shaw, S. J. Stolfo, H. Ibrahim, B. Hillyer, G. Wiederhold, and J. Andrews, "The NON-VON database machine: A brief overview," *IEEE Database Eng. Bull.*, vol. 4, no. 2, pp. 41–52, 1981.

[27] H. S. Stone, "A logic-in-memory computer," *Computers, IEEE Transactions on*, vol. 100, no. 1, pp. 73–78, 1970.

[28] A. Udipi, N. Muralimanohar, and R. Balasubramonian, "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores," in *Computer Architecture (ISCA), 2010 ACM/IEEE International Symposium on*. IEEE, 2010.

[29] M. Zaharia, M. Chowdhury, and T. Das, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing ," in *Proceedings of NSDI*. IEEE/ACM, 2012.

[30] J. Zebchuk, B. Falsafi, and A. Moshovos, " Multi-grain coherence directories," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE/ACM, 2013, pp. 359–370.