Access Intensity and Row Buffer Locality-based Data Migration in Hybrid Memory Systems

A M. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Soumya Asati
(224101046)

under the guidance of

Prof. Hemangee K. Kapoor

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

<u>CERTIFICATE</u>

*This is to certify that the work contained in this thesis entitled "**Access Intensity and Row Buffer Locality-based Data Migration in Hybrid Memory Systems**" is a bonafide work of **Soumya Asati (Roll No. 224101046)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and it has not been submitted elsewhere for a degree.*

Supervisor: **Prof. Hemangee K. Kapoor**

Professor,

May, 2024 — Department of Computer Science & Engineering,

Guwahati. — Indian Institute of Technology Guwahati, Assam.

Abstract

*This thesis "**Access Intensity and Row Buffer Locality-based Data Migration in Hybrid Memory Systems**" explores the data migration techniques within the design structure of combining 3D-stacked DRAM along with off-chip DRAM to form a hybrid memory system. Various approaches and challenges associated with such hybrid systems are identified through a review of related literature, providing a detailed overview of the existing research area. This study explores fundamental techniques, particularly threshold-based or cache-based approaches, to improve migration strategies such that overall performance should be optimized. We propose an innovative hybrid memory architecture, AI-RBL, for parallel architecture and threshold-based technique, which leverages row buffer locality and access intensity for efficient memory management. We evaluate the performance, energy consumption, and migration overhead of AI-RBL in comparison to baseline configurations and state-of-the-art techniques. Extensive simulations and analyses demonstrate that AI-RBL offers improvements in key performance metrics. These findings highlight the effectiveness of AI-RBL in optimizing memory management and enhancing overall system performance in hybrid memory architectures.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Modern computing systems, like the demand for big data analytics, high computing systems, data centers, and many machine learning applications, are continuously growing and incorporate multi-core design methodologies, which led to the increasing demand for extra main memory to serve the requirement for many data-intensive applications. Therefore, utilizing the Hybrid memory system is indeed a potential solution. However, memory hierarchy has different characteristics, due to which the system's performance has a large impact. Although many technology advancements come into the picture like 3D stacking technology that provides high bandwidth as they are placed near the processor, using it alone doesn't fulfill the scarcity of resources. So, it must be complemented by the high-capacity memory with low access latency and bandwidth. One hybrid system combines the near memory with low capacity as 3D-DRAM and the other with high capacity as a conventional DRAM (DDR3/DDR4).

1.1.1 Data Migration

The process of data movement between different levels of memory is referred to as Data Migration. In a hybrid memory system, different memory technologies are used to take benefits of each in different contexts. For example, Virtual Memory is used when the program size is much larger than the main memory, and it is nothing but an illusion to give programmers that their programs are large enough as main memory and can be executed in it. In general, it happens like the programs are stored in secondary memory. Whenever there is a need to execute that part of the program, it is brought into main memory executed, and taken back into secondary memory, which is referred to as the swap-in and swap-out process. Similarly, Data Migration is also swapping the data between two memories, one connected with a high bandwidth channel and the other connected with a conventional DDR4 channel. This process is necessary because these two memories have differences in cost, access time, and capacity, which will help optimize the process execution and improve the overall performance. It has some components required to make this process functional.

- **Remapping Table:** This table holds the entries of Far Memory segments that migrated to Near Memory with their physical address. It will used if that segment has to be sent back to its actual location. This table is also stored in Near Memory such that to access it, latency should be low.

- **Inverted Remapping Table:**This table holds the entries for the NM segment, which are migrated to FM due to the FM segment being brought to NM to access fast. It also stores in NM only to fetch the information about the NM segment where it is mapped to in FM.

- **Migration Rule:** The decision rule triggers the migration, that is, the criteria when we require swapping because excessive migrations bring overheads and may lead to performance degradation, so the decision should be accurate and critical.

1.2 Existing Challenges

There have been many advances in memory technologies; we have discussed that 3D-stacking technology is a trend nowadays, which can be placed in the same package or above the processor that provides high bandwidth connected with a high bandwidth channel, but it doesn't serve the purpose of memory requirement solely because it has limited capacity and DRAM has its own challenges in terms of refresh power and leakage [1]. On the other hand, conventional DRAMs, which are treated as high-capacity memory, serve the memory requirement, but it has low access latency as they are connected to a low bandwidth channel that is placed far from the processor. Therefore, both are used as complementary to each other to serve the purpose of memory requirement.

The challenges faced in the existing approaches of Data Migration algorithms in Hybrid memory systems are either cache-like or threshold-based methods. In the cache-like method, the data segment of Far Memory is migrated to Near Memory when accessed and would be written back using the Least recently used method from the NM when it has no more space to accumulate the FM pages. The problem with this approach is there would be a high overhead of migration and metadata management.

The other approach is threshold-based; it works such that whenever the threshold is crossed, the data is selected for migration. For each segment of Far Memory, the access counters are maintained and set some threshold values. If these counters are exceeded, then that segment is considered to be placed in Near Memory. The problem is that the working set of applications is dynamic in nature and can change frequently, so the threshold-based method is not beneficial in every circumstance.

Therefore, we need such a migration algorithm that will take care of changing working set applications needs and will not yield higher overheads; also, the benefits of data migration should not be suppressed and should improve the system's performance.

1.3 Motivation

The challenges mentioned above in existing Data Migration algorithms in Hybrid memory systems led to the motivation that there is a critical need to enhance the overall performance and the efficiency of modern computer architectures. As technology advances day by day, there is a constant demand for powerful and responsive computing systems, and memory plays an important role in meeting these expectations. The combination of 3D-stacking technology and traditional DRAMs shows an effective strategy for balancing the limitations and benefits of various memory types. This hybrid memory architecture, however, introduces its own set of challenges. Although providing high bandwidth, the limited capacity of 3D-stacking technology requires a complementary relationship with high-capacity but latency-prone conventional DRAMs. This balance is required to meet the different memory requirements of modern applications and workloads.

The existing approaches of data migration, either cache-like or threshold-based, both have their own shortcomings that restrict the fixed method used in this type of memory technology. For instance, we have discussed that the cache-like method has major overheads due to the fact that every access to FM brings data into the NM, which requires lots of metadata management every time. Whereas the threshold-based approaches face challenges in adapting to the dynamic nature of application working sets, i.e., the fixed threshold will limit the overall performance as it might or might not be effective for some applications.

So, the motivation lies in the search for such a migration algorithm that can dynamically respond to the changing needs of applications and contribute positively to the overall system performance. We aim to find this more effective solution to the growing demands of modern systems.

1.4 Organization of The Report

This chapter describes the introduction of our thesis work, i.e., the background of the current architecture of memory systems, which one we are working on, and the Data migration problem and its terminologies. Next, we have explored the existing challenges of the data migration approaches in these types of memory architectures. And through these existing challenges, we derive some sort of motivation for what we are focusing on and which type of problem we will tackle next. The organization of the report is listed below:

- **Title Page:** It includes the report's title and other relevant details like the author's name, supervisor's name, and institution information.
- **Table of Contents:** It provides a curated list of sections and subsections in the report and their corresponding page numbers.
- **List of Figures:** It contains the list of all figures used in the report.
- **Introduction:** It gives a brief description of the topic with some background information along with the challenges and motivation.
- **Literature Survey:** It explains the research of the current methods used in the field of memory scheduling techniques in the study.
- **Proposed Methodology:** It explains the details of the proposed algorithm for data migration in hybrid memory architectures.
- **Evaluation & Results:** Interpret the results and provide a detailed analysis of the findings.
- **Conclusion:** It summarizes the main points discussed in the report. Highlight the area of the research, motivation, and the work that has been done along with the results. Lastly, the future work.
- **References:** It provides a list of all the sources cited in the report.

Chapter 2

Review of Prior Works

This chapter details the literature survey while exploring the field's research area. It provides comprehensive knowledge of the current state of the art and the direction of the research that has been done.

2.1 Hybrid Memory System Architecture

Hybrid memory systems combine different components to provide the advantages of these components in a single architecture, aiming to exploit the best of these to alleviate the impact of performance [2]. In general, it combines the two memories, one of which offers high bandwidth and low access latency with higher cost but limited capacity. In contrast, the other provides high density with lower cost, low bandwidth, and high access latency. One such model of a hybrid memory system is the 3D-stacked DRAM (fast memory component) and an off-chip (DDR3/DDR4) DRAM (slow memory component), which we have considered for our work. Another model is DRAM, and NVM-related work is [3]. Also, related work on NUMA architecture is [4].

3D stacked DRAM is placed near the processor connected with a high bandwidth channel called Near Memory. It has a smaller capacity and needs to be supported with the larger capacity memory, which offers low bandwidth, referred to as Far Memory. All the prior works are broadly categorized into two leading approaches, i.e., the first uses the Near Memory DRAM as a cache, and the other uses both the Near Memory and Far Memory as the flat address space.

The two categories are:

1. Stacked DRAM as Cache
2. Stacked DRAM as Part-of-Memory

2.2 Stacked DRAM as a Cache

The faster memory component in a Hybrid Memory System can be treated as a cache in some prior approaches. This near memory is used similarly to the cache memory functions. It copies the data of far memory to near memory, i.e., caching brings down the amount of capacity of near memory from the system, and the NM space can't be visible to OS.

Using a cache has its advantages and disadvantages. Caches adapt quickly to application changes and enhance the overall performance due to their spatial and temporal locality property [5], where many memory references access fast memory and provide software transparency. However, it copies the data from far memory to near memory, reducing the overall capacity of the memory system, which can impact the performance and even lead to performance degradation where the memory footprint is very large. Also, tag and data management require proper organization.

For example, if a system has an overall capacity of 20GB and 4GB out of 20GB can be taken away as a cache, then only 16GB out of 20GB can be visible to the OS. If this 4GB space of NM can be exposed to OS, then more tasks can be scheduled by OS, improving the

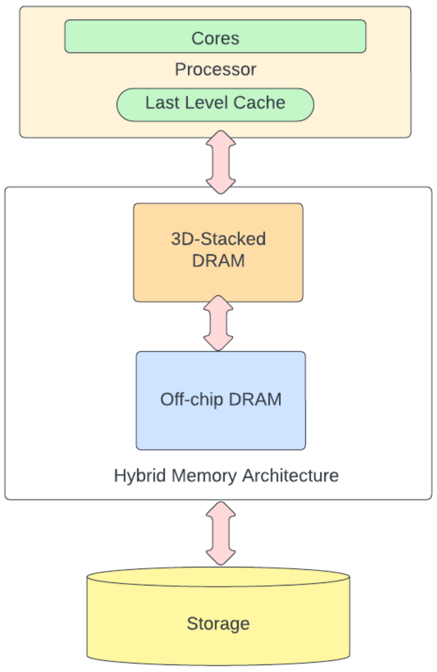

Fig. 2.1 Hybrid system with DRAM as cache

overall waiting time of the processes, improving the throughput, and reducing page faults.

2.2.1 Chameleon: Dynamically Reconfigurable System

The problem with the cache is that it takes away the capacity of NM and can not fulfill the requirement of a high memory footprint, leading to performance degradation. So, [6] propose Chameleon, a hardware-software-based architecture that dynamically switches on the two memory regions, i.e., PoM and cache modes, based on the memory requirements. It is a hybrid architecture that bridges the gap between cache-based and PoM-based architecture. When the memory footprints are very high, then Chameleon uses the Near Memory as the part of memory with Far Memory as a flat address space to serve the high memory requirement, which also allows the migration between these two memories that is swapping the data rather than coping like caching, But when the memory footprint of an application is smaller than the total available physical capacity, Chameleon taking advantage of the free space to use it as a hardware-managed cache to optimize the performance.

They have made changes in the Instruction Set Architecture (ISA) and provide two new instructions, i.e., *ISA_Alloc()* and *ISA_Free()* to support the design of Chameleon, OS will notify the hardware when there is a free page or an allocated page. The base design is an existing PoM proposal, and they have restricted the migration between the memories in segment groups referred to as "Congruence Groups." The meta-data structure is known as the Segment Restricted Remapping Table to track the remapped pages with additional information, i.e., Alloc bit Vector, Mode bit, and Dirty bit, to support the design of Chameleon. Alloc bit vector size is the number of pages in the segment group, which will denote whose pages are allocated, and mode bit will tell currently in which mode the design is working. '0' is for PoM mode, and '1' is for cache mode, and the dirty bit will indicate if the mode is in cache mode, then the page is cached is dirty or not. ISA_Alloc instruction will transition from cache mode to PoM mode, and ISA_Free instruction will transition from PoM mode to cache mode so that the free space of NM can have the advantage of caching. By this, they have improved the average performance of 11.6% over PoM architecture.

2.2.2 Hybrid: Combining Caching and Migration

Caching and Migration are different approaches that serve different purposes in different contexts. The goal of caching is to store the data temporarily so that subsequent access to that data can be served quickly, reducing the need to access the slower memory. Whereas migration involves swapping the data between NM and FM such that the frequently access data can be maintained in the fast memory and the least frequent data can be present in either NM or FM. Hence, the overall access time is reduced. In Migration, the NM capacity is used as the flat address space to serve the memory requirement, whereas this NM capacity can be taken away by caching. On the other hand, migration introduces the double overhead of copying both NM and FM data, while caching only has the overhead of copying data from FM to NM. Therefore, both caching and migration have their advantages and disadvantages. Despite their differences, they have several typical troubles. One of them is to find the balance between the size of data, i.e., granularity, and the metadata overhead. If we take the cache-line size smaller, we'll have a large tag array to manage it. Similarly, if we take the migration block size smaller, large metadata is required to manage the remapping to track the data. So, the overheads should be minimal for effective performance. Another challenge is the trade-off in the granularity of the data [7]. If the granularity is coarse, it will benefit in the case of workloads with a high spatial locality, but it leads to over-fetching in the case of a low spatial locality. On the other hand, if the granularity is fine, then we can avoid the risk of over-fetching but not fully utilizing the spatial locality, depending upon the requirement.

So, [8] propose the Hybrid memory architecture system, which combines the advantages of caching with the migration scheme so that the whole capacity can't be taken by caching. Only a small part of NM serves the purpose of caching. Their technique involved the following contributions listed below:

- The tag array for the DRAM cache is small so that it can entirely fit on-chip and is further extended to be used as a remapped cache for the migrated segments. It is a set-associative cache where each entry denotes the information for the segment having valid and dirty flags and the additional information they have implemented an Access counter and two pointers, i.e., NM and FM pointer. The access counter keeps track of the number of accesses of that segment. The NM pointer is used to specify the NM location of the segment allocated to the DRAM cache, and further on, eviction will decide if it has to migrate to NM or write back to FM. The FM pointer represents the segment's physical location. to act as the remapped cache, which avoids looking up the remapping table.
- For caching, there would be two types of requests in the memory access path: an

XTA hit and an XTA miss. In XTA hit again, there would be two cases; for that particular segment, the cache-line hit or miss. If the cache line hits, we can directly use the NM pointer to read, but if a cache-line miss happens, then use the FM pointer to read the data and the NM pointer to write in the NM. In case of an XTA miss, there is no entry, so you need to allocate the entry of that segment and then check the remap table to identify if the segment is in NM or FM. If it is NM, then read the cacheline from the NM, and if the segment is in FM, then the cache line is read from the FM and write the cache line to the NM.

- On DRAM evictions, the migration decision would be taken based on the current location of the segment. If the location is NM, then there is no need to move anything, but if the location is FM, then check for the criteria that it is suitable for migration, and also, that segment should be the most eligible victim amongst all segments in the DRAM cache. After those checks, only the segment is selected for migration to NM; otherwise, all the cache lines of that segment would written back to the FM.

2.3 Stacked DRAM as a Part-of-Memory

Earlier, we have seen that one part of the research will use Near Memory as the cache. Another area of research is to utilize Near Memory as a Part-of-Memory. This approach is the inverse of the DRAM cache in that it provides the capacity of the stacked DRAM accessible to the system, allowing it to use the bandwidth of each of the memories to serve memory requests. As a result, data migration could have the benefits of both bandwidth and capacity by dynamically moving data between conventional DRAM and 3D-stacked DRAM.

PoM architectures [9] can be managed by OS or Hardware. Some work uses the Operating System to select the data to move into the 3D stacked DRAM with some hardware support. By adopting the OS, the selected data to be migrated may be better, and page

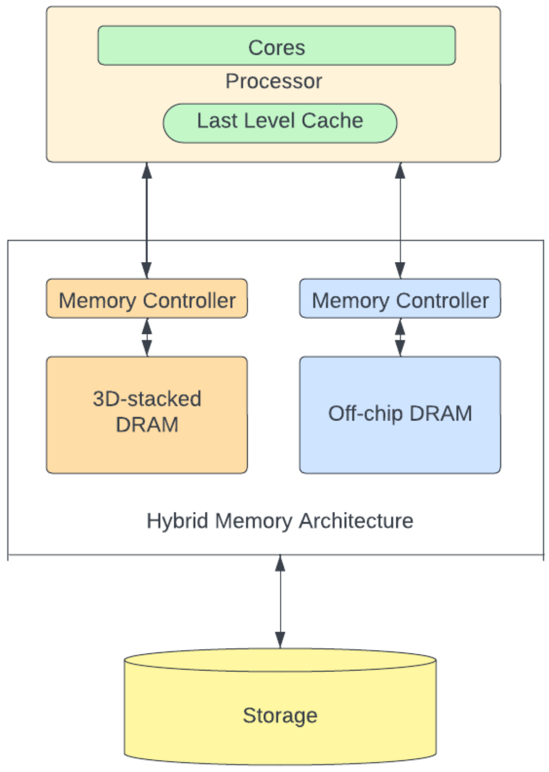

Fig. 2.2 Hybrid system with DRAM as part-of-memory

tables can be used to monitor the remapped data. However, OS schemes are slower to adapt to changes in the working set, have a large overhead, and only allow data migration at the granularity of an OS page. On the other hand, hardware-managed PoM systems perform better than OS-managed memory systems because they can quickly adjust to changing memory needs. However, more space and power are required to handle hardware address indirection and large area swaps between fast and slow memories. For hardware mechanisms to stay transparent to the operating system, address remapping in hardware must be managed. More migration granularities can be supported and faster response times are achieved by hardware mechanisms. he overheads of address remapping are affected by the migration granularity. Hence, some of the earlier works employed the idea of congruence groups—that is, to divide the memory into groups and permit data swapping between the memories within these groups—to avoid the address remapping overheads.

2.3.1 LGM: LLC Guided Data Migration

This work [10] has considered the PoM architecture, i.e., the fast memory component (Near Memory) is used as the part of memory as a flat address space with complement with the Far Memory to provide the benefits of both memories. They propose the LLC-guided Data Migration scheme (LGM), a hardware-based design that provides migration-based policy in a Hybrid Memory System, they took the best-performing state-of-the-art concerning this aspect which is Mempod[11] and SILC-FM[12] and while comparing these two they found to go with Mempod for further comparison of designs. The following are their contributions listed below.

- **Segment Selection for Migration:** Their first objective is to improve the data selection for the migration. For this, they have found that the Last Level Cache (LLC) has enough information (i.e., state and content) to guide selecting the most suitable data to reside in Near memory to be readily available when subsequent access for that occurs. But there is a difference between the cache replacement policy and the migration policy, i.e., the cache replacement works on the cacheline granularity (generally 64 bytes). In contrast, migration works on the segment granularity (generally 2 KB, which is 32 cacheline). So, the segment's cachelines present in the LLC give an idea about the spatial and temporal locality, which can be used to identify the particular segments of interest. So, for this purpose, they have taken the two counters and two thresholds, i.e., Valid and Dirty counters and Valid Threshold and Dirty Threshold. These thresholds are used to tell after which point the migration will trigger. The Valid counter is incremented when the cacheline is inserted in the LLC and decremented when it is evicted from the LLC, while the Dirty counter is incremented when there is a write-back to the LLC from the higher level cache and decremented upon eviction from the LLC. This is implemented in the migration controller mechanism, which will be responsible for required operations.

- **Reduction in Migration traffic:** The second objective is to reduce the migration traffic, so the segment that is selected for the migration purpose, instead of migrating the whole segment LGM only migrates the cacheline which is not already present in the LLC, this way on an average more than half of the migration traffic is reduced. For this purpose, they have used the DSC (Decouple Sectored Cache) as an LLC, which has the back pointers for the cacheline to point to the particular segment to keep track of the cachelines of FM segment that are present in the LLC, and if the segment is selected then all the bit vector of information about the present cacheline is forwarded to the migration controller to make the decision about which cachelines need to move in NM, it also has AW bit associated with it, i.e., Always-write-back which will set after the selection of the segment so they have always written back to memory whenever evicted from the LLC.

- **Adaptive Migration Threshold:** They have also provided the adaptive mechanism for the threshold for the migration to trigger. Setting the values for the threshold is a critical task. If the threshold value is set to maximum, then the migration only takes place when all the cachelines of the segment are entirely present in the LLC. In contrast, if the value is set to low, then migrations are taking place aggressively. So to tradeoff between these two, they have proposed the dynamically adjustable threshold algorithm which uses the fixed time interval, if the number of migrations is greater than the thresholds, then the thresholds are set to max otherwise thresholds are periodically lowered to meet the previous requirement. Also, they have taken the high watermark, indicating the maximum number of acceptable migrations. They have taken the thresholds MIN and MAX values as 4 and 32, respectively, and the value of the High watermark is experimentally set as 64 with an interval of 24 microseconds.

2.3.2 RHPM: Relative Hotness Page Migration

This work [13] is also based on the congruence group remapping flexibility similar to CAMEO [14], which is the memory is divided into groups in a certain ratio, and the data is swapped amongst the group only, although this remapping flexibility is fixed and has limited options to map the migrated segment, which leads to minimum complexity, but it is not scalable if the NM to FM ratio increases. Here, they identified that the existing migration-based strategies have not handled the extra metadata query latency, so they proposed the RHPM (Relative Hotness Page Migration), which identifies the hot page that is the page that is selected to be migrated to NM by competing amongst the set of pages in a group. The migration only triggers when any new page wins the competition, and when the data is fetching, they also fetch the metadata parallel using different channels to overlap the querying latency such that they improve the energy by 44.14% and performance by 13.34%. The following are their contributions.

- **RHPM Migration Policy:** As mentioned above, they have used the remapping flexibility as the congruence groups so the FM pages can have only one position in NM using this fixed remapping technique, and the Migration Rule is set such that whenever there is a new page whose hotness counter exceeds from the other pages of the same group then it will swap with that NM position by this it is assumed that hottest page is maintained in NM. They have benefited from congruence groups through this fixed remapping, which has a lower cost of comparison and reduces the time for address translation. The metadata required for remapping has less storage. To find the hottest page, they used one counter, the hotness counter, which has a 4-bit size. So if the page is accessed, the hotness counter will increase by 1, and other pages of the same group are decreased by 1 as they are not in that particular instance. If that c-group is accessed, only that c-group's hottest page will be compared with the currently accessed page, and changes will be made accordingly. But there is a case when the hotness counter of two or more pages will become the hottest, so to overcome this issue, they have a reset rule that is when a page is ready to migrate and become the hottest, then reset the hotness counter of other pages.

- **Handling of querying metadata:** In many of the prior works, the handling of metadata is not done in a mannered way, so they took advantage of bus-level parallelism while fetching the data, also fetch the remapped addresses, which can be useful further to overlap the latency of metadata querying. For this purpose, they use prefetcher, which stores the remapping addresses, and if the size of the remap buffer gets full, then to evict some entries from it, use the LRU policy. While prefetching the data, the actual location of the data may be NM or FM, so to identify that, they used a 1-bit predictor, which will be set if the last request was served by FM. They have also used the concept of migration buffer because, in fixed-remapping flexibility, the swapping cost is increased by swapping first the NM page to its original location and then NM and FM page swapping so to avoid this, they maintained the entry in the migration buffer and fetch NM and FM by using bus-level parallelism.

2.4 Page Migration based on Machine Learning Algorithms

Some works apply decision logic for data migration and placement using machine learning algorithms.

2.4.1 Kleio: A page Scheduler with Machine Intelligence

In data migration, the main focus of all the approaches is to identify the most frequently access pages so that they can be placed in the Near Memory to utilize the advantage of high bandwidth to be readily available for the upcoming accesses. For that, the past accesses of that data are gathered to make this decision fruitful, so Machine learning algorithms can also be applied here. One such work is Kleio [15], which tries to cover the gap between the state-of-the-art History-based scheduler and the optimistic and ideal Oracle-based scheduler. They first try to understand which machine learning should be used in this context, so they explore different options and try eliminating them based on pros and cons. Lastly, they went with the Neural Networks and used Recurrent Neural Networks. For each page, utilize the train an RNN to identify the access counts per page, and these counts in the current epoch can be further used to make decisions in the next scheduling epoch. The components of this system are below.

First, Page Selector, which identifies only those categories of pages that require machine intelligence, more precisely the pages that the history-based predictor misplaces because the history-based method can only make decisions based on the last scheduling epoch, and it may lead to a scenario that currently hot page is now no more needed as the requirement changes with time. So, the portion of pages that are misplaced by history needs sophisticated decisions. After identification, they again order them based on priority as the training for all pages requires more time. Then, the performance curve was utilized with the help of an Oracle-based scheduler. Important pages are selected and placed into the DRAM to achieve performance gain.

2.4.2 LSTM: Page Scheduling for HMS

As we have discussed, DRAM has certain disadvantages like scalability issues, high energy consumption, and high leakage, and it requires refresh power to ensure that the data inside the memory is active. So, complex analytical simulations performed in DRAMs are neither sustainable nor scalable. Hence, to overcome these shortcomings, recent approaches adopt cost-effective NVMs, which consume less energy and are persistent and highly scalable. The clustering of pages is a practical technique for identifying tradeoffs between performance and energy usage in hybrid systems, so [16] proposes a Page scheduling machine learning approach based on LSTM to provide an accurate and quick prediction of access to memory per page to take advantage of the temporal and spatial position with knowledge of neighboring pages.

Their methodology is made up of various components. First, a critical page selector is responsible for efficiently finding the critical pages that significantly impact the overall performance based on memory access. Because deploying a single LSTM for each application page could increase the resource demand and decision latency. Because classification is highly related to the number of previous read and write accesses to each page, they considered weighted access of page 'p' for the scheduling epoch 'i' as follows.

$$\text{Access}_i(p) = \alpha \times \text{writes}_i(p) + \beta \times \text{reads}_i(p) \tag{2.1}$$

And, depending on the placement choice, build a binary function 'MP(p)' that tells where 'p' was misplaced on scheduling epoch 'i'. When the number of previous accesses to a page exceeds a user-defined threshold, the page is considered hot. Here is how the profit function (p) is defined.

$$\text{Profit}(p) = \sum_{i=0}^{N} \text{Access}_i(p) \cdot \text{MP}_i(p) \tag{2.2}$$

If the profit exceeds a certain threshold, the page becomes critical. These weights are updated to contribute to the cost function.

$$\text{Loss}_i(p) = \text{real}_i(p) \cdot \log(\text{predicted}_i(p)) \cdot w_i(p) \tag{2.3}$$

Hence, by this, they achieved a performance gain of up to 65.5% and saved energy on an average of 20.2%.

2.5 Conclusion

In conclusion, in this chapter, we have discussed the prior works considered state-of-the-art in the domain of page migration and page placement. We have explained the different categories of work and the aspects considered in this field. More broadly, the Data migration approaches can be either using machine learning paradigms or may be caching or threshold-based, and the main aim is to find the data that has chances of getting frequent access so that it can be available in fast memory to improve the performance and reduce the overall access latency. This also tells in which direction the current trend is going, how it will change with modern technologies, and how we can use these technologies more effectively. Following these studies, we have taken some motivations and observations, which will be discussed in further sections, and define the related problem statement of our work.

Chapter 3

Proposed Methodology

This chapter introduces the Access Intensity and Row Buffer Locality-based Data Migration algorithm for hybrid memory systems, which is designed to address the challenges faced by existing algorithms.

3.1 Overview

Hybrid memory systems combine the strengths of two different memory technologies to achieve better performance than the individual one could offer. In such systems, data migration is the main factor for the improved performance. State-of-the-art techniques related to page migration are either based on the cache-like or threshold-like methodology where the decision will be made on the access of the page of far memory or the accumulated access counter of the page of far memory crosses the fixed threshold. Since Near Memory has a low latency, it's possible to increase the application's execution time by making the most of the access while the page is in NM. Accurately identifying and migrating the pages at the right time is essential to maximize the number of NM hits. Since the HBM size is limited, it might not be beneficial to migrate every page whose access count surpasses the threshold. Also, data migration is swapping the pages of NM and FM, i.e., migrating a page from NM to FM and FM to NM, which causes a large migration overhead.

Because the migration process is time-consuming and expensive, we cannot base our decisions only on access counts. Choosing the right candidates to get the most out of page migration is essential. As a result, unlike access counts, which take into account the behavior over time of memory pages, access intensity takes into account the behavior of the pages right now. Therefore, in order to enhance the performance of hybrid memory systems, it is preferable to choose the migration candidates based on the intensity of access. Along with that, we have incorporated the row buffer locality factor to improve our selection of the page for the migrations, and the following sections will have the details of the algorithm.

3.2 System Architecture

We have considered the hybrid memory system composed of High Bandwidth memory with an off-chip DDR4 DRAM, where both memories are used as the flat address space and serve the memory requirements. The HBM consists of one channel, and DDR4 consists of 7 channels, where each channel has a capacity of 1GB, so the total memory capacity in our setup is 8GB. Figure 3.1 displays the proposed system architecture of AI-RBL, where the required extra data structures and hardware are shown in the highlighted area.

The AI-RBL is implemented in the memory controller, referred to as the Hybrid memory controller, where all the decision logic and metadata structure are executed; this logic is implemented between the last level cache and the memories. It is responsible for maintaining and updating the counters, address translation, and data migration. Three types of counters have been used, which are associated with each page of the far memory. These are access counter, access intensity, and row misses; also, there is one pending queue that will store the address of the page along with their access intensity as the candidate for the migration purpose. At a fixed interval of time, the controller will check the pending queue and select a candidate that has the highest access intensity for the migration among the

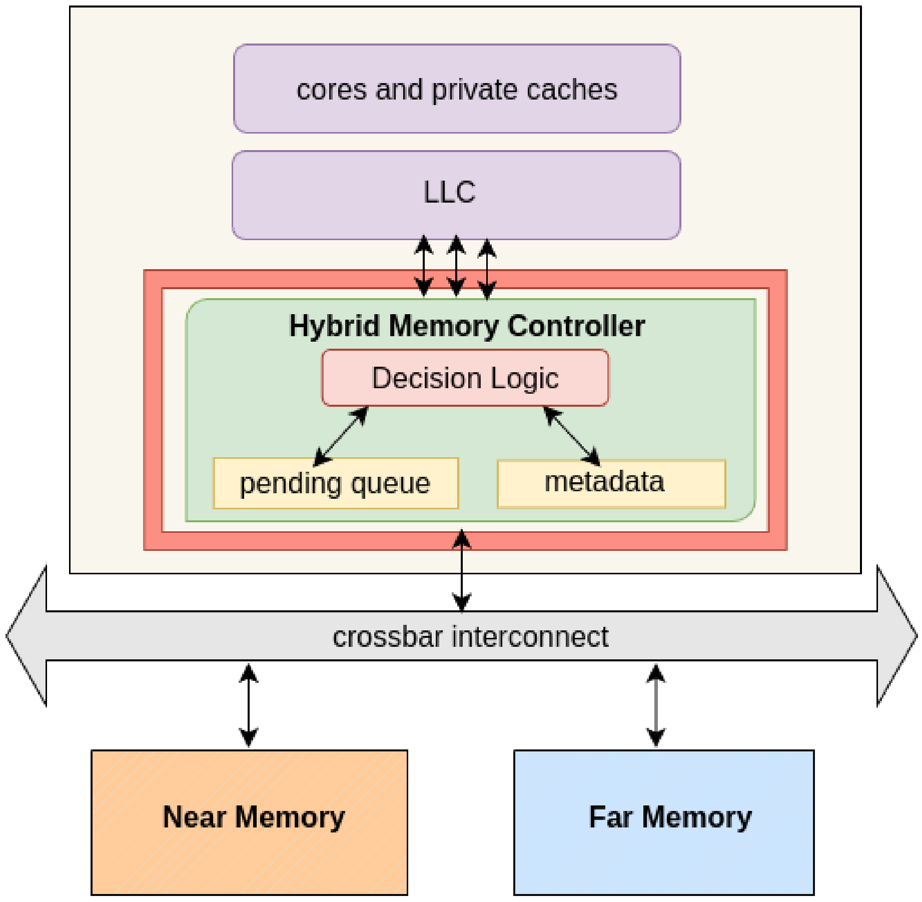

Fig. 3.1 System Architecture

possible candidates. As the capacity ratio of these memories is 1:7, then the distribution of the requests is also assumed to be in that ratio, which is 12.5% requests for NM and the remaining 87.5% for FM. We have also maintained the LRU(least recently used) list of pages for the NM to keep track of pages that are least accessed and swap them with the FM pages. With the assistance of other parts in the migration unit, the decision & control unit thus manages the entire page migration within the memory controller, from NM to FM and vice versa.

The hybrid memory controller comprises the following essential components:

1. **Decision Logic:** This core logic module is responsible for maintaining information integrity and executing migration processes based on predetermined conditions.

2. **Pending Queue:** As a repository, the pending queue houses a comprehensive list of pages earmarked as potential migration candidates. Each entry in this queue is associated with its respective page reference and access intensity metrics.

3. **MetaData:** This crucial unit encompasses a range of data structures essential for the algorithm's functionality. These structures include counters and flags meticulously designed to meet the specified requirements of the overall system design.

3.3 AI-RBL Migration Design

As the migration process is expensive, we need to focus on two important things: which candidate to select and how to select. So, to keep these things in consideration, we have come up with two terminologies and combined them to make them useful in the migration algorithm. Let's discuss them first, and later, we will dive deep into the algorithm.

3.3.1 Access Intensity

Access Intensity refers to the number of access counts per unit of time. The difference between the access counts and the access intensity is that the accumulated access counts may not reflect the current behavior of the memory accesses. In contrast, access intensity adapts the run-time behavior and illustrates how memory pages behave over time. It helps to identify the frequently accessed pages within a specific window of time, which is crucial for efficient memory management techniques. The equation to calculate access intensity is as follows:

$$AI = \frac{number\ of\ access}{time\ since\ the\ page\ was\ first\ accessed} \tag{3.1}$$

3.3.2 Row Buffer Locality

Every memory is made up of several banks, each of which is made up of rows and columns of two-dimensional memory cells. While each bank is able to function independently, all banks within a channel share the command, data, and address buses. The row buffer is an internal buffer present in every bank. The complete row containing the data is brought into the row buffer when data is accessed from a bank. As a result, the row buffer can provide data from the same row at a later time without requiring access to the array. We refer to this type of access as a row buffer hit. The contents of the row buffer must be written back to the row in the event that a subsequent access is made to data in a different row, and the contents of the new row must be brought into the row buffer. Row buffer miss (or row buffer conflict) is the term used to describe such an access. A row buffer miss has a significantly higher latency than a row buffer hit because the latency of a row buffer conflict/miss is often considerably higher in denser memories. Still, the latency of a row buffer hit is similar across memory types.

Row buffer locality is the percentage of row buffer hits among all memory accesses to a row. Because row buffer misses on low-locality pages are more common and are handled more quickly in the fast memory, we may anticipate that moving such pages to the fast memory will improve performance. On the other hand, since most accesses to pages with high row buffer locality hit the row buffer and a row buffer hit has a comparable latency in both the fast memory and the slow memory, we may anticipate that relocating a page with high row buffer locality will not significantly improve performance [17],[18]. To calculate such, we have maintained a row buffer miss counter, which will count the row buffer misses for that page using the flag associated with each request, i.e., row_hit; if the flag is false, we will increment the counter by 1.

Algorithm 1 AI-RBL based Page Migration

```
$Counter[]$: Array of access counts of pages
$RowMiss[]$: Array of row miss counts of pages
$AI[]$: Array of access intensity of pages
$time[]$: Array storing time since first access to pages
$MissTh$: Miss Threshold
$CP$: Candidate Page; $PQ$: Pending Queue

function AI-RBL
    $time[P]$ = time elapsed since page $P$ was first accessed
    for every write request coming to page $P$ do
        // increment access count and update access intensity
        $Counter[P]++$
        $AI[P] = \frac{Counter[P]}{time[P]}$
        // increment row miss count if row_hit flag is false
        if !row_hit($P$) then
            $RowMiss[P]++$
        end if
        if $RowMiss(P) \geq MissTh$ then
            // add $P$ to pending queue
            $PQ = PQ \cup \{P\}$
        end if
    end for
    for every interval boundary do
        sort $PQ$ based on access intensity
        $CP = P | P \in PQ \wedge AI[P] = \max$
        migratePage(CP)
    end for
end function

function migratePage($CP$)
    $v_{HBM}$ = LRU(HBM)
    Migrate $v_{HBM}$ to DDR
    Migrate $CP$ to HBM
    // remove $CP$ from the pending queue
    $PQ = PQ \setminus \{CP\}$
end function
```

3.3.3 Algorithm

Algorithm 1 describes the proposed algorithm for page migration, which selects the page for migration based on row buffer locality and access intensity. Every time a request comes for the far memory page, it will increment the access counter and update the access intensity for that page. We also maintain a row miss counter if, for that request, the row_hit flag is set as false, then we will increment the row misses by 1. It continuously updates the counter (line 12), access intensity (line 13), and the row misses (line 16). It compares the number of row misses for that page against the fixed Miss Threshold (MissTh). If it is greater than MissTh, the page might be a possible candidate for the migration, so it moves to the pending queue along with its access intensity (lines 18-21). At the end of each regular interval, we will select a candidate from the pending queue that has the highest access intensity among all and migrate it to the near memory. For migration, we need to swap the pages from NM to FM and vice-versa, so to select a victim page from NM that can be replaced by the selected candidate for migration from FM, we maintained the LRU list for the NM pages, and it will be updated on every access to an FM page. The migratePage function (lines 30-36) will find the LRU page from FM and swap it with the candidate page from the pending queue; after that, the candidate page will be removed from the pending queue.

3.4 WorkFlow

Figure 3.2 describes the flow of the algorithm. The flow diagram illustrates the sequential steps in the proposed page migration algorithm, highlighting the key decision points and data flow. Each algorithm component is represented by a specific process or decision node, with arrows indicating the flow of control and data between them. As memory access requests are received, the algorithm monitors each request and the first task is to find the channel number and the page number of each incoming request into the controller, after

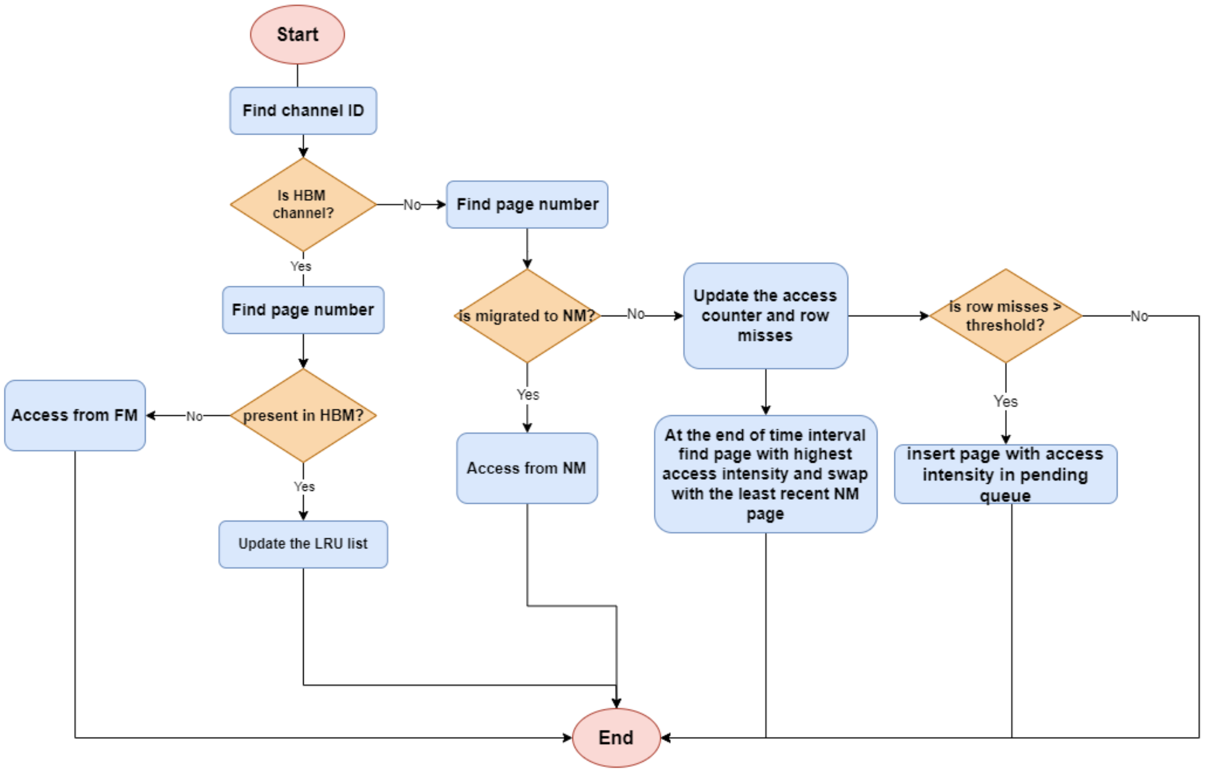
Start
Find channel ID
Is HBM channel?
No
Find page number
Yes
Find page number
is migrated to NM?
No
Update the access counter and row misses
is row misses > threshold?
No
Yes
insert page with access intensity in pending queue
Access from FM
No
present in HBM?
Yes
Update the LRU list
Yes
Access from NM
At the end of time interval find page with highest access intensity and swap with the least recent NM page
End

Fig. 3.2 Flow Diagram for Proposed Algorithm

which we will be able to identify if the request is for NM or FM by using the channel number, if the channel number is 1 i.e. the request is for NM else it would be for FM. For near memory requests, we have to find if it's present in HBM, if it is present then update the LRU list otherwise access it from the far memory (that would be the case where the page got swapped with the FM page). As for the far memory requests update the access counter, access intensity, and the row miss counts and check if it will become the possible candidate for the migration. At the end of the interval select one page with the highest access intensity to move into the NM, if the request is already present in the NM then access it from there.

3.5 Hardware Cost

The data structures we have taken for the implementation are 3 8-bit counters i.e. for access counts, access intensity, and row miss counts for each page. The total capacity of the memory in our design is 8GB with a page size is 256 bytes, so the total size required by these data structures would be 32 MB. The size of the pending queue would be approximately 1 KB. Therefore the storage overhead required additionally in our design is 32.0009 MB which is nearly 0.39% of the total memory capacity.

3.6 Observation

We have covered the shortcomings of the current method in this section, along with the features that set our algorithm separate. Firstly, RHPM uses the congruence group concept which means they are restricting the migrations to happen within the group. Still, we find that an all-to-all remapping strategy is more appealing since it expands the number of migration options by letting any page move between the two memories. Secondly, they are performing the migrations right after when the FM page hotness counter value is greater than the page residing in the NM so there may be a case that multiple groups are performing the migrations at the same time, whereas we have considered the time interval concept which selects one page at a time to migrate.

3.7 Conclusion

In conclusion, we proposed an algorithm for page migration that leverages row buffer locality and access intensity to optimize memory management in hybrid memory systems. The algorithm is designed to optimally select pages for migration between near memory (NM) and far memory (FM) based on their current access patterns during the interval and row miss rates, to minimize memory access latency and improve overall system performance. The algorithm effectively identifies pages with high potential for migration based on their access intensity and row miss rates, leading to a reduction in overall memory access latency. By leveraging row buffer locality, the algorithm ensures that migrated pages exhibit improved access patterns in the target memory tier, further enhancing system performance. The use of a Least Recently Used (LRU) list for victim page selection in the NM ensures that pages selected for migration do not disrupt the overall memory hierarchy, minimizing the impact on system stability.

Chapter 4

Evaluation & Results

In this chapter, we discussed the system configuration or setup, the evaluation of the state-of-the-art Data Migration algorithm, and our proposed algorithm AI-RBL with the help of many factors.

4.1 Experimental Setup

For the experiment purpose, we have used the Simulation mode of the GEM5 simulator [19],[20],[21]. We have taken the configuration file of memory components, DDR4 as the conventional off-chip DRAM, and HBM2 as the 3D-stacked DRAM memory in the hybrid memory configuration. There are eight channels, out of which one channel is for stacked DRAM, and the other seven are used for conventional DRAM. First, Let's see the configuration parameters for the Far Memory, a conventional DRAM. The clock frequency is 3200 MHz, and the CPU frequency is 3000 MHz; it has 8 banks per rank, 2 ranks per channel, and 7 channels, and the bus width is 64 bits. The timing parameters are tCAS is 22 cycles, tRCD is 22 cycles, tRP is 22 cycles, tRAS is 38 cycles, and tBURST is 8. The Energy for RD/WR + I/0 per bit is 33 pJ. Next, the configuration for the 3D-stacked DRAM is the clock frequency is 2 GHz, and the CPU frequency is 3000 MHz; it has 8 banks per rank, 1 rank per channel, and 1 channel, and the bus width is 128 bits. The timing parameters are tCAS, which is 7 cycles; tRCD, which is 7 cycles; tRP, which is 7 cycles; tRAS, which is 16 cycles; and tBURST, which is 4. The Energy for RD/WR + I/0 per bit is 6.4 pJ. Other details are mentioned in Table 4.1.

Table 4.1 System Configuration

System Components	**Parameters**
Processor	x86
Cores	8
CPU_Type	TimingSimpleCPU
L1 (I/D)	32 KB, 2-way associativity
L2	2 MB, 8-way associativity
Near Memory	2 GHz HBM2, tCAS-tRCD-tRP-tRAS: 7-7-7-16, RD/WR+IO energy: 6.4 pJ
Far Memory	3200 MHz-DDR3, tCAS-tRCD-tRP-tRAS: 22-22-22-38, RD/WR+IO energy: 33 pJ
Interval Size	5μs
Page Size	256 Bytes

For simulation, we used the benchmark SPEC2017 [22],[23] suite, from which we have taken 6 applications mix of high and low MPKI workloads for evaluation purposes listed in Table 4.2. These are *lbm, sjeng, cactus, xalancbmk, leela parest*, all the requests are proportionally distributed and randomly assigned to both Far and Near Memory. We have run all the benchmarks, providing all the parameters with 1 Billion instructions and 250 million warmups.

Table 4.2 Benchmarks for SPEC CPU2017

Workload	**Benchmark**
High Workloads	lbm sjeng cactus
Low Workloads	parest leela xalancbmk

Table 4.3 represents the total number of requests distributed over the memories for different benchmarks of the 1 Billion instructions.

Table 4.3 Requests distribution of 1 Billion instructions

Benchmark	**Near Memory**	**% of req in NM**	**Far Memory**	**% of req in FM**
parest	3047	12.5	21365	87.5
leela	7550	12.5	52417	87.5
xalancbmk	85868	11.9	631265	88.1
cactus	288626	7.8	3407760	92.2
sjeng	5706592	13.2	37640454	86.8
lbm	6512865	12.5	45486721	87.5

We compare our work with baseline configuration and the state-of-the-art technique RHPM, which was explained in section 2.3.2 of chapter 2 (Literature review).

1. **Baseline:** Baseline has the same capacity without having NM, i.e., only Far Memory with total capacity same as our design.

2. **RHPM:** This design is worked on congruence group flexibility where both memories are divided into groups and the migration is restricted within these groups, hotness counter is maintained for each page, it is incremented upon access and other decremented in the same group and if the currently accessed page of FM has greater than the page present in NM then migration will be triggered, and other page counters will be reset to 0 because of reset rule.

3. **AI-RBL:** It is the proposed policy for migration based on row buffer locality and access intensity. Low-row buffer locality pages will be sent to the pending queue. Among the pages present in the pending queue, the highest access intensity will be selected for migration during the interval.

4.2 Timing Parameters

According to the parameter values we took for the experiment, the read and write cycles can be calculated as follows.

Read Cycles:

$$\mathrm{t}_{read} = \mathrm{t}_{CAS} + \mathrm{t}_{RCD} + \mathrm{t}_{BURST} \tag{4.1}$$

Write Cycles:

$$\mathrm{t}_{write} = \mathrm{t}_{CWD} + \mathrm{t}_{RCD} + \mathrm{t}_{BURST} + \mathrm{t}_{WR} \tag{4.2}$$

After migration, the FM page will have latency as NM, and the NM page will have latency as FM. Let's assume P_NM and P_FM are the pages of NM and FM, respectively, which are going to be swapped, so P_NM will go to FM and P_FM will go to NM. Therefore, the new cycle calculation for these pages after migration will be calculated below.

New Read Cycles:

$$\begin{gathered}\mathrm{t}_{NMread} = \mathrm{t}_{NMread} + (\mathrm{t}_{FMread} - \mathrm{t}_{NMread}) \\ \mathrm{t}_{FMread} = \mathrm{t}_{FMread} - (\mathrm{t}_{FMread} - \mathrm{t}_{NMread})\end{gathered} \tag{4.3}$$

New Write Cycles:

$$\begin{gathered}\mathrm{t}_{NMwrite} = \mathrm{t}_{NMwrite} + (\mathrm{t}_{FMwrite} - \mathrm{t}_{NMwrite}) \\ \mathrm{t}_{FMwrite} = \mathrm{t}_{FMwrite} - (\mathrm{t}_{FMwrite} - \mathrm{t}_{NMwrite})\end{gathered} \tag{4.4}$$

4.3 Results

In this section, we have discussed all the results evaluated with our proposed design, the baseline, and the state-of-the-art technique using the 6 applications of the SPEC CPU2017 benchmark suite, each with 1 Billion instructions.

4.3.1 IPC

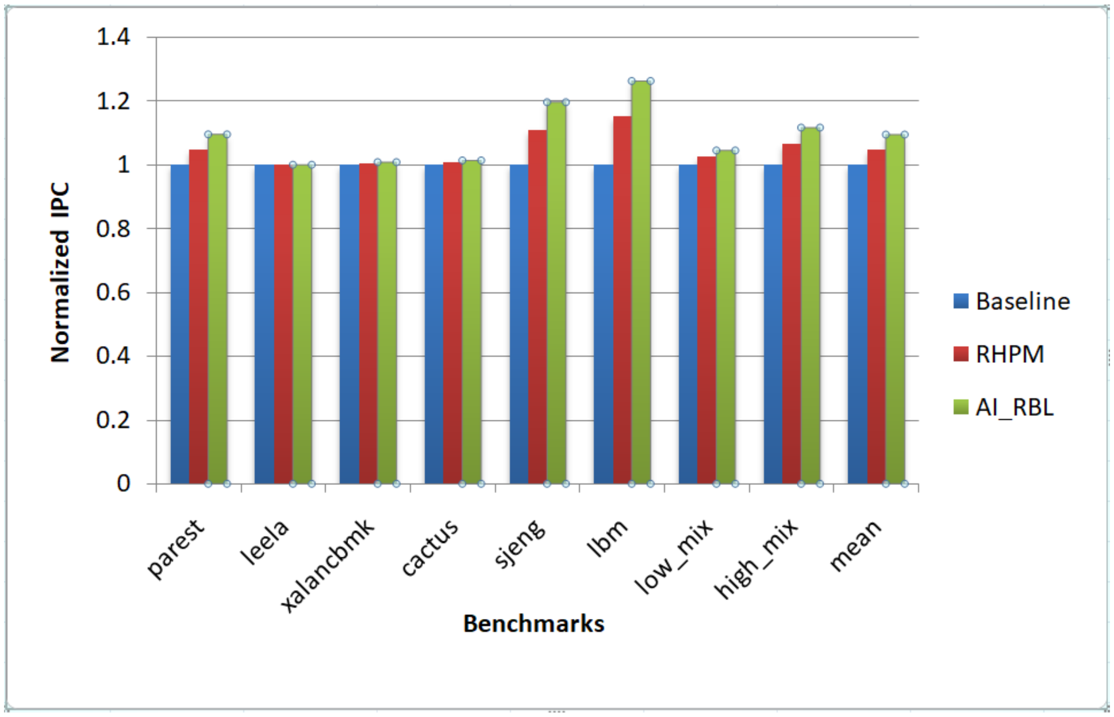

Fig. 4.1 Normalized Speedup (higher is better)

The IPC is Instructions Per Cycles, defined as the reciprocal of the CPI (simulation time). It is used to quantify the performance of the system, also referred to as speedup. Thus, improved performance is associated with increased speedup. Figure 4.1 shows the IPC values for RHPM and AI-RBL normalized with the baseline for different benchmarks. On average, our proposed design AI-RBL improves performance by 10.75% over baseline, whereas RHPM only improves by 6.15% over baseline, and our design improves performance by 5% over RHPM. Also, we have observed that highly access-intensive workloads like sjeng and lbm give 16.4% and 20.8% improvement over baseline, respectively, while RHPM improves by 9.7% and 13.1%, respectively. And for low access-intensive workloads like leela and xalancbmk do get benefit from migrations.

4.3.2 Memory Service Time

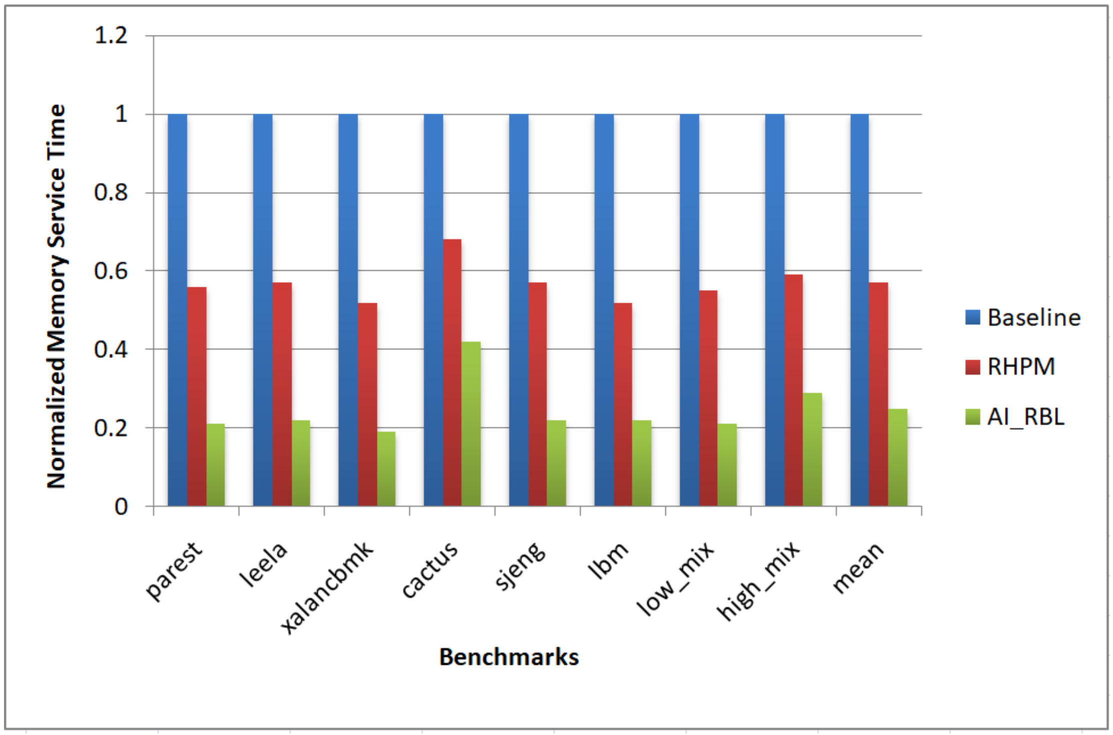

Fig. 4.2 Normalized Memory Service Time (Lower is better)

Memory access time shows how effectively memory works in terms of the speed at which a request may be fulfilled. Faster memory access times lead to quicker data retrieval by the CPU, which translates to faster program execution and overall system responsiveness. Here, the lower the service time, the better the performance. Figure 4.2 depicts the service time values normalized to the Baseline. We have improved the memory service time of our design AI-RBL by 74.6% on an average over baseline, while RHPM improves by 42.5% over baseline. In RHPM, the number of migrations is greater than our design, and the migrated candidate doesn't take the benefit of migration and overpowers the migration overhead. Overall, memory service time (access time) is a critical aspect of system performance and identifies the effects of different memory technologies.

4.3.3 Energy

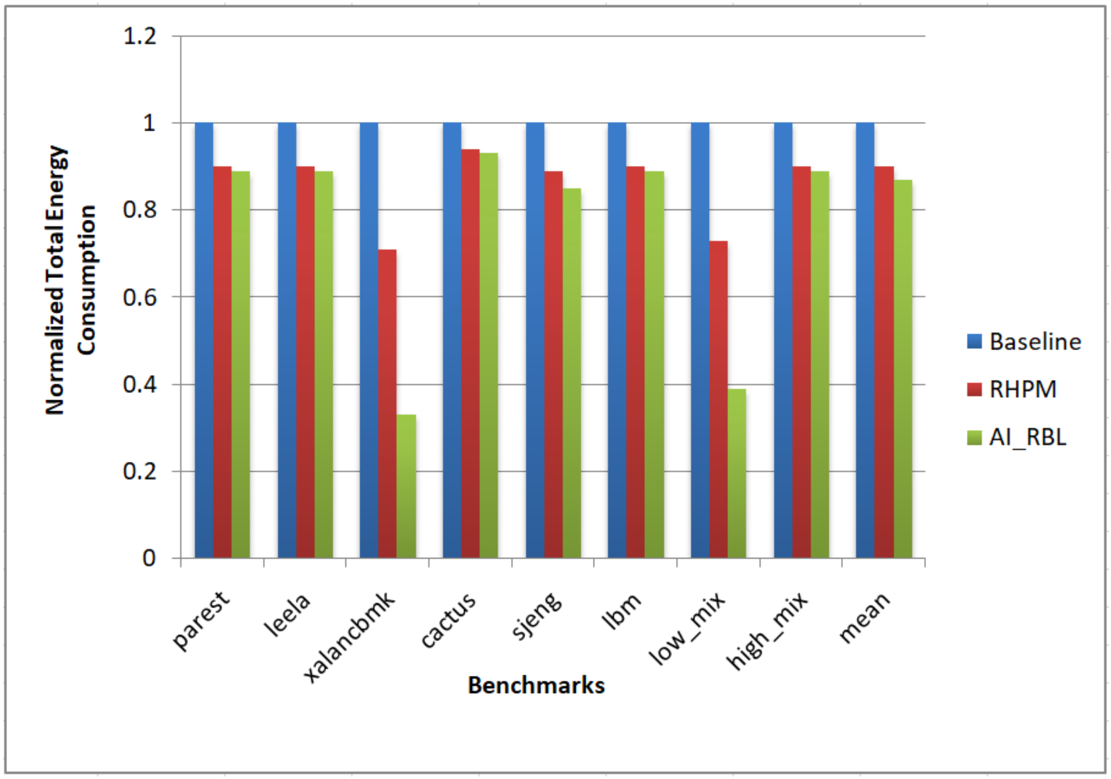

Fig. 4.3 Normalized Total Energy Consumption (Lower is better)

The total energy consumption of a system refers to the total amount of energy a system uses over a specific period. This can be measured in various units depending on the context, such as watts (W) for continuous power draw, watt-hours (Wh) for energy used over time, or joules (J) for total energy. The total energy consumption of hybrid memory includes the energy required for migration as well as routine read/write requests. The read/write energy can be calculated using the equations given below.

$$\text{Read Energy} = \text{Read}_{count} * \text{Read}_{energy\_per\_bit} * \text{Read}_{bit} \tag{4.5}$$

$$\text{Write Energy} = \text{Write}_{count} * \text{Write}_{energy\_per\_bit} * \text{Write}_{bit} \tag{4.6}$$

The total energy consumption can be calculated with the equation given below:

$$\begin{aligned}\text{TotalEnergy} =& \#\text{ReadsNM} \times \text{ReadEnergyNM} + \#\text{WritesNM} \times \text{WriteEnergyNM} \\ &+ \#\text{ReadsFM} \times \text{ReadEnergyFM} + \#\text{WritesFM} \times \text{WriteEnergyFM} \\ &+ \#\text{Migration} \times \text{ReadEnergyNM} + \#\text{Migration} \times \text{WriteEnergyFM} \\ &+ \#\text{Migration} \times \text{ReadEnergyFM} + \#\text{Migration} \times \text{WriteEnergyNM}\end{aligned} \tag{4.7}$$

In figure 4.3, we have shown the energy comparison of AI-RBL with baseline and RHPM. AI-RBL improves energy by 19.8% over baseline, whereas RHPM improves by 12.06% over baseline. The difference in energies between AI-RBL and RHPM can only differ by the number of migrations.

Table 4.4 Percentage of Performance Improvement

Metrics	**RHPM**	**AI-RBL**
IPC	6.15	10.75
Memory	42.5	74.6
Energy	12.06	19.8

4.4 Migration Overhead

The page size taken in our design is 256 Bytes. For near memory, the bus width is 128 bits, and the burst length is 4, so one read/write request has 64 bytes. For far memory, the bus width is 64 bits, and the burst length is 8, so again, one read/write request has 64 bytes. Hence, the number of times required to read/write an NM or FM page is 256/64 = 4. Let's denote this as n. For migration, we require swapping pages from NM to FM and vice-versa, i.e., reading an NM page and writing to FM and vice-versa. which is twice the number of times it is required to read multiplied by read cycles plus write cycles. So, the total number of cycles required extra for migration is given below.

Migration time:

$$\mathrm{t}_{MO} = n * \mathrm{t}_{FMread} + n * \mathrm{t}_{NMwrite} + n * \mathrm{t}_{NMread} + n * \mathrm{t}_{FMwrite} \tag{4.8}$$

Table 4.5 Number of migrations of both designs

Benchmarks	**RHPM**	**AI-RBL**	**% Reduction**
parest	379	67	82.3
leela	483	55	88.6
xalancbmk	11248	975	91.
cactus	17574	4557	74
sjeng	81696	7537	90.7
lbm	52250	4749	90.9

4.5 Conclusion

In this chapter, we presented a comprehensive analysis of our proposed hybrid memory architecture, termed AI-RBL, in comparison with the baseline configuration and the state-of-the-art technique, RHPM. Our investigation focused on evaluating performance metrics such as IPC, memory service time, energy consumption, and migration overhead across various workloads from the SPEC CPU2017 benchmark suite. Through our simulations and analyses, we observed notable improvements achieved by our AI-RBL design over both the baseline and RHPM. Specifically, in terms of IPC, our proposed design demonstrated an average improvement of 10.75% over the baseline, outperforming RHPM by 5%. Furthermore, AI-RBL exhibited superior memory service time, with an average improvement of 74.6% over the baseline, surpassing RHPM by 32.1%. This improvement underscores the effectiveness of our approach in enhancing memory access efficiency.

Moreover, our investigation into energy consumption revealed promising results, with AI-RBL reducing total energy consumption by 19.8% compared to the baseline, outperforming RHPM by 7.74%. These findings highlight the energy-efficient nature of our proposed design, which can contribute to overall system sustainability and cost-effectiveness. Additionally, our analysis of migration overhead sheds light on the practical implications of our hybrid memory architecture. By carefully considering page size, bus width, and burst length, we minimized migration time, ensuring efficient data transfer between near and far memory components.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Lastly, to conclude, in the current work of this thesis, we understand the system paradigm and the methodology used, which is to aggregate 3D-stacked DRAM and off-chip DRAM to establish a Hybrid Memory System and focus on Data Migration or Page Migration in this paradigm. Following with a review of related literature, which was enlightened in this study, provides an understanding of this evolving field. We provided a thorough analysis of our suggested hybrid memory architecture, known as AI-RBL, contrasting it with the industry standard configuration and the most advanced method, RHPM. We found that our AI-RBL design significantly outperformed the baseline and RHPM based on our simulations and analyses. More specifically, our suggested design outperformed RHPM by 5% in terms of IPC, with an average improvement of 10.75% above the baseline. Furthermore, our migration overhead analysis clarified the usefulness of our hybrid memory design. To ensure effective data transmission between near and far memory components, we decreased migration time by carefully considering page size, bus width, and burst length.

5.2 Future Work

The following points outline the key considerations for embarking on future work initiatives:

1. **Thresholding Sensitivity Analysis:** Currently we have implemented the algorithm using a fixed threshold, we will analyze to evaluate the effect of threshold variation on system performance metrics and identify the optimal value. Also, we will try to identify the thresholds per benchmark as the requirements of different workloads are different so the threshold should be adaptive.

2. **Analysis of Migration Granularity:** Migration granularity is the size or scale at which the data is migrated or transferred between the different memory modules in the hybrid memory system. We have already discussed how important is to consider the granularity, as the small size will not take the benefit of spatial locality, and the large size leads to over-fetching. So varying different values will identify the best size while minimizing the system overheads.

3. **Understanding workload characterization:** In our evaluation we have considered the summation of read and write access counts for the criteria to be considered for finding victim page selection. Further, we will try to identify which workloads are more read-sensitive or write-sensitive, and balance read and write-intensive workloads by assigning appropriate priorities to ensure optimal system performance and resource utilization in hybrid memory systems

References

[1] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens, "Challenges and future directions for the scaling of dynamic random-access memory (dram)," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 187–212, 2002.

[2] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "Hbm (high bandwidth memory) dram technology and architecture," in *2017 IEEE International Memory Workshop (IMW)*, pp. 1–4, IEEE, 2017.

[3] F. Meng, Y. Xue, and C. Yang, "Power-and endurance-aware neural network training in nvm-based platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2709–2719, 2018.

[4] Z. Duan, H. Liu, X. Liao, H. Jin, W. Jiang, and Y. Zhang, "Hinuma: Numa-aware data placement and migration in hybrid memory systems," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pp. 367–375, IEEE, 2019.

[5] 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 *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 235–246, IEEE, 2012.

[6] J. B. Kotra, H. Zhang, A. R. Alameldeen, C. Wilkerson, and M. T. Kandemir, "Chameleon: A dynamically reconfigurable heterogeneous memory system," in *2018*

51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 533–545, IEEE, 2018.

[7] Z. Peng, D. Feng, J. Chen, J. Hu, and C. Huang, "Agdm: An adaptive granularity data migration strategy for hybrid memory systems," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2023.

[8] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Hybrid2: Combining caching and migration in hybrid memory systems," in *2020 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 649–662, IEEE, 2020.

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

[10] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Llc-guided data migration in hybrid memory systems," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 932–942, IEEE, 2019.

[11] A. Prodromou, M. Meswani, N. Jayasena, G. Loh, and D. M. Tullsen, "Mempod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 433–444, IEEE, 2017.

[12] J. H. Ryoo, M. R. Meswani, A. Prodromou, and L. K. John, "Silc-fm: Subblocked interleaved cache-like flat memory organization," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 349–360, IEEE, 2017.

[13] Z. Peng, D. Feng, J. Chen, J. Hu, and C. Huang, “Rhpm: Using relative hotness to guide page migration for hybrid memory systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[14] C. 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 *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–12, IEEE, 2014.

[15] T. D. Doudali, S. Blagodurov, A. Vishnu, S. Gurumurthi, and A. Gavrilovska, “Kleio: A hybrid memory page scheduler with machine intelligence,” in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 37–48, 2019.

[16] M. Katsaragakis, K. Stavrakakis, D. Masouros, L. Papadopoulos, and D. Soudris, “Adjacent lstm-based page scheduling for hybrid dram/nvm memory systems,” in *14th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and 12th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2023)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[17] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, “Utility-based hybrid memory management,” in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 152–165, 2017.

[18] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, “Row buffer locality-aware data placement in hybrid memories,” *SAFARI Technical Report*, vol. 5, 2011.

[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[20] M. Poremba and Y. Xie, "Nvmain: An architectural-level main memory simulator for emerging non-volatile memories," in *2012 IEEE Computer Society Annual Symposium on VLSI*, pp. 392–397, IEEE, 2012.

[21] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, 2015.

[22] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[23] A. Limaye and T. Adegbija, "A workload characterization of the spec cpu2017 benchmark suite," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 149–158, 2018.